



Purpose-built hardware & NixOS

Una nuova prospettiva al self hosting



POLITECNICO OPEN
unix LABS

Lorenzo Andreasi
lolloandr@poul.org

Leonardo Salvini
jep@poul.org

Speaker notes

Domande.

Cos'è il self hosting?

Self-hosting is the practice of running and maintaining a website or service using a private web server, instead of using a service outside of the administrator's own control. Self-hosting allows users to have more control over their data, privacy, and computing infrastructure, as well as potentially saving costs and improving skills.



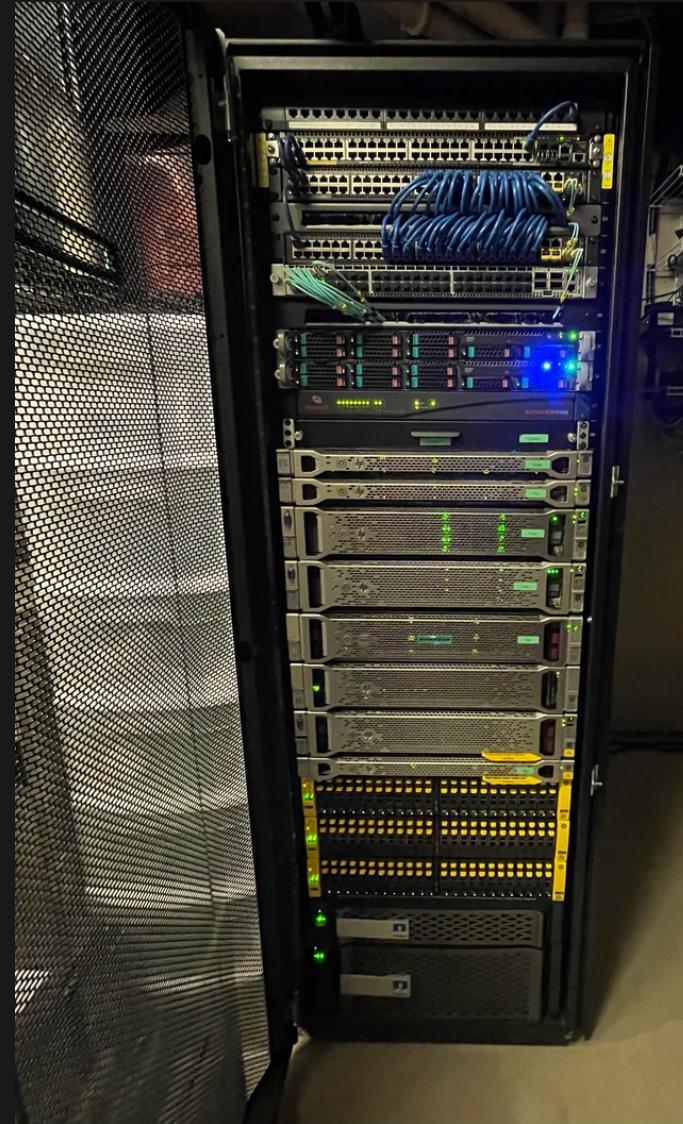
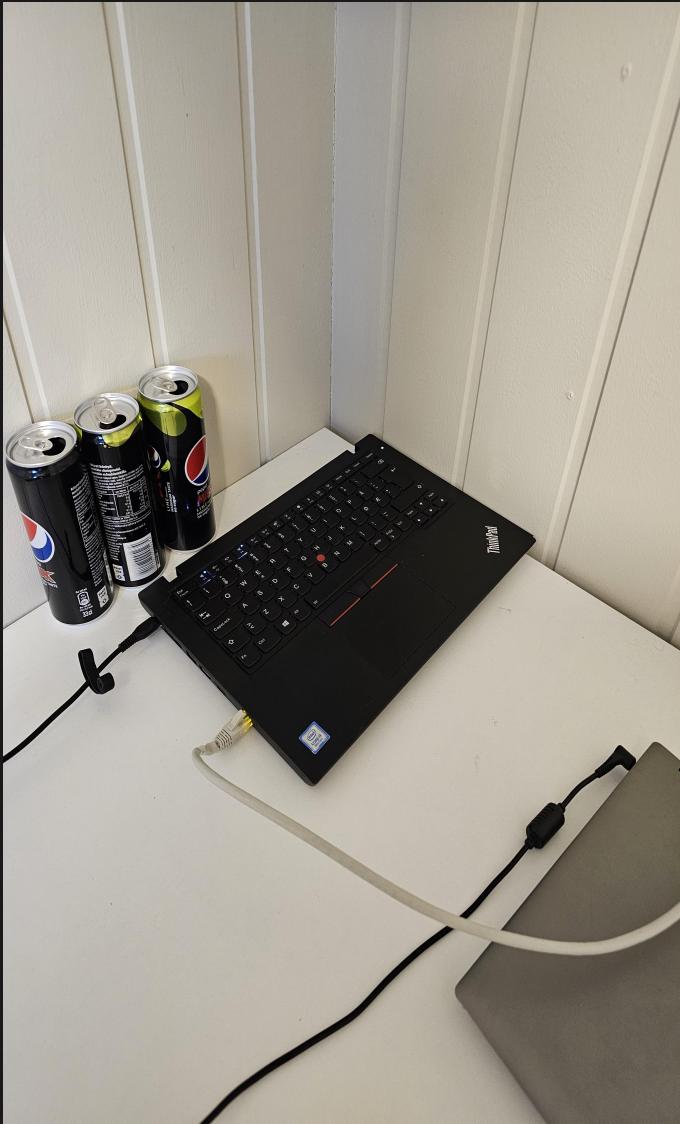
Al di là dell'utilità il vero vantaggio è la possibilità di **sperimentare**
ed **imparare**

Lista della spesa

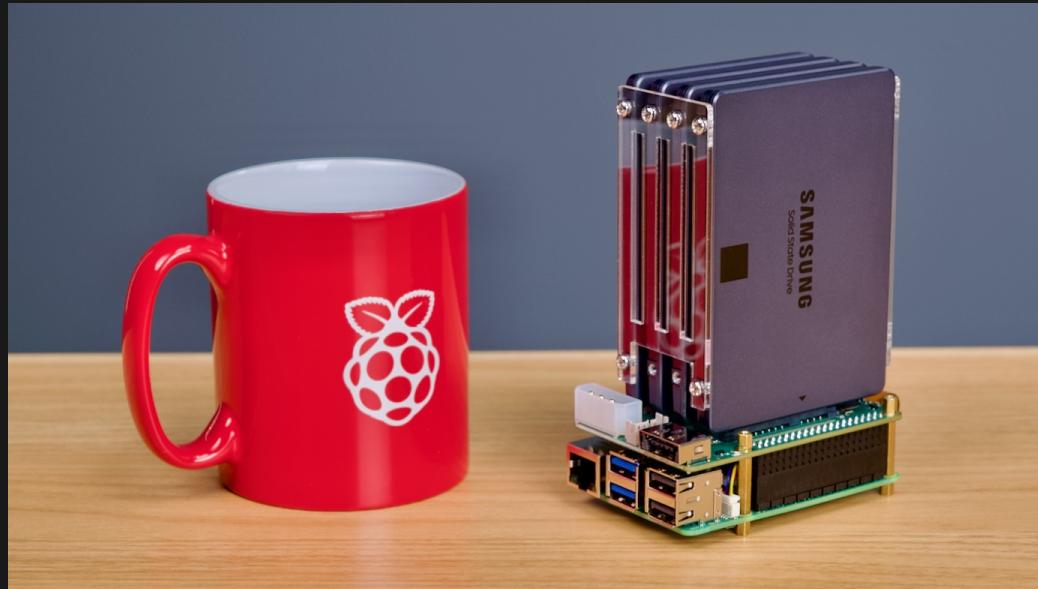
Cosa ci serve per iniziare a selfhostare:

- un computer
- una connessione ad internet
- un po' di tempo e **voglia di imparare**

La definizione di **computer** in questo contesto è molto ampia:



Se non si hanno molte pretese una soluzione potrebbero essere i
single board computer



Raspberry Pi 5 con HAT SATA, dal blog di  Jeff Geerling

Pro:

- Possiamo usare il RP5 che abbiamo a casa
- Form factor **compatto** e supporto fino a 4 dischi
- **Costo ridotto** (sotto i 200 euro dischi esclusi)

Contro:

- Il bus PCIE del RP5 è **gen 3** (non esistono porte SATA)
- Il Raspberry ha generalmente poca RAM ed è **poco prestante**

È davvero la soluzione migliore?

Speaker notes

Bottleneck dato sia dalla cpu, sia dalla rete che sia dal bus PCiE

La vera soluzione

Esiste hardware studiato appositamente per il self-hosting.

Per esempio: **Zimaboard** e **Zimablade**



	Zimaboard	Zimablade
CPU	N3350/N3450	N3350/N3450
RAM	2/4/8 GB	SODIMM Slot fino a 16GB DDR3L
PCIe	1xPCIe 2.0 x4	1xPCIe 2.0 x4
Ethernet	2x GbE LAN	1x GbE LAN
SATA	2x SATA 6.0 Gb/s	2x SATA 6.0 Gb/s
Prezzo	89/159/199 USD	69/119 USD

Vantaggi

Ereditano tutti i vantaggi dei SBC ed in più:

- CPU x86
- Espandibilità ed IO
- Porte ethernet

Tuttavia, questo tipo di hardware **consuma generalmente di più**

Speaker notes

TDP 6W

Alternative



HP MicroServer Gen8, Zimacube

Pro:

- Maggiore possibilità di **espansione**
- Generalmente supporto per un numero maggiore di **dischi**
- Generalmente hardware più **performante**

Contro:

- Più **ingombranti**
- Maggior consumo **energetico**
- Più **rumorosi**
- Generalmente più **costosi**

Software: che OS installare?

3 categorie principali:

- Distro per NAS: **openmediavault**, **truenas** e simili
- Distro pensate per il self-hosting: **CasaOS**
- Distro Linux “base” da personalizzare: **Ubuntu Server**, **Arch Linux**,
Fedora Server

Obiettivi

- Impostare un **IP statico** al nostro server, **DDNS**, caricare **chiavi SSH** e configurare una **VPN**
- Installare un **firewall**, configurarlo ed installare **fail2ban**
- Installare e configurare **docker/podman** (o simili)

Tanti file di configurazione diversi, salvati in diversi path. Difficile tenere traccia dei cambiamenti e fare backup di tutto.

Se solo esistesse una distribuzione che ci permetta di definire **esattamente** come vogliamo configurare il nostro OS in **un** file che può essere portato da una macchina all'altra per rendere il sistema **completamente riproducibile**.

Sarebbe anche davvero interessante se la stessa distro avesse un package manager **potente** e addirittura ci permetta di  **creare una ISO** con la nostra configurazione personalizzata.

NixOS



Caratteristiche

- Distro basata su il package manager **nix**
- **Fixed release** ma anche **Rolling release**
- **Stabile**

Speaker notes

Rilasciato nel 2003, tesi di dottorato nel 2006

Vantaggi

- Completamente **dichiarativo**
- Configurazione **centralizzata**
- **Rollbacks** semplici
- Facilità nella gestione dei servizi
- Maggiore **sicurezza e controllo** sulle dipendenze
- Creazione di **ISO personalizzate** in modo semplice
- Deploy da  remoto

Speaker notes

Dichiarativo VS imperativo

Linking delle dipendenze

Svantaggi

- Il rebuild del sistema richiede un **po' di tempo**
- La documentazione su alcuni aspetti è **carente**
- Funzionamento **diverso** da altre distro

Speaker notes

Non esiste /bin, filesystem non FHS tutto in /nix/store + eccezione per sh

NixOS for dummies

nix-shell

Tool molto comodo che permette di installare **momentaneamente** un pacchetto. Il pacchetto esiste solo nella shell in cui si è eseguito nix-shell

```
$ nix-shell -p git
```

Installa momentaneamente git nella shell

File(s) di configurazione

Durante l'installazione sono generati due file:

- **configuration.nix**: contiene alcune impostazioni di default
- **hardware-configuration.nix**: contiene la configurazione fisica del sistema, viene importato da configuration.nix, è autogenerato

Speaker notes

file in /etc/nixos

configuration.nix

```
{ config, pkgs, ... }:

{
  require = [
    ./hardware-configuration.nix
  ];

  boot.loader.systemdboot.enable = true;

  hardware.pulseaudio.enable = true;

  networking.hostName = "nixisgood";
  networking.networkmanager.enable = true;

  time.timeZone = "Europe/Rome";
```

Per avere una configurazione **più ordinata** possiamo creare più file.

Ad esempio:

Creiamo un file `pkgs.nix`

```
{pkgs, ...}: {  
    # Questo è un commento  
}
```

Dopo di che in `configuration.nix`:

```
#...  
{  
imports = [  
    # Include the results of the hardware scan.  
    ./hardware-configuration.nix  
    ./pkgs.nix  
];  
}
```

nixos-rebuild

Il comando permette di applicare la configurazione definita in configuration.nix al sistema.

```
# nixos-rebuild switch
```

Builda il sistema ed applica la configurazione

Speaker notes

Derivation (derivazione) nixos-rebuild switch --Rollback nixos-rebuilt test nixos-rebuild build-vm

Iniziamo a configurare il nostro server...

Configurare un IP statico

```
networking = {  
    hostName = "nixos";  
    domain = "example.com";  
    dhcpcd.enable = false;  
    interfaces.enp2s1.ipv4.addresses = [ {  
        address = "192.168.1.2";  
        prefixLength = 24;  
    } ];  
    defaultGateway = "192.168.1.1";  
    nameservers = [ "1.1.1.1" "8.8.8.8" ];  
};
```

Speaker notes

IP statico, SSH, vpn, firewall, fail2ban

SSH

Attiviamo il servizio SSH:

```
services.openssh = {  
    enable = true;  
    ports = [ 22 ];  
    settings = {  
        PasswordAuthentication = false;  
        PermitRootLogin = "no";  
    };  
};
```

Aggiungiamo le chiavi pubbliche per l'autenticazione:

```
users.users."user".openssh.authorizedKeys.keys = [  
    "ssh-rsa AAAAB3Nz....60WM= user"  
];  
# Oppure possiamo inserire le chiavi in un file:  
users.users."user".openssh.authorizedKeys.keyFiles = [  
    "/etc/nixos/ssh/authorized_keys"  
];
```

Speaker notes

Porta aperta in automatico

Server VPN con Wireguard

Ci sono diversi modi per configurare wireguard, in questo esempio useremo **wg-quick**

```
networking.wg-quick.interfaces = {
    wg0 = { # nome dell'interfaccia wireguard
        address = [ "10.0.0.1/24" ];
        privateKeyFile = "/root/wireguard-keys/privatekey";
        peers = [
            { # peer0
                publicKey = "{client public key}";
                presharedKeyFile = "/root/wireguard-keys/preshared_from_peer0_key";
                allowedIPs = [ "10.0.0.2/32" ];
            }
            # Altri peer...
        ];
    };
};
```

Per semplicità in questo esempio è stato omesso il comando di postUp e preDown

Una volta buildato il sistema il server wireguard sarà visto dal sistema come un **servizio systemd**

Usando i classici comandi di `systemctl` possiamo gestire il servizio appena creato col nome `wg-quick-wg0.service`

Il procedimento per configurare un **client** wireguard è molto simile.

Firewall

NixOS usa come firewall di default **iptables**

```
networking.firewall = {  
    enable = true;  
    allowedTCPPorts = [ 80 443 22 ];  
    allowedUDPPortRanges = [  
        { from = 4000; to = 4007; }  
        { from = 51810; to = 51830; }  
    ];  
};
```

Sono disponibili molte altre opzioni, è possibile vederle tutte sul sito  search.nixos.org cercando `networking.firewall`

Scripting di demoni systemd

Nella configurazione è possibile scriptare demoni di systemd.
Questo è uno strumento molto potente che permette di avere un
estremo controllo sul sistema.

Un piccolo esempio:

```
systemd.services.smart-test = {
    path = [
        pkgs.busybox
        pkgs.smartmontools
    ];
    after = ["network.target"];
    description = "Smartd monitor";
    serviceConfig = {
        Type = "oneshot";
        Restart = "on-failure";
        RestartSec = 30;
        ExecStart = "${pkgs.bash}/bin/bash -e /etc/bot/smart-test.sh";
    };
};
```

Speaker notes

Linking dipendenze e path pkgs.bash

Tramite un servizio systemd possiamo anche aggiornare un **DDNS**:

```
systemd.services = {
    dynamic-dns-updater = {
        path = [
            pkgs.curl
        ];
        script = "curl https://www.duckdns.org/update?domains=exampledomain&token=token";
        startAt = "hourly";
    };
};
```

Fail2ban

Di base per abilitare Fail2Ban basta una singola riga nella configurazione:

```
services.fail2ban.enable = true;
```

In questo modo si abilita anche una jail pre-configurata per SSH.

È possibile anche avere configurazioni più complicate:

```
services.fail2ban = {
    enable = true;
    # Ban IP after 5 failures
    maxretry = 5;
    ignoreIP = [
        # Whitelist some subnets
        "10.0.0.0/8" "172.16.0.0/12" "192.168.0.0/16"
        "8.8.8.8" # whitelist a specific IP
        "nixos.wiki" # resolve the IP via DNS
    ];
    bantime = "24h"; # Ban IPs for one day on the first ban
    bantime-increment = {
        enable = true; # Enable increment of bantime after each violation
        formula = "ban.Time * math.exp(float(ban.Count+1)*banFactor)/math.exp(1*banFactor)";
        multipliers = "1 2 4 8 16 32 64";
        maxtime = "168h"; # Do not ban for more than 1 week
    }
}
```

Sono disponibili vari esempi su  nixos.wiki

Installare docker

Si installa e abilita con una riga:

```
virtualisation.docker.enable = true;
```

È anche possibile usare la modalità rootless:

```
virtualisation.docker.rootless = {  
    enable = true;  
    setSocketVariable = true;  
};
```

Docker compose con nix

I container docker possono essere scriptati con nix

```
virtualisation.oci-containers = {
  backend = "docker";
  containers = {
    foo = {
      volumes = [
        "volume_name:/path/inside/container"
        "/path/on/host:/path/inside/container"
      ];
      ports = [
        "8080:9000"
      ]
      extraOptions = [
        "--network=host"
      ]
      # ...
    };
  };
}
```

Tuttavia

- Non si ha controllo fatto bene sulle docker networks
- Poca portabilità
- Poca documentazione

Soluzione

Metodi tradizionali

- docker-compose
- portainer

In questo caso NixOS si comporta come una **qualsiasi altra distribuzione**

Riassumendo

NixOS permette di:

- **Centralizzare** le configurazioni
- **Gestire** i servizi in modo più intuitivo
- Effettuare il **backup** delle config in modo semplice
- **Replicare** il sistema appena configurato

Effettivamente va a risolvere alcuni dei problemi che si incontrano nel self-hosting ed in ogni ambiente di produzione.

Per saperne di più:

-  wiki.nixos.org
-  nixos.wiki (wiki non ufficiale)
-  [Vimjoyer](https://www.youtube.com/user/Vimjoyer) (su youtube)
-  discourse.nixos.org (forum ufficiale)

Grazie per l'attenzione!

Talk linux day 2024



POLITECNICO OPEN
unix LABS

Speakers

Lorenzo Andreasi
lolloandr@poul.org

Leonardo Salvini
jep@poul.org

Slides authors

Lorenzo Andreasi
lolloandr@poul.org

Leonardo Salvini
jep@poul.org



This work is licensed under a Creative Commons
Attribution-ShareAlike 4.0 International License.
Source code available [here](#)