DESIGN OF NETWORKS AND COMMUNICATION SYSTEMS

EMANUELE BOSCARI - MATTEO GRECO - THOMAS DALLA VIA

OUR MISSION

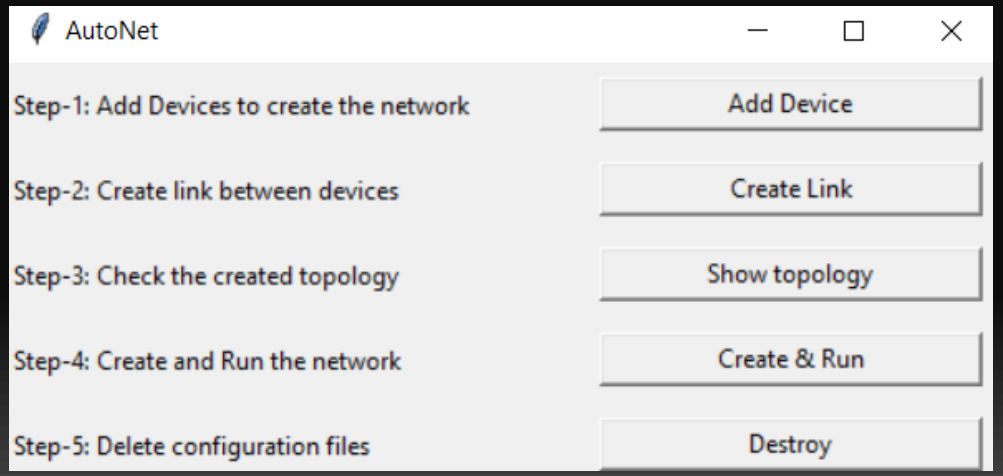
CREARE UN'APPLICAZIONE PER PRECONFIGUARARE ED ESEGUIRE UNA RETE DI VM/CONTAINERS E OPENVSWITCHES UTILIZZANDO VAGRANT

5 STEPS

- Configurazione dei dispositivi
- Collegamento dei dispositivi
- Verifica della topologia di rete
- Scrittura bash scripts, vagrantfile e avvio di Vagrant
- Cancellazione bash scripts

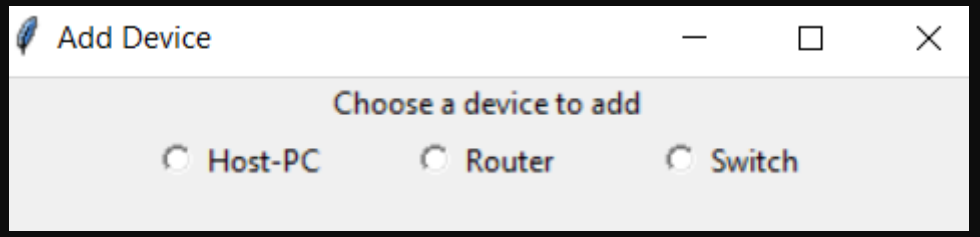
LET'S TAKE A LOOK

- GUI USER-FRIENDLY SVILUPPATA CON TKINTER, IL PACKAGE STANDARD PER L'IMPLEMENTAZIONE DI PROGRAMMI CHE NECESSITANO DI UNA GUI

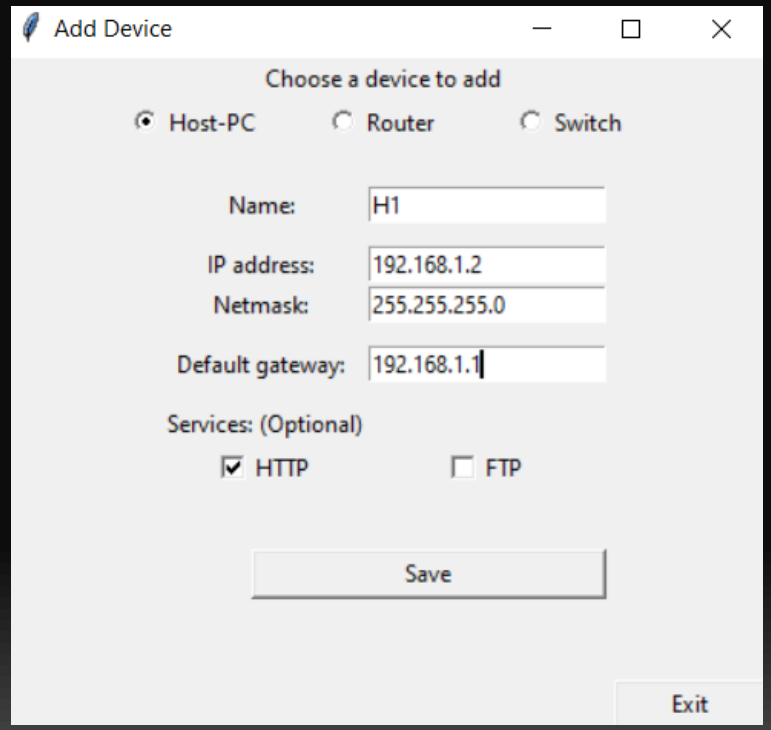


STEP 1 - ADD DEVICES

- CON IL PULSANTE ADD DEVICE SI APRE UNA SECONDA FINESTRA DALLA QUALE POSSIAMO SCEGLIERE TRAMITE RADIO BUTTON CHE DISPOSITIVO AGGIUNGERE



STEP 1 - ADD DEVICES



- DOPO AVER SELEZIONATO LA TIPOLOGIA SI APRE NELLA STESSA FINESTRA UNA FORM DOVE CONFIGURARE IL DISPOSITIVO SCELTO
- NEL CASO DI UN HOST, È POSSIBILE SCEGLIERE SE AGGIUNGERE O MENO UN SERVIZIO TRA I DUE DISPONIBILI (HTTP - FTP)

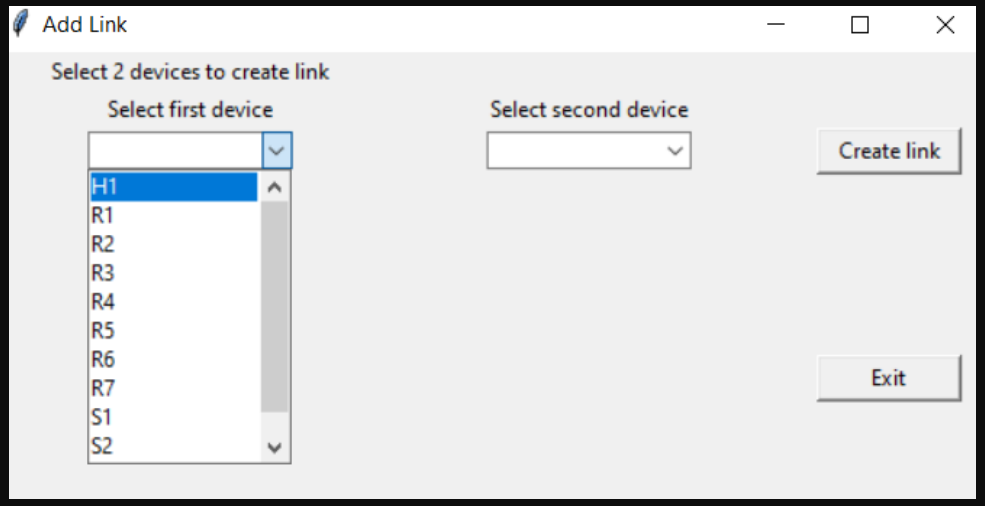
STEP 1 - ADD DEVICES

```python
@dataclass
class Host:
    def __init__(self, name, ip, mask, gateway, service):
        self.name = name
        self.ip = ip
        self.mask = mask
        self.gateway = gateway
        self.link = []
        self.service = service

    def __str__(self):
        return self.name
```

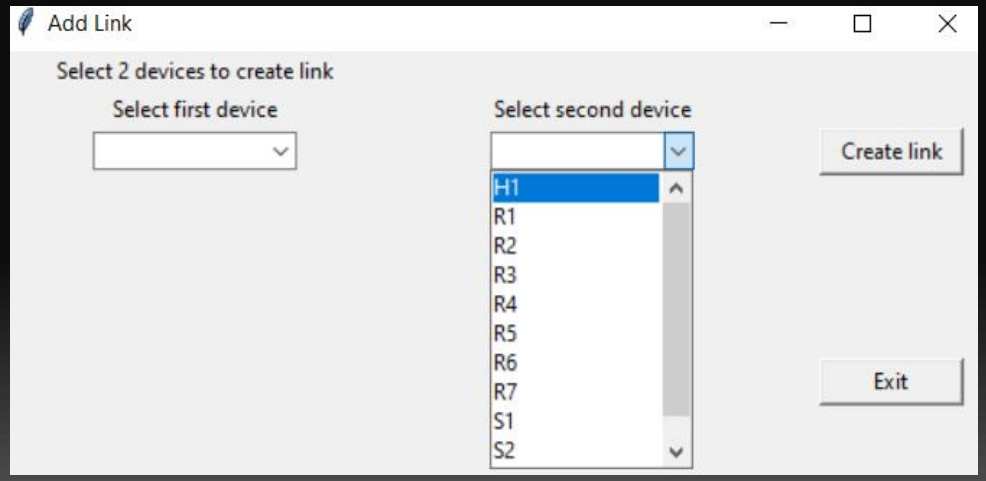
- I DISPOSITIVI CON LA RELATIVA CONFIGURAZIONE VENGONO SALVATI NELLE ISTANZE DELLE SPECIFICHE CLASSI

```python
def new_host(name, ip, mask, gate, serv):

    h = Host(name, ip, mask, gate, serv)
    print(h.name + " " + h.ip + " " + h.mask + " " + h.gateway + " " + str(h.service))
    trovato = False
    for i in device_list:
        if isinstance(i, Host):
            if name == i.name or ip == i.ip:
                print("esiste già")
                trovato = True
    if not trovato:
        device_list.append(h)
```

STEP 2 - CREATE LINKS



- COLLEGAMENTO DEI DISPOSITIVI PER LA CREAZIONE DELLA RETE



STEP 3 – SHOW TOPOLOGY

```
def show_tree(window):

    label_h = Label(window, width=20, text="Host list").pack()
    cols = ('name', 'ip', 'mask', 'gateway', 'link')
    treehost = ttk.Treeview(window, columns=cols, show='headings')
    treehost.pack(expand=True, fill="both")
    for col in cols:
        treehost.heading(col, text=col)
```

- POSSIBILITÀ DI VISUALIZZARE LE CONFIGURAZIONI DEI DISPOSITIVI CREATI

Host list

name	ip	mask	gateway	link
Host1	168.192.1.2	255.255.255.0	168.192.1.1	broadcast_Host1_Switch1
Host2	168.192.1.3	255.255.255.0	168.192.1.1	broadcast_Host2_Switch1

STEP 4 – CREATE & RUN

ATTRAVERSO «CREATE&RUN» SI CREANO:

- I FILE BASH PER OGNI DISPOSITIVO
- IL VAGRANTFILE

VENGONO ESEGUITI I COMANDI:

- VAGRANT UP
- VAGRANT STATUS

STEP 4 – CREATE & RUN

- PER UTILIZZARE LE CONFIGURAZIONI INSERITE DALL'UTENTE SI CREANO DEI FILE BASH CHE VENGONO POI INSERITI NEL VAGRANTFILE COME BASE PER LA CONFIGURAZIONE DELLE VMS

```
def write_sh():
    for x in device_list:
        if isinstance(x, Host):
            f = open(x.name + ".sh", "w")
            f.write("export DEBIAN_FRONTEND=noninteractive\n"
                    "sudo apt install -y curl\n"
                    "sudo ip addr add " + x.ip + "/" + str(mask_in_slash(x.mask)) + " dev enp0s8\n"
                    "sudo ip link set enp0s8 up\n"
                    "sudo ip route add default via "+x.gateway+"\n")
```

STEP 4 – CREATE & RUN

- IL VAGRANTFILE È UTILIZZATO PER CONFIGURARE VAGRANT IN BASE AL PROGETTO. LA FUNZIONE PRINCIPALE DEL VAGRANTFILE È DESCRIVERE E CONFIGURARE LE MACCHINE VIRTUALI

```
config.vm.define "host1" do |host1|
  hosta.vm.box = "ubuntu/bionic64"
  hosta.vm.hostname = "host1"
  hosta.vm.network "private_network", virtualbox__intnet: "broadcast_host1_switch1", auto_config: false
  hosta.vm.provision "shell", path: "host1.sh", run: 'always'
  hosta.vm.provider "virtualbox" do |vb|
    vb.memory = 256
  end
```

STEP 4 – CREATE & RUN

- PER POPOLARE IL VAGRANTFILE VENGONO UTILIZZATE LE CONFIGURAZIONI PRECEDENTEMENTE SALVATE ALL'INTERNO DELLE ISTANZE DELLE CLASSI

```
for x in device_list:
    f.write("   config.vm.define \"" + x.name + "\" do |" + x.name + "|\n"
            "       " + x.name + ".vm.box = \"ubuntu/bionic64\"\n"
            "       " + x.name + ".vm.hostname = \"" + x.name + "\"\n")
    for l in x.link:
        f.write("       " + x.name + ".vm.network \"private_network\", virtualbox__intnet: \"" + l + "\", auto_config: false\n")

    f.write("       " + x.name + ".vm.provision \"shell\", path: \"" + x.name + ".sh" + "\", run: 'always'\n")
    f.write("       " + x.name + ".vm.provider \"virtualbox\" do |vb|\n"
            "           vb.memory = 256\n"
            "       end\n"
            "   end\n")
f.write("end")
f.close()
```

STEP 4 – CREATE & RUN

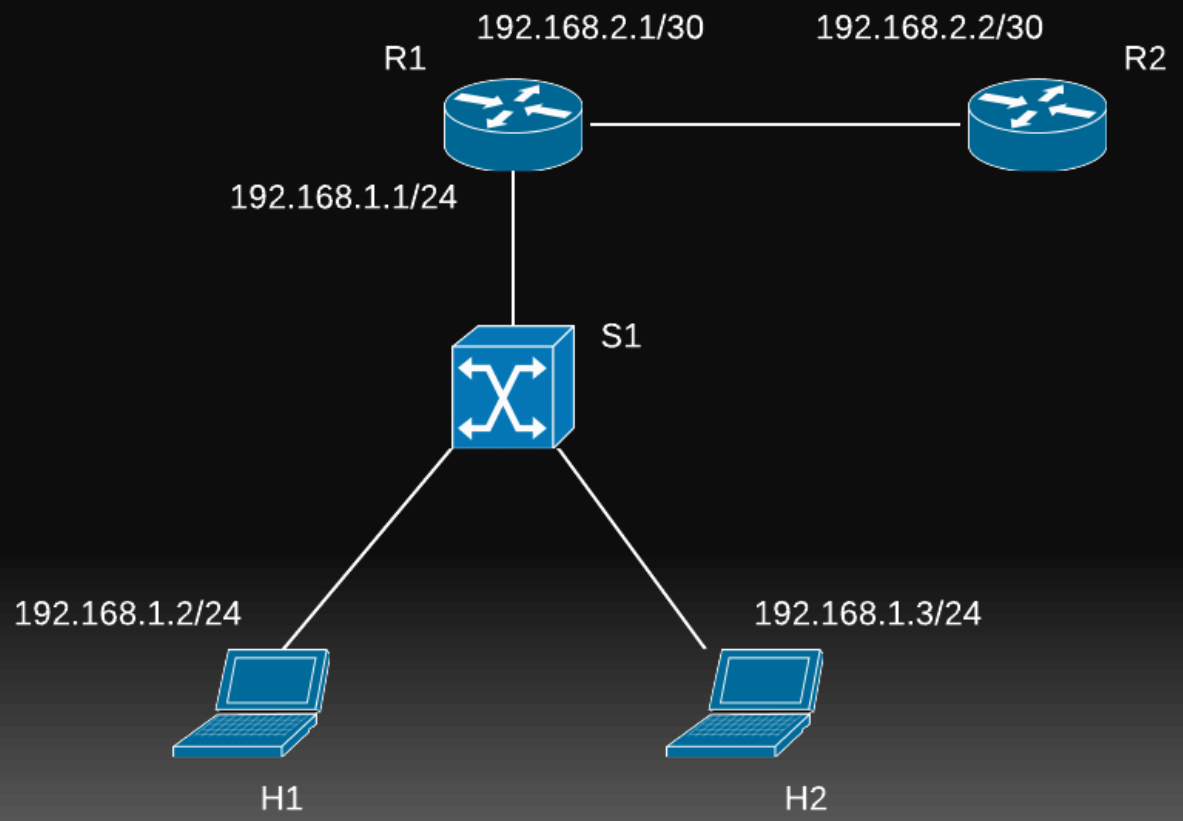
- SUCCESSIVAMENTE VIENE ESEGUITO IL COMANDO «VAGRANT UP» PER CREARE E CONFIGURARE LE VM IN BASE AL VAGRANTFILE
- UNA VOLTA FINITA L'ESECUZIONE DI «VAGRANT UP» VIENE ESEGUITO IL COMANDO «VAGRANT STATUS» PER VERIFICARE LO STATO DELLE VM CREATE

STEP 5 – DESTROY

- INFINE È POSSIBILE CANCELLARE I BASH SCRIPTS TRAMITE IL PULSANTE DESTROY PER RIPULIRE LA DIRECTORY PER UNA NUOVA CONFIGURAZIONE DI RETE

```
def destroy_net():
    if os.name == 'nt':      # Windows
        delete_sh = os.system("del \"*.sh\" /s /f /q")
        print("delete ran with exit code %d" % delete_sh)
    else:                    # Linux
        delete_sh = os.system("find . -type f -iname *.sh -delete")
        print("delete ran with exit code %d" % delete_sh)
```

DEMO RUN


192.168.2.1/30
192.168.2.2/30
R1
R2
192.168.1.1/24
S1
192.168.1.2/24
192.168.1.3/24
H1
H2

DEMO RUN