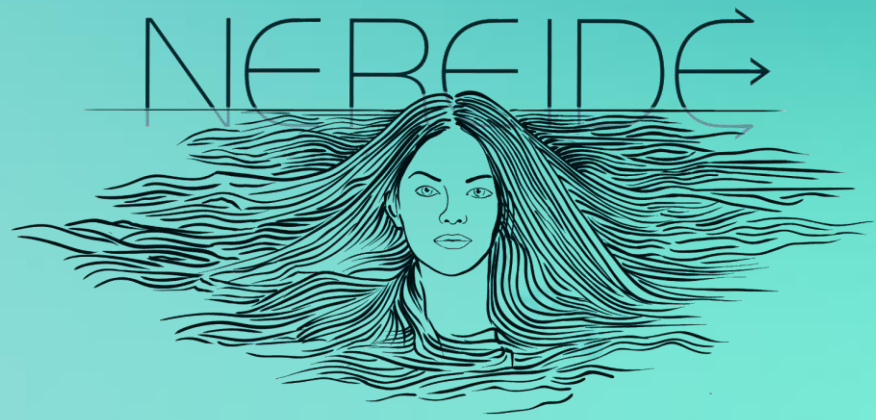
WORKSHOP GARR 2024

NET MAKERS

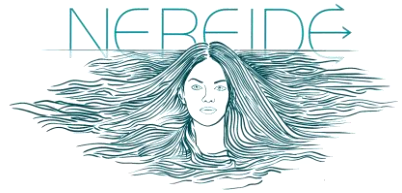
Infrastructure as Code: Dynamic Resource Allocation within Virtual Data Center

Stefano Cacciaguerra & Stefano Chiappini

ISTITUTO NAZIONALE
DI GEOFISICA E VULCANOLOGIA



NE.RE.I.D.E.



Real case: EMSO (ERIC focused on marine sciences)
└ Italy (coordinator country)
└ INGV (representing entity)

INGV implemented the NEw REsearch Infrastructure Datacenter for EMSO near the Western Ionian Sea site, at Portopalo di Capopassero (SR), Italy.

An ICT infrastructure for:

- archiving, processing, and sharing **scientific data** from marine observatories
- developing **advanced services**
- promoting multidisciplinary **research** in the deep marine environment.

Virtual Data Centers - the Core of NEREIDE

A **VDC** is a **managed environment** designed to facilitate **the control and operation of cloud resources** by DSs, including Virtual Machines (VMs), behind a **customizable gateway** with a dedicated **public IP address**.

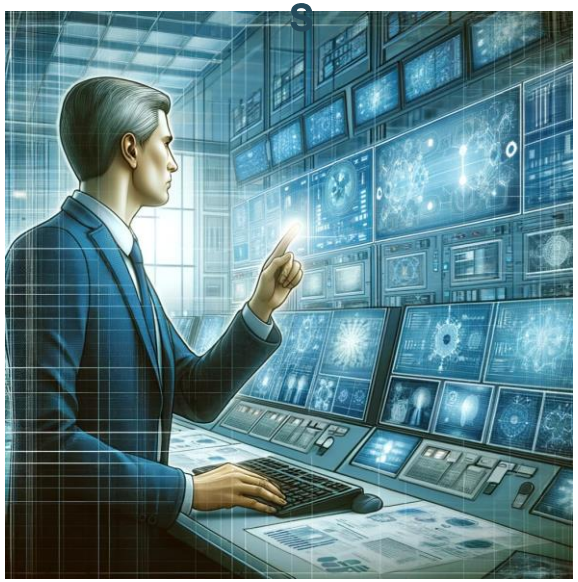
VDC

Customizable Gateway
Provides DSs with a unique, customizable **entry point**, enhancing security and personalization.

Management levels
DSs possess **administrative privileges** over their **cloud segments**, enabling them to create, manage, and configure VMs, networks, **firewalls**, **VPNs**, and **routing** configurations autonomously.
The whole cloud infrastructure and physical data center are managed by **Infrastructure Administrators (IAs)**, maintaining a balance between autonomy and control.

Roles

Infrastructure Administrators



Architecture design, VDC creation

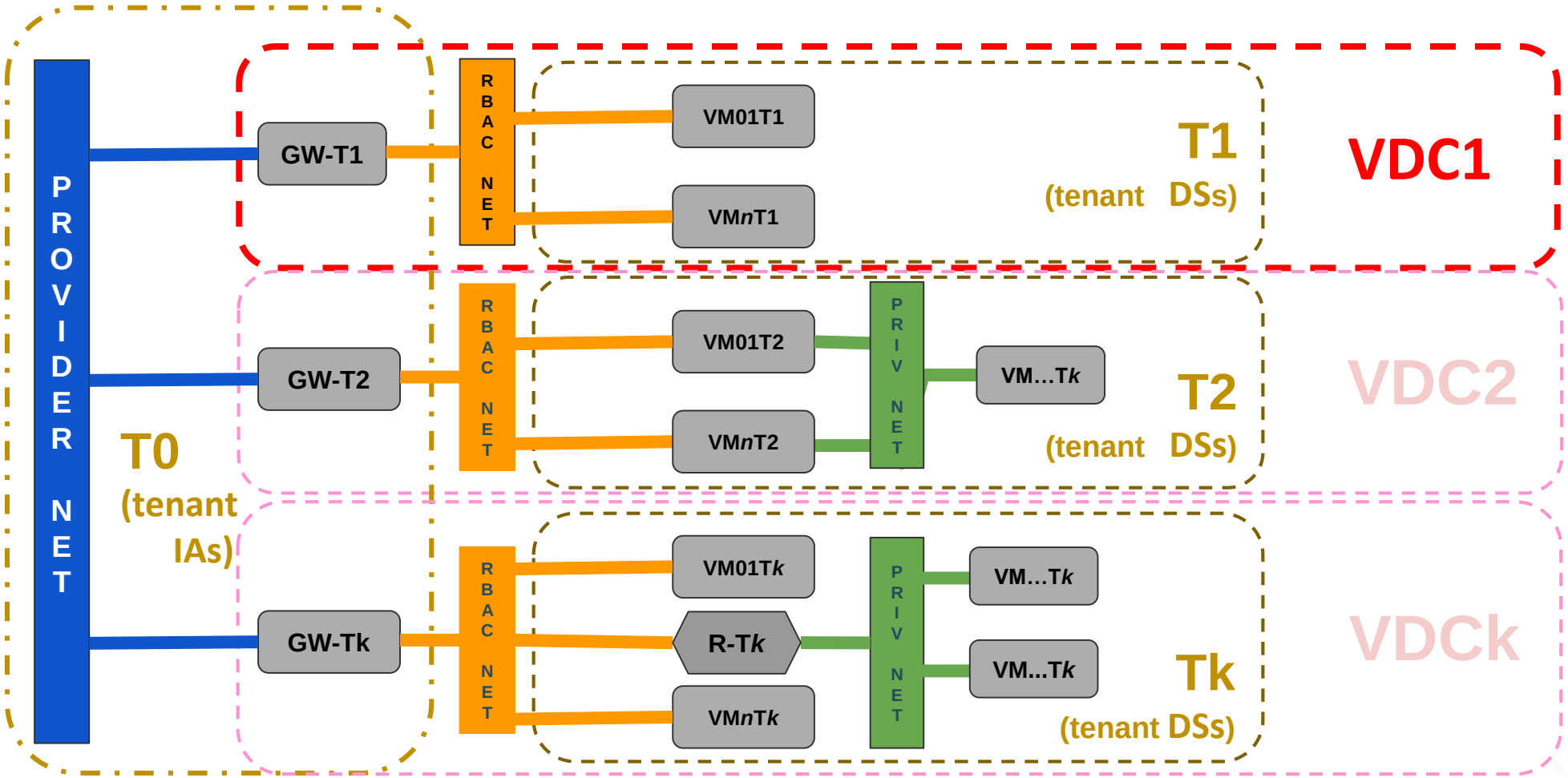
Data Scientists

Data Services development inside VDC

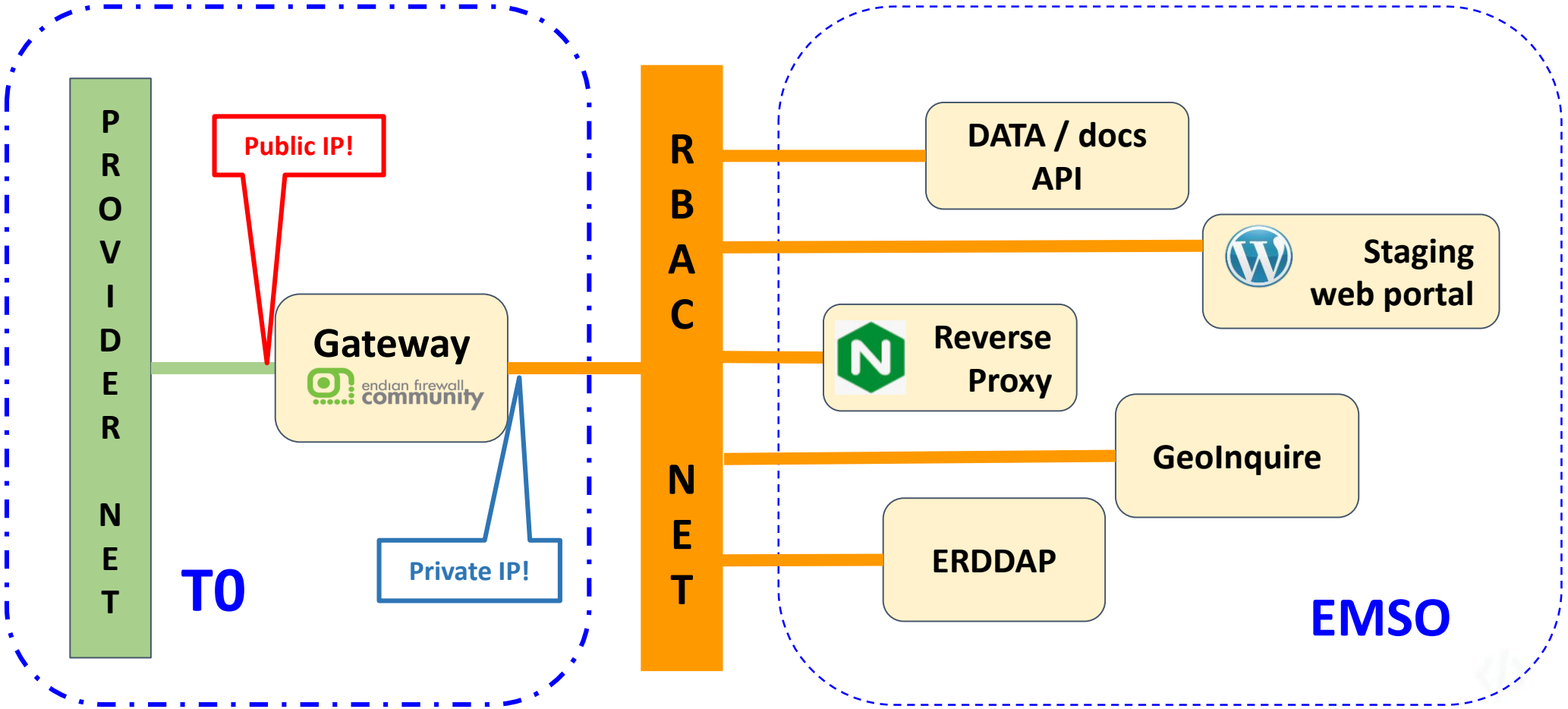
End Users

Use of Data Services

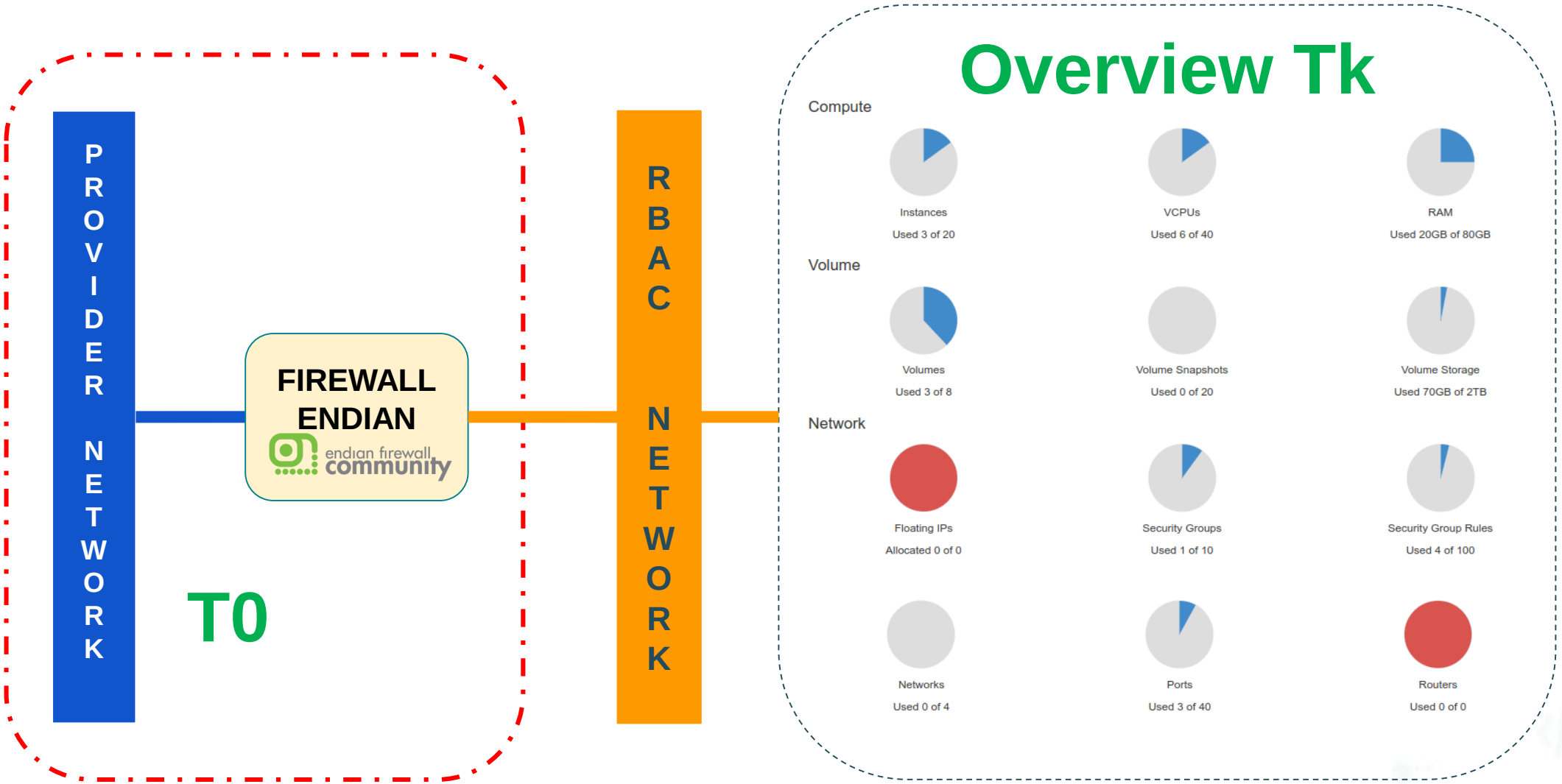
The Implementation of Virtual Data Centers



EMSO services migration - Project Anfitrite



VDC - Paradigm Shift



Advanced Management of VDCs

VDCs integrate **Infrastructure as Code** to allow DSs to manage and provision their Data Services through version-controlled scripts.

- IaC eliminates the need for manual environment setups, reducing operational overhead and increasing reproducibility.
- By codifying infrastructure, VDCs create **consistent, repeatable**, and **scalable** environments that can be rapidly provisioned.

This integration makes **complex data science tasks** more accessible and efficient across various scientific domains in a reproducible environment.

INGV

Cloud Automation vs Orchestration

IaC provides the framework and definitions that make possible:

Cloud Automation

- Creates **repeatable tasks** quickly with minimal operator intervention.
- Example: **CLI** to launch instances and **cloud-init** to automate them.

Cloud Orchestration

- Provides **coordination** among multiple automated tasks.
- Manages **dependencies** between different tasks, ensuring the correct order and automatic recovery in case of errors.
- Example: Ansible, Heat, Puppet, Terraform.

Cloud-init

A tool for initializing and automatically customizing VMs:

- Package installation, configuration management, and script execution
- **Configurations** can be reused for deploying **multiple VMs**, significantly reducing the time and effort required for their setup.

IT works to possible into two different phases:

- **Local Boot**: Before the network is active
- **Late Boot**: After applying the network configuration

It is native in Openstack and responsible for initializing a VM during their **installation,**

Bash, CLI and Cloud-init

Using a combination of Bash, CLI and Cloud-init, it is possible to obtain a **rudimentary cloud orchestration**

1. write a **Cloud-init YAML file**
2. with a **CLI command** assign a Cloud-Init script
3. use **bash script** to run more CLI commands

```
#cloud-config
package_update: true
package_upgrade: true
packages:
  - nginx
runcmd:
  - [ systemctl, enable, nginx ]
  - [ systemctl, start, nginx ]
```

1

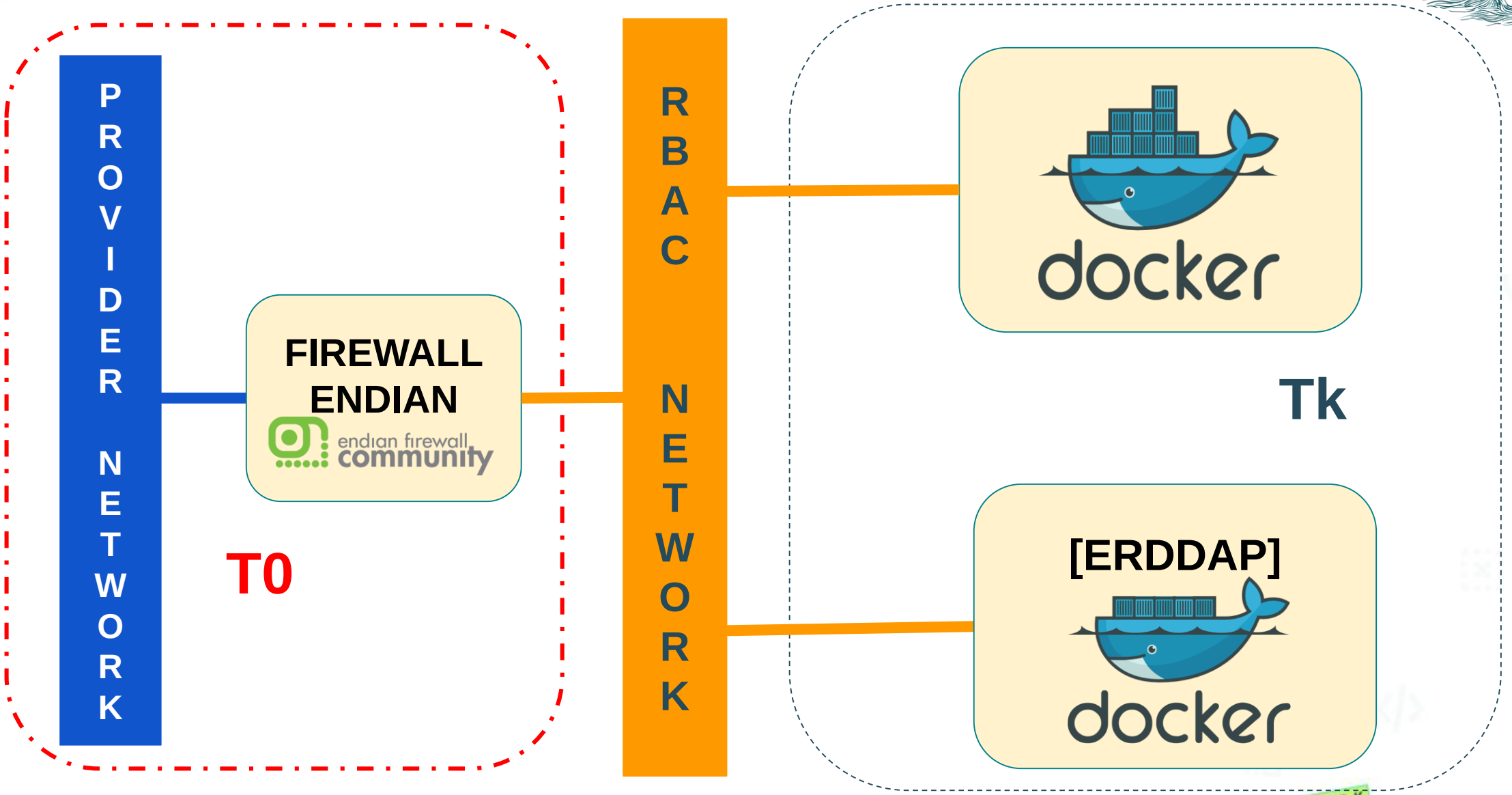
```
openstack server create --image <image-id> --flavor <flavor-id> --key-name <keypair-name>
<...> --user-data /path/to/your/cloud-config.yml <instance-name>
```

2

```
for (( i=1; i<=3; i++ ))
do
        openstack server create <…> --user-data ./cloud-init-slurm-slaves.yaml
        <...> v4-fixed-ip=172.16.0.2$i myslurmslave$i
done
```

3

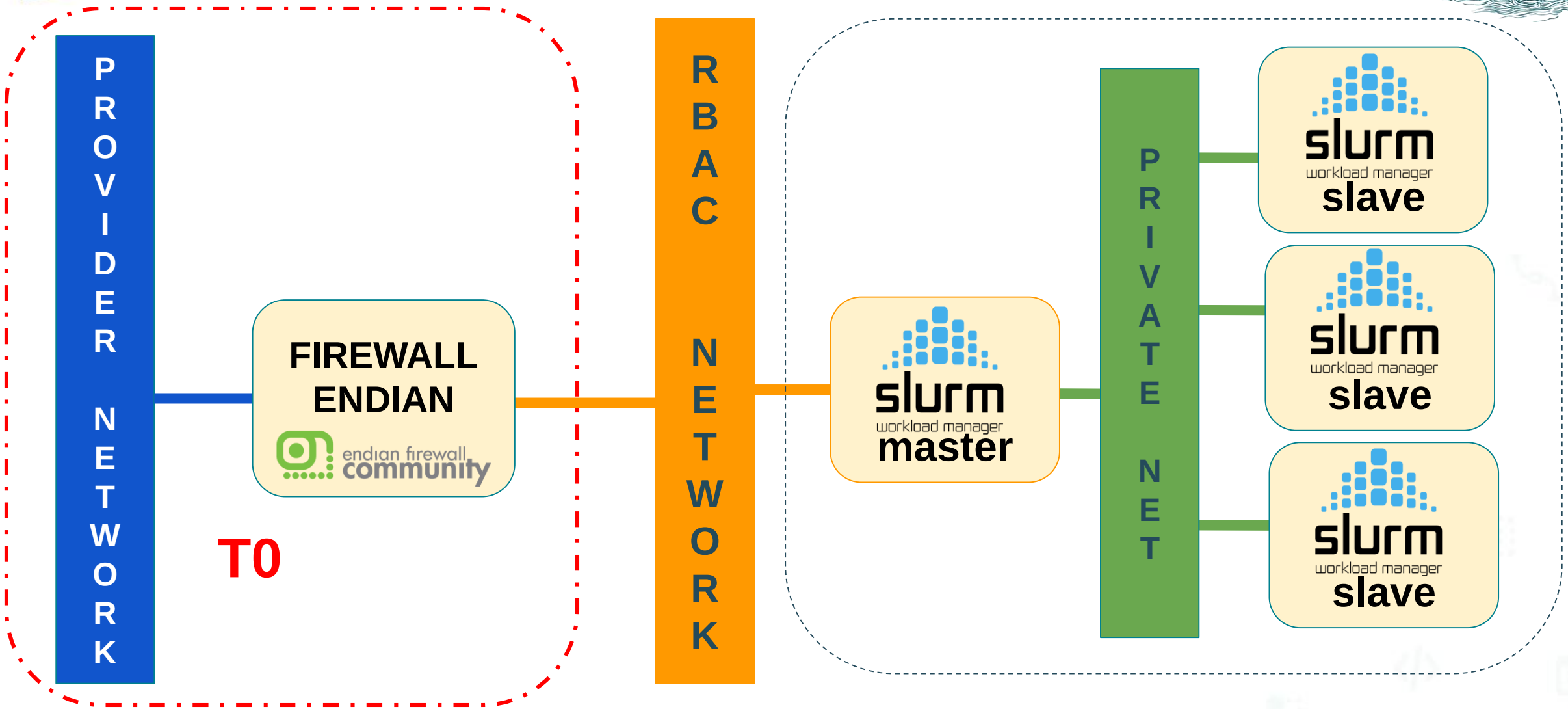
VDC - Docker


PROVIDER NETWORK
FIREWALL ENDIAN
endian firewall community
T0
RBAC NETWORK
docker
Tk
[ERDDAP]
docker

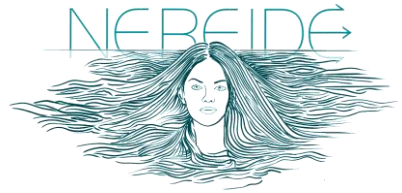
Docker

```
#cloud-config

# Update Packages
package_update: true
package_upgrade: true

# Install Specific Packages
packages:
  - ca-certificates
  - curl
  - gnupg
  - lsb-release

# Run CMDs
runcmd:
  - ['su', '-', 'root', '-c', 'curl -fsSL https://download.docker.com/.../gpg | sudo gpg --dearmor -o  .../docker-archive-keyring.gpg']
  - ['su', '-', 'root', '-c', 'echo "deb [arch=amd64 signed-by=.../docker-archive-keyring.gpg] https://download.docker.com/... jammy
stable" | sudo tee /etc/apt/sources.list.d/docker.list > /tmp/docker.log']
  - ['su', '-', 'root', '-c', 'apt update']
  - ['su', '-', 'root', '-c', 'apt install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin']
  - ['su', '-', 'root', '-c', 'usermod -aG docker ubuntu']
```

VDC - SLURM


PROVIDER NETWORK
FIREWALL ENDIAN
endian firewall community
T0
RBAC NETWORK
slurm workload manager master
PRIVATE NET
slurm workload manager slave
slurm workload manager slave
slurm workload manager slave

SLURM MASTER YAML



1

```
#cloud-config
...
# NFS, Slurm, Munge and slurm-client
packages:
  - nfs-kernel-server
  - slurm-wlm
  - munge
  - slurm-client
# Configure Services
write_files:
  # /etc/hosts
  - path: /etc/hosts
    content: |
      172.16.0.2 myslurmmaster
      172.16.0.21 myslurmslave1
      172.16.0.22 myslurmslave2
      172.16.0.23 myslurmslave3
    append: true
  # NFS
  - path: /etc/exports
    content: |
      /home/ *(rw,sync,no_root_squash,no_subtree_check)
      /etc/munge/
*(rw,sync,no_root_squash,no_subtree_check)
```

2

```
# Slurm.conf
  - path: /etc/slurm/slurm.conf
    content: |
      ClusterName=Vcluster
      ControlMachine=myslurmmaster
      ControlAddr=172.16.0.2
      ...
      # TIMERS
      SlurmctldTimeout=300
      ...
     # LOGGING
      SlurmctldDebug=3
      SlurmctldLogFile=.../slurm/slurmctld.log
      # Partitions
      NodeName=myslurmslave[1-3] CPUs=4 Boards=1 SocketsPerBoard=4 CoresPerSocket=1 ThreadsPerCore=1 RealMemory=7937
      NodeName=myslurmmaster CPUs=4 Boards=1 SocketsPerBoard=4 CoresPerSocket=1 ThreadsPerCore=1 RealMemory=7937
      PartitionName=allqueue Nodes=ALL Default=YES MaxTime=30 State=UP
```

3

```
## Enable and Start services
runcmd:
  #nfs server
  - exportfs -a
  - systemctl enable --now nfs-kernel-server
  # munge
  - systemctl enable --now munge
  # slurmctld
  - mkdir -p /var/spool/slurm
  - mkdir -p /var/spool/slurm/d
  - chown root:root /var/spool/slurm
  - chown root:root /var/spool/slurm/d
  - systemctl enable --now slurmd
  - systemctl enable --now slurmctld
```

SLURM SLAVES YAML

1

```
#cloud-config
…
# Install nfs-client, slurm and munge
packages:
  - nfs-common
  - slurm-wlm
  - munge

# Write Configuration Files
write_files:
 # Hosts
  - path: /etc/hosts
    content: |
      172.16.0.2 myslurmmaster
      172.16.0.21 myslurmslave1
      …
# Slurm.conf
  - path: /etc/slurm/slurm.conf
    content: |
       ClusterName=Vcluster
       ControlMachine=myslurmmaster
       ControlAddr=172.16.0.2
        …
```

2

```
# Partitions
     NodeName=myslurmslave[1-3] CPUs=4 … RealMemory=7937
     NodeName=myslurmmaster CPUs=4 … RealMemory=7937
     PartitionName=allqueue Nodes=ALL … State=UP
     PartitionName=threequeue Nodes=myslurmslave[1-3] … State=UP

# Start Services
runcmd:
  # Enabled Munge
  - systemctl enable --now munge
  - rm -f /etc/munge/munge.key
  # NFS mount
  - echo "172.16.0.2:/home /home nfs defaults 0 0" >> /etc/fstab
  - echo "172.16.0.2:/etc/munge/  /etc/munge/ nfs defaults 0 0" >> /etc/fstab
  - mount -a
  # Restart Munge
  - systemctl restart munge
  # Enabled and Started Slurm

 …
  - systemctl enable --now slurmd
```

Start Slurm Cluster

```
#!/bin/bash
…
openstack server create --image <UbuntuServer22.04> --flavor 4 --key-name <scaccia_edcsa>
--user-data ./cloud-init-slurm-master.yaml --nic net-id=<ReteRBAC> --nic net-id=
<RetePrivata>,v4-fixed-ip=172.16.0.2 --security-group <SLURM-Sec> "myslurmmaster"
…
sleep 10
…
for (( i=1; i<=3; i++ ))
do
openstack server create --image <UbuntuServer22.04> --flavor 4 --key-name <scaccia_edcsa>
--user-data ./cloud-init-slurm-slaves.yaml --nic net-id=<RetePrivata>,v4-fixed-ip=172.16.0.2$i
--security-group <SLURM-Sec> "myslurmslav$i"
…
done
```

Delete Slurm Cluster

```
#!/bin/bash
filename='list-slurm.tmp'
openstack server list | grep slurmslave | cut -f 2 -d ' ' > list-slurm.tmp
openstack server list | grep slurmmaster | cut -f 2 -d ' ' >> list-slurm.tmp
...
n=0
while read line; do
        openstack server stop $line
        n=$((n+1))
        sleep 1
done < $filename
...
```

1

```
...
n=0
sleep 10
while read line; do
        openstack server delete $line
        n=$((n+1))
        sleep 1
done < $filename
```

2

Conclusions
«The Swiss Army Knife of Scientific Computing»

VDCs represent the evolution in the **concept of virtualization**, moving beyond individual VMs to include comprehensive data center infrastructures, ready-to-use, fully equipped with the advanced tools.

By codifying infrastructure, VDCs enable the creation of **consistent**, **repeatable**, and **scalable** environments that can be rapidly provisioned.

Like a technological Swiss Army Knife, it can offer a comprehensive toolkit providing an array of functions, from **data analysis** to **complete infrastructure management**, all within a **single platform**.

Grazie

WORKSHOP GARR 2024

NET MAKERS

Per le domande:
wooclap.com e codice WSGARR24