OPENNEBULA AND CLOUD ARCHITECTURE

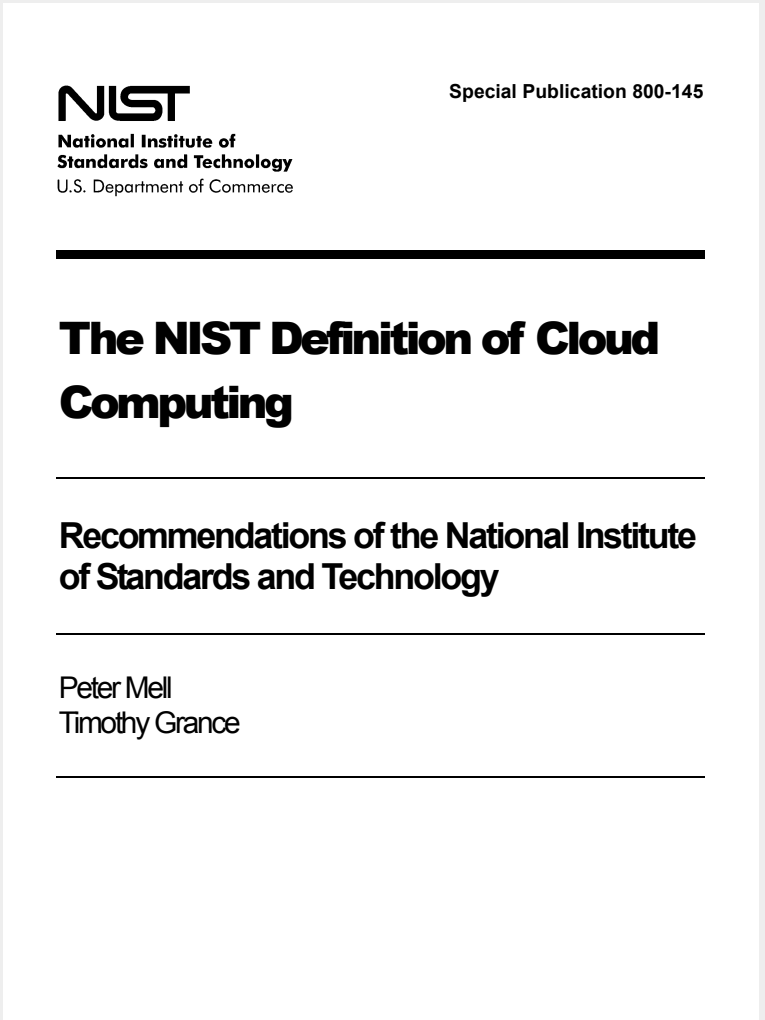
Stefano Bagnasco | INFN Torino

OUTLINE

- Recap from yesterday
- OpenNebula Open Cloud Reference Architecture
- OpenNebula internal achitecture
- OpenNebula concepts & tools

DEFINITION OF CLOUD COMPUTING

- Fundamental Characteristcs
  - On-demand self-service
  - Broad network access.
  - Resource pooling.
  - Rapid elasticity.
  - Measured service.
- Service models
  - Software as a Service
  - Platform as a Service
  - Infrastructure as a Service
- Deployment Models
  - Private Cloud
  - Community Cloud
  - Public Cloud
  - Hybrid Cloud


NIST
National Institute of Standards and Technology
U.S. Department of Commerce

Special Publication 800-145

The NIST Definition of Cloud Computing

Recommendations of the National Institute of Standards and Technology

Peter Mell
Timothy Grance

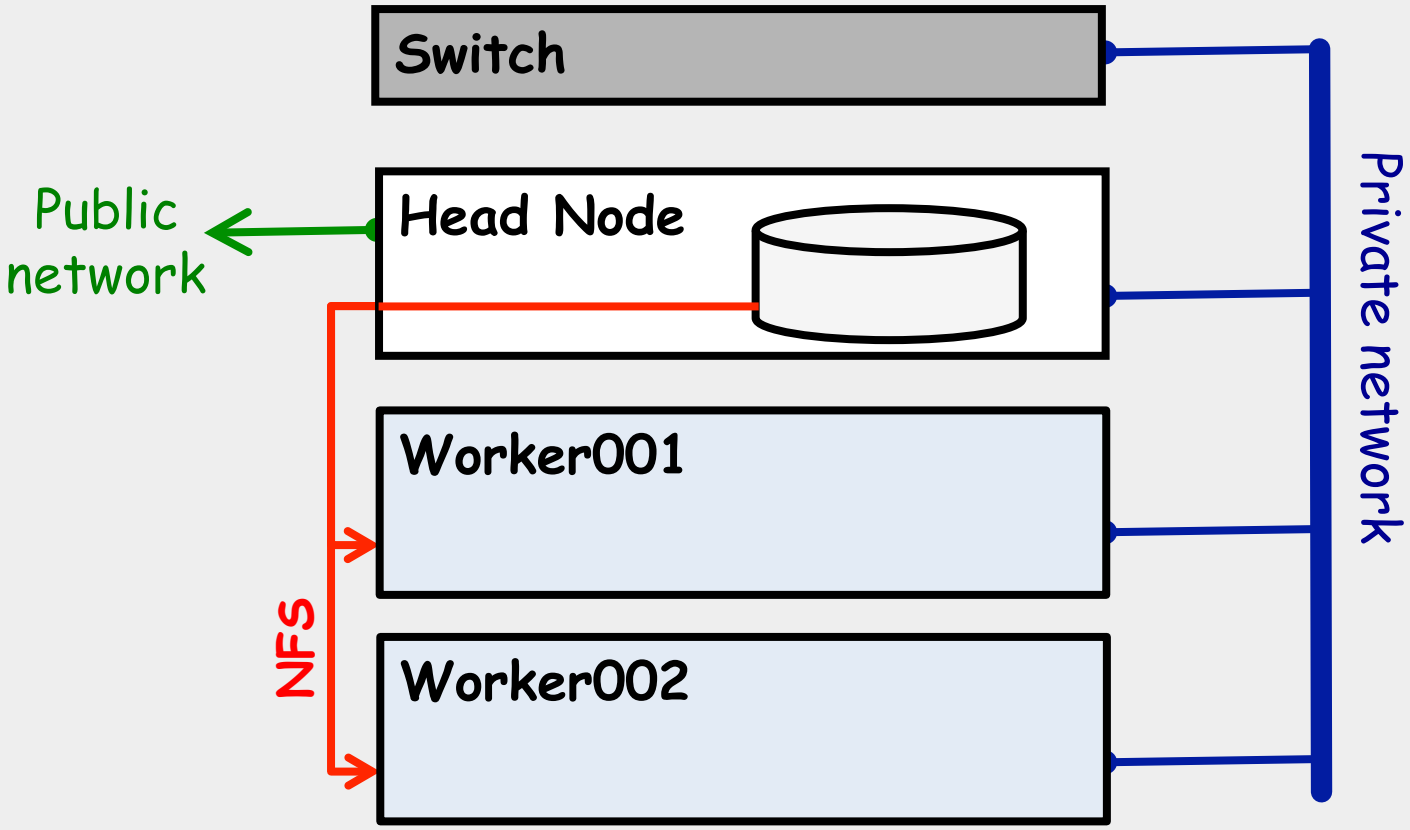
http://goo.gl/DN1yrg

CATTLE AND PETS

- Cloud-aware applications:
  - Intrinsically distributed
  - Stateless
  - Failover managed in-app
  - Scaling managed in-app

- "Legacy" applications:
  - Client-server
  - No horizontal scalability
  - Failover managed by infrastructure
  - Scaling managed by infrastructure

- Build a Private Cloud infrastructure to host a number of Virtual Batch Farms



SHOPPING LIST

- Hardware
- Virtualization tools
  - See yesterday's lesson and hands-on
- OS Images
  - Bare OS or appliance
- Contextualization tools
  - ...
- Cloud platform
  - OpenNebula

OpenNebula

Open Cloud Reference Architecture

Version 1.0

Abstract

The OpenNebula Cloud Reference Architecture is a blueprint to guide IT architects, consultants, administrators and field practitioners in the design and deployment of public and private clouds fully based on open-source platforms and technologies. It has been created from the collective information and experiences from hundreds of users and cloud client engagements. Besides main logical components and interrelationships, this reference documents software products, configurations, and requirements of infrastructure platforms recommended for a smooth OpenNebula installation. Three optional functionalities complete the architecture: high availability, cloud bursting for workload outsourcing, and federation of geographically dispersed data centers.

The document describes the reference architecture for Basic (small to medium-scale) and Advanced (medium to large-scale) OpenNebula Clouds and provides recommended software for main architectural components, and the rationale behind them. Each section also provides information about other open-source infrastructure platforms tested and certified by OpenNebula to work in enterprise environments. To complement these certified components, the OpenNebula add-on catalog can be browsed for other options supported by the community and partners. Moreover, there are other components in the open cloud ecosystem that are not part of the reference architecture, but are nonetheless important to consider at the time of designing a cloud, like for example Configuration Management and Automation Tools for configuring cloud infrastructure and manage large number of devices.

OPEN CLOUD REFERENCE ARCHITECTURE

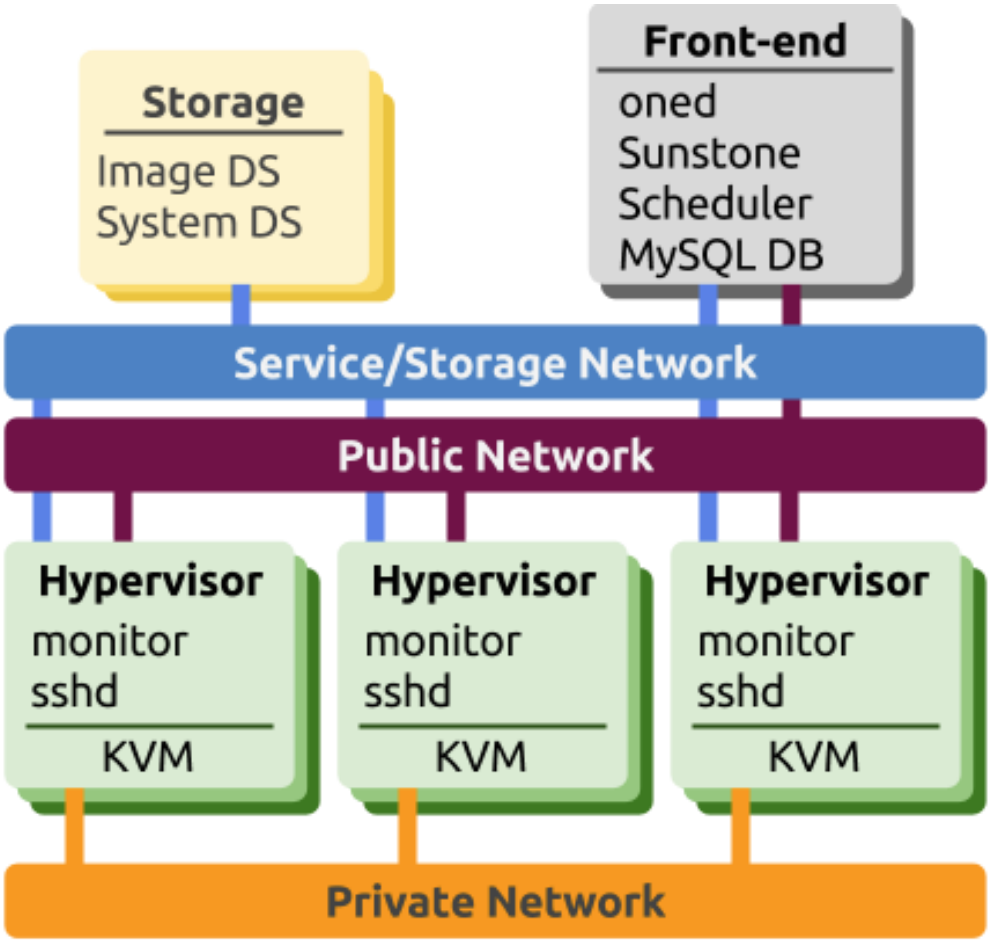


Figure 1. Reference Architecture, a bird's eye view

PHYSICAL HOSTS

- Servers that will host the Virtual Machines
  - Often called “Hypervisors” (like the software)
  - KVM (OpenNebula supports also vCenter and Xen)
  - Monitoring daemons
  - sshd for system connection

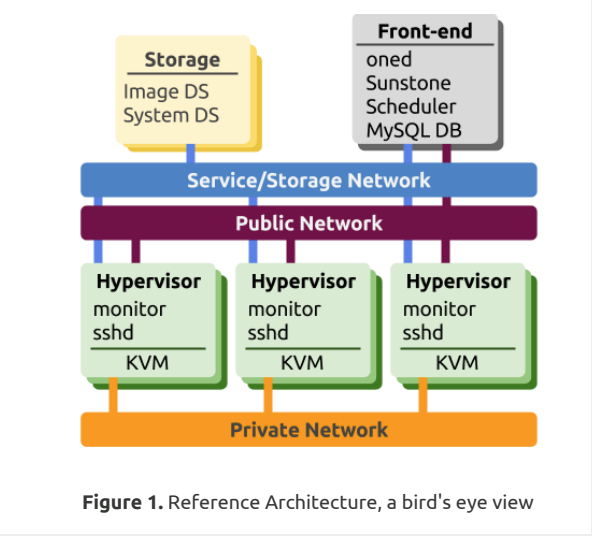


Figure 1. Reference Architecture, a bird's eye view

Used by OpenNebula and the infrastructure:

- Service and Storage network
  - Monitoring and control information
  - Image transfers

Used by the Virtual Machines:

- Private Network
  - Private IPs
  - Intra-cloud communications
- Public Network
  - Public IPs
  - Incoming connectivity to VMs

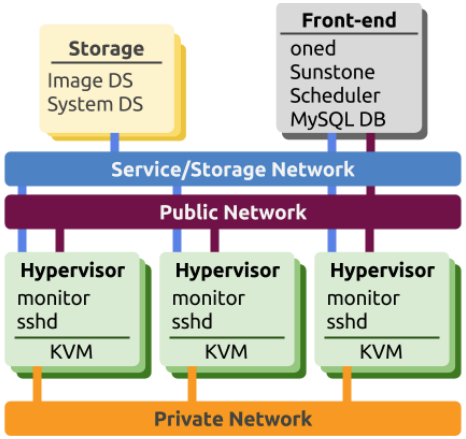


Figure 1. Reference Architecture, a bird's eye view

- Service datastores don't necessarily need to be shared across VMs
  - Images can be transferred to the hypervisors' disk through ssh and started locally
- Image Repository Datastore
  - Holds the OS images
- System Datastore
  - Holds the running instances
  - If it's a shared FS, VMs can be "live-migrated"

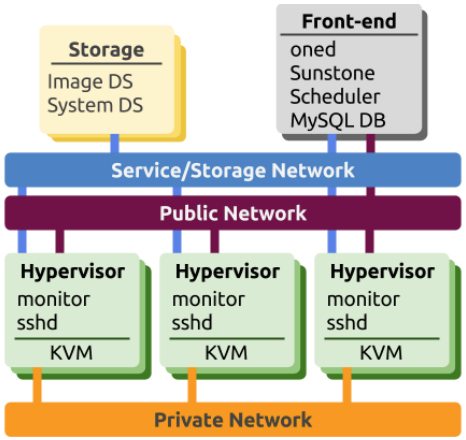


Figure 1. Reference Architecture, a bird's eye view

THE CONTROL NODE

- Runs the OpenNebula stack
  - oned (the main daemon)
  - schedd (the VM scheduler)
  - Sunstone (the web-based GUI)
  - MySQL DB backend (can be separate)
  - API services (OCCI or EC2)
  - Advanced services (OneFlow, OneGate,...)
  - Control node unavailability does not affect running VMs
  - Only control on them (start & stop, monitoring,...)

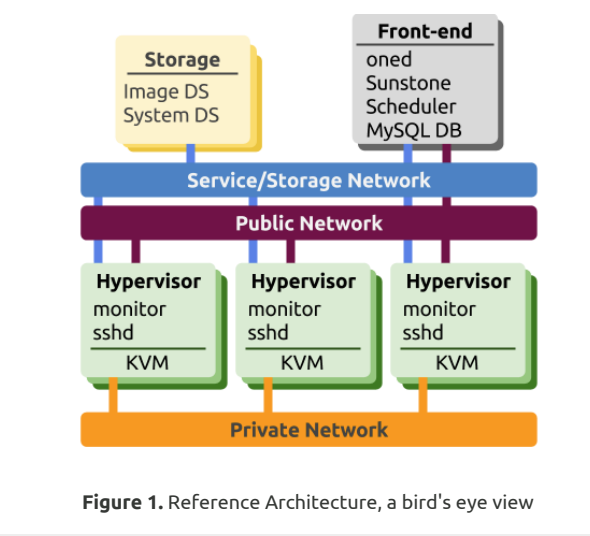


Figure 1. Reference Architecture, a bird's eye view

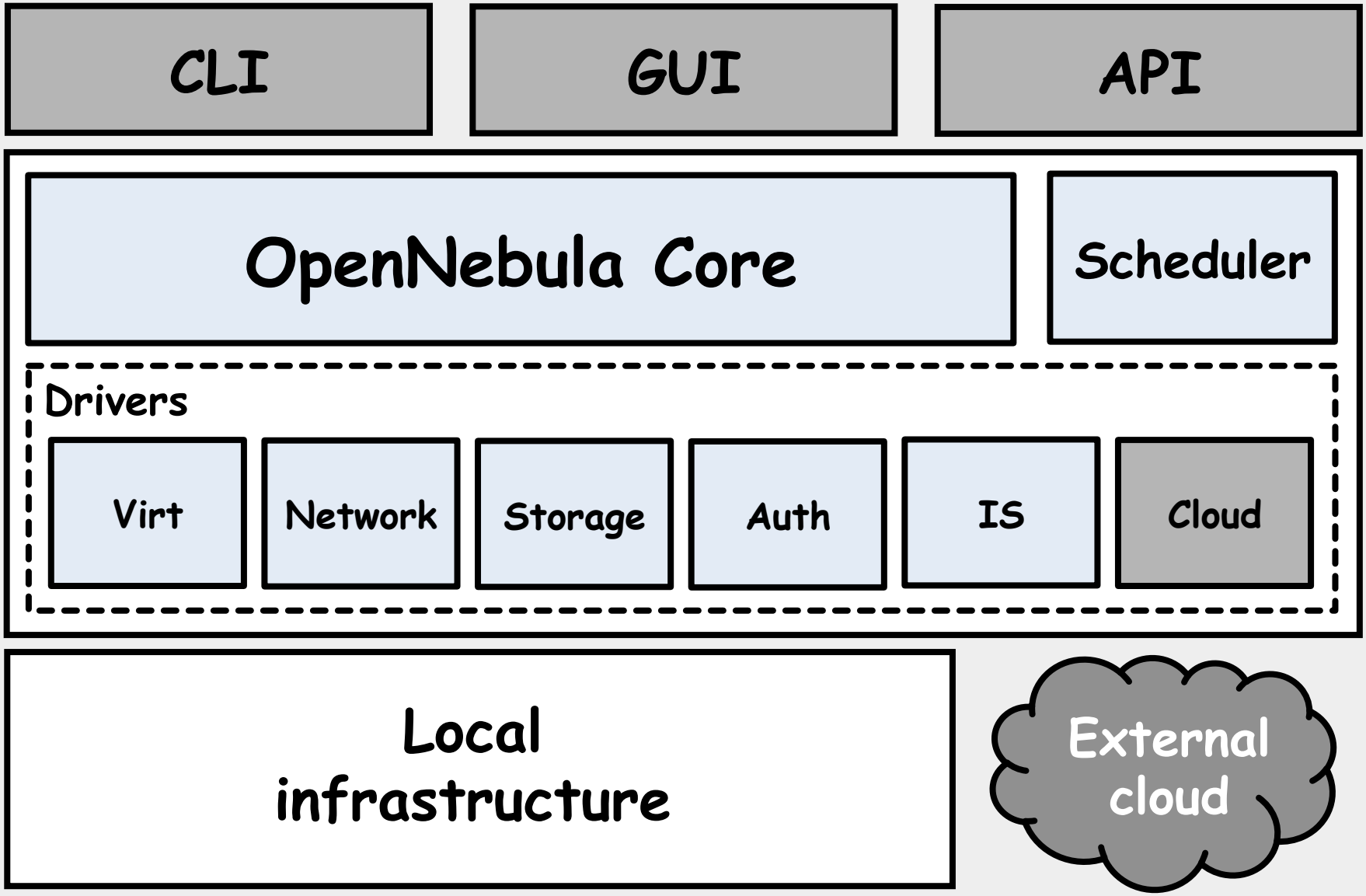
OPENNEBULA

- First established as a research project back in 2005 by Ignacio M. Llorente and Rubén S. Montero.
- Since its first public release of software in March 2008, it has evolved through open-source releases and now operates as an open source project.
- OpenNebula.org is a project now managed and supported by OpenNebula Systems.

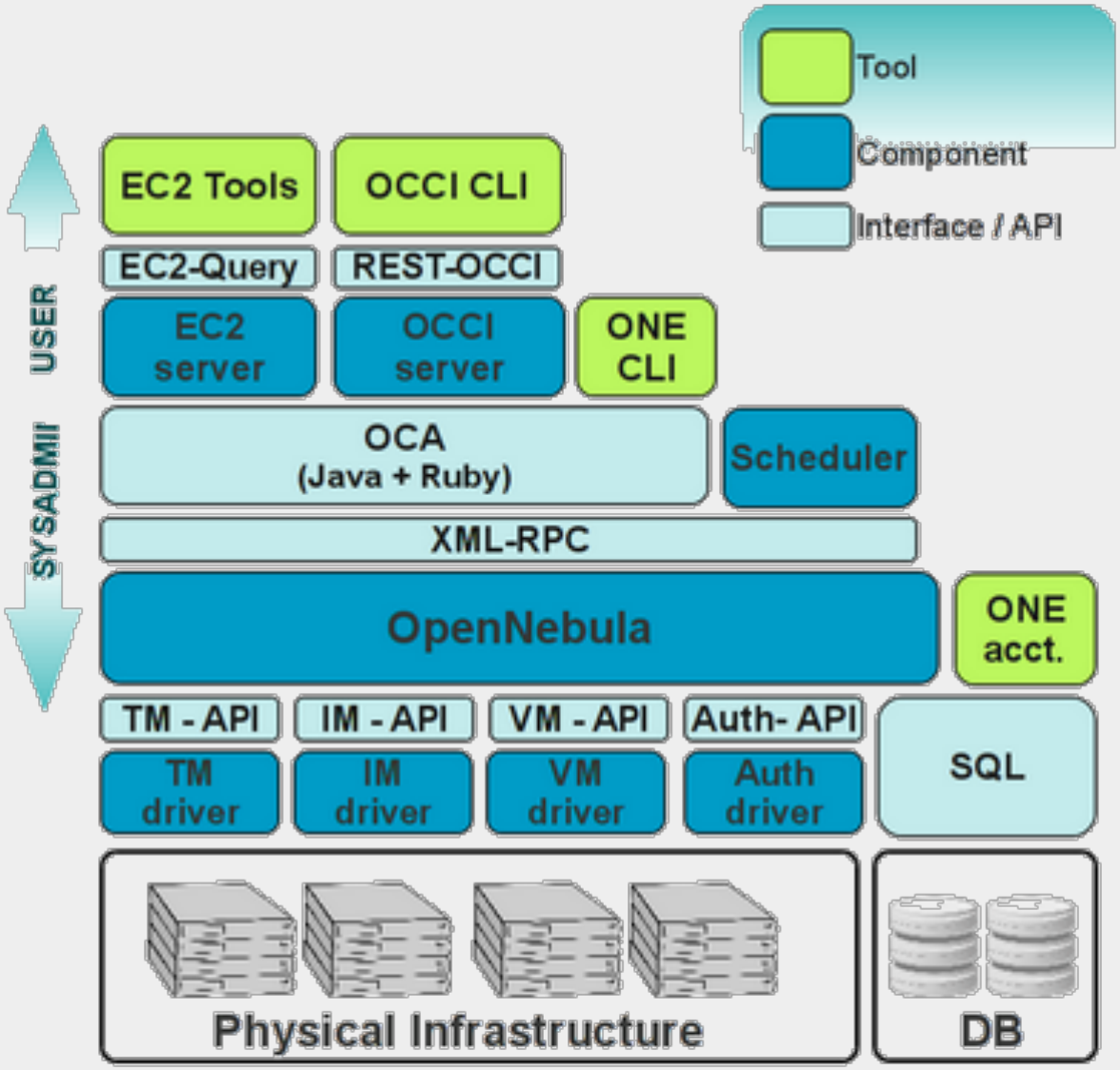
DISCLAIMER

- This is only a partial overview of OpenNebula most basic concepts and capabilities
  - Time is limited...
- Reference release for this is 4.8
  - Oldish, but we know it well
- Daniel Molina will give a talk about the future and exciting new features in 4.14
  - For everything in between, everything we could not fit in the time allotted, and any kind of extra details please refer to OpenNebula documentation
  - http://docs.opennebula.org/4.12

OPENNEBULA INTERNAL ARCHITECTURE



A MORE DETAILED VIEW



BASIC RESOURCES AND COMPONENTS

- **Host**: Physical machine running a supported hypervisor.
- **Cluster**: Pool of hosts that share datastores and virtual networks.
- **Template**: Virtual Machine definition.
  - Also, everything else is defined through templates
- **Image**: Virtual Machine disk image.
- **Virtual Machine**: Instantiated Template.
  - A Virtual Machine represents one life-cycle, and several Virtual Machines can be created from a single Template.
- **Virtual Network**: A group of IP leases that VMs can use to automatically obtain IP addresses.

IMAGES

Disk Images are key component of any cloud system

- Disk images can be **persistent** or **volatile**
  - Persistent images are saved back to the repository when the VM is destroyed
  - Changes on volatile images are lost
- Three types of images:
  - OS
  - Datablock
  - CDROM
  - Plain file (Kernel, RamDisk, Context)

DATASTORES

OpenNebula uses **Datastores** to handle the VM disk Images.

- A Datastore is any storage medium used to store disk images for VMs
  - Typically, a datastore will be backed by SAN/NAS servers.
  - Each Datastore has to be accessible through the front-end using any suitable technology NAS, SAN or direct attached storage.
- Image datastores can be of different type depending on the underlying technology:
  - **File-system**
  - vmfs
  - FS LVM
  - Block LVM
  - Ceph
  - GlusterFS
  - Devices
  - ...

TRANSFER MANAGERS

The Disk images registered in a datastore are transferred to the hosts by the **transfer manager** drivers.

- TM drivers perform low-level storage operations.
- OpenNebula includes several different TMs
  - Not all TMs work with all Datastores!
    - **shared**
    - **ssh**
    - **qcow2**
    - vmfs
    - ceph
    - lvm
    - fs_lvm
    - dev

VIRTUAL NETWORKS

OpenNebula allows the creation of **Virtual Networks** by mapping them on top of the physical ones.

- They will be available to the VMs through the corresponding bridges on hosts.
- A virtual network definition consists of three different parts:
  - the underlying physical network infrastructure
  - The logical address space available (IPv4, IPv6, dual stack)
  - Context attributes (e.g. netmask, DNS, gateway,...)
- OpenNebula also comes with a **Virtual Router** appliance
  - to provide networking services like DHCP, DNS,...

NETWORK ISOLATION

OpenNebula connects a newly created VM's network interfaces to the bridge or physical device specified in the definition.

- This will allow the VM to have access to different networks, public or private.
- In some cases, the relevant bridges are also created on the host
- Each host has an associated Network Driver, with different network isolation technologies:
  - **dummy**
  - fw
  - 802.1Q
  - **ebtables**
  - **ovswitch**
  - VMware

TEMPLATES

Resources and components are defined in OpenNebula through templates.

- All templates share the same syntax
- The most important ones are **VM templates**
  - Managed through the **onetemplate** command

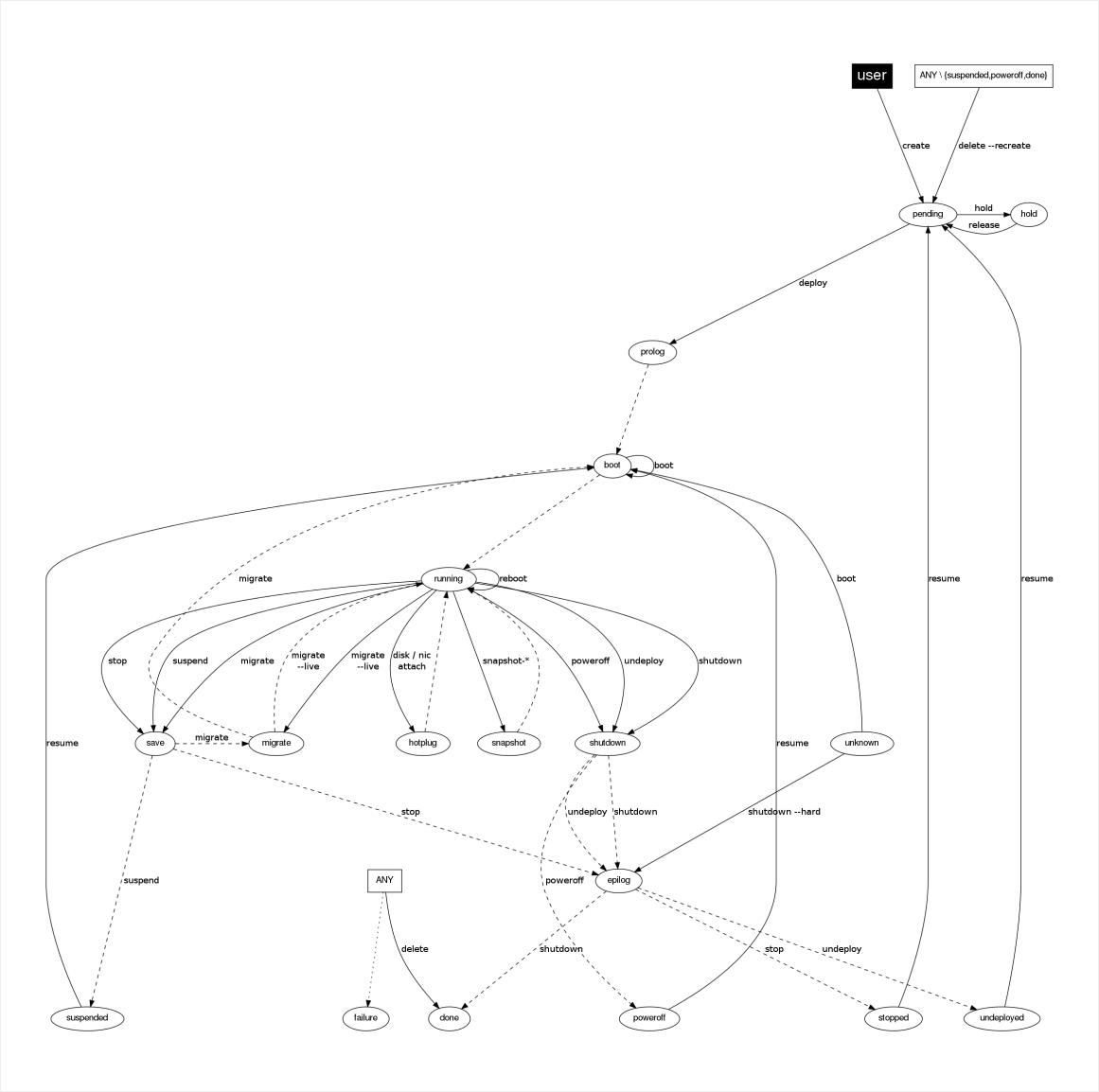
```
NAME   = test-vm
MEMORY = 128
CPU    = 1

DISK = [ IMAGE  = ”CentOS6" ]
DISK = [ TYPE     = swap,
         SIZE     = 1024 ]

NIC = [ NETWORK = "Public", NETWORK_UNAME="oneadmin" ]

GRAPHICS = [
  TYPE    = "vnc",
  LISTEN  = "0.0.0.0"]
```

BASIC VM LIFECYCLE

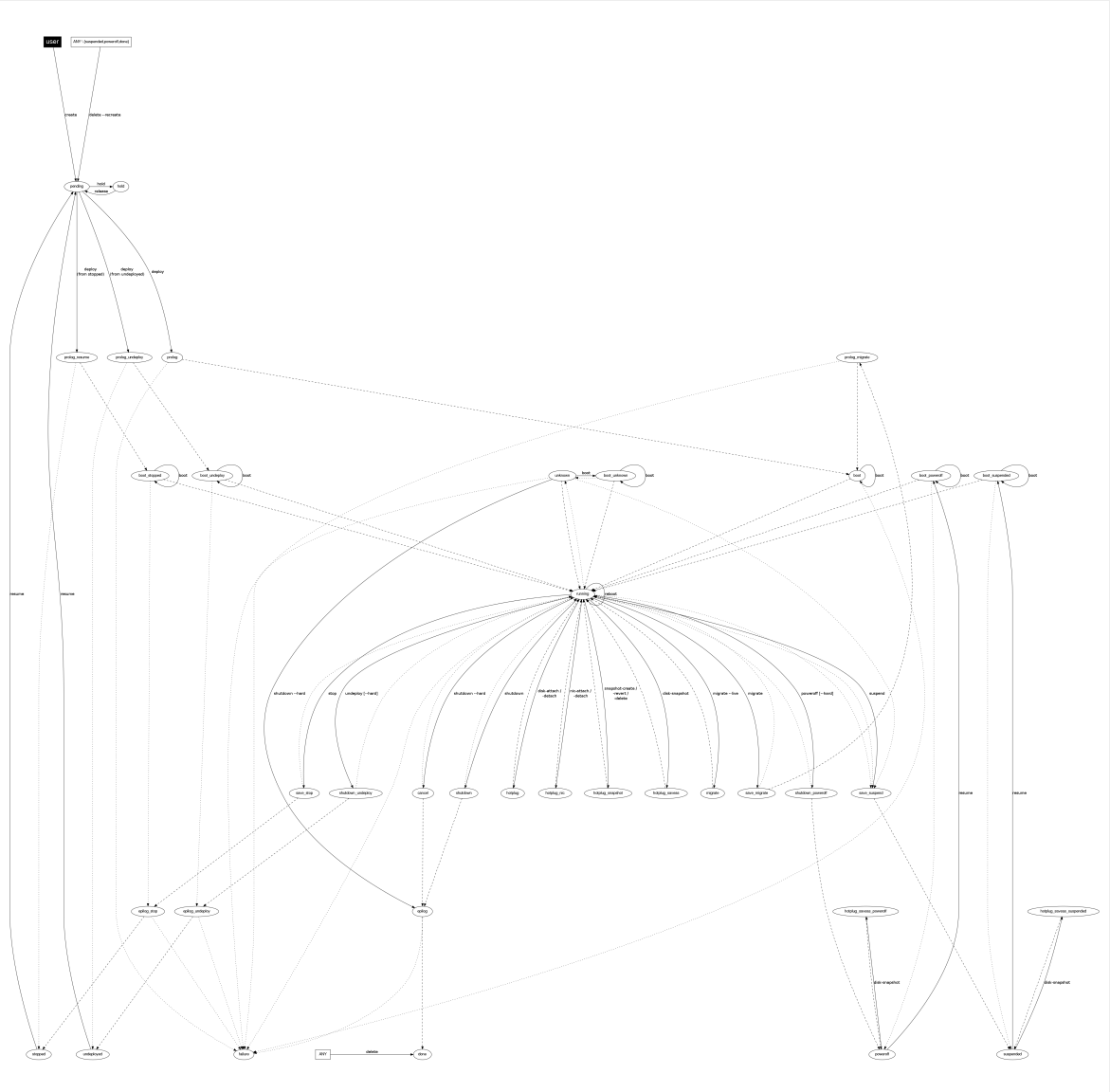
- **Pending** Initial state, waiting for a resource to run on.
- **Prolog** The system is transferring the VM files to the host.
- **Boot** OpenNebula is waiting for the hypervisor to create the VM.
- **Running** The VM is running, the virtualization driver will monitor it.
- **Save** The system is saving the VM files after a migration, stop or suspend operation.
- **Epilog** The system is cleaning up the host, and disk images to be saved are copied back to the system datastore.
- **Shutdown** OpenNebula has sent the VM the shutdown ACPI signal, and is waiting for it to complete the shutdown process.
- **Stopped, Suspended, Poweroff** The VM is stopped.
- **Failed** The VM failed.

(There are more...)

SIMPLIFIED VM LIFECYCLE



FULL VM LIFECYCLE



THE SCHEDULER

The OpenNebula **scheduler** matches the VMs in the queues with the available hosts

- Implemented as a separate daemon
- Just a FIFO buffer, no scheduling algorithms like FairShare
- Several policies can be implemented:
  - Striping
  - Packing
  - Load-Sensitive
  - ...

CONTEXTUALIZATION

Default method for contextualization is the **Virtual CDROM**

- The image needs the `opennebula-context` RPM package
- ONe generates a `context.sh` file based on information in the VM template
  - Networking
  - root ssh key
- ONe generates a virtual CDROM image with context.sh
  - Also custom extra files can be added
  - mounted at boot time on the new VM
- `context.sh` (with ONe-generated context) and possibly `init.sh` (with custom context) are executed at boot time
  - By `/etc/one-context.d/00-network`

CONTEXTUALIZATION

```
CONTEXT=[

  SSH_PUBLIC_KEY="ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEApph23\
  K7jDUqm24mnewnnJkFgY/W8mIfytrKEIADUp9I5ksxOgMwRFkYvWmhUAY\
  CCkagTxQwFPvNAJbkdEYBEVvpCSw2Nfbb8uOMZovIoBEziV7cZAn2SDrT\
  XfaGUOblR14a9THoK7P2FpoXn86lFDcfrXcEnU3i2qoRaJ4cW+QLSIzRr\
  zI4NnFfLdbFQetrU2U+Bjbtytfq+alwmSUnfY1FWcmKtzCOJ06fCZik1d\
  Q6gKgDcuSzEHVKIi9Jc7HrWNfXhsLlqdnNbC9WbCmDvSlOCAQ1Gl12WHz\
  tHBJJQGfrJP5u76n+C0WB8scS4yaHaYf5w+xfvmgXVw==\
  oneadmin@one-master.to.infn.it",

  FILES="/var/lib/one/devel/context/TestVM/init.sh\
         /var/lib/one/devel/context/TestVM/bagnasco.pub”,

  TARGET="vde”

]
```

When instantiating a new VM...

- Start from a very basic OS image and use tools to configure it
  - Decouple OS from application
  - Maintain only a limited number of images
  - Basic OS images are small
  - Contextualization scripts can be complex and difficult to maintain
  - Complex contextualizations can be (very) slow
- Instantiate an OS image, configure it, save it and re-use it for subsequent instantiations
  - Short VM start-up time
  - Saved images are always consistent
  - Large number of images to maintain
  - Fully configured images can be very large

NOW...

- More about this after the coffee break
  - With live demo!

Questions?

Stefano.Bagnasco@to.infn.it