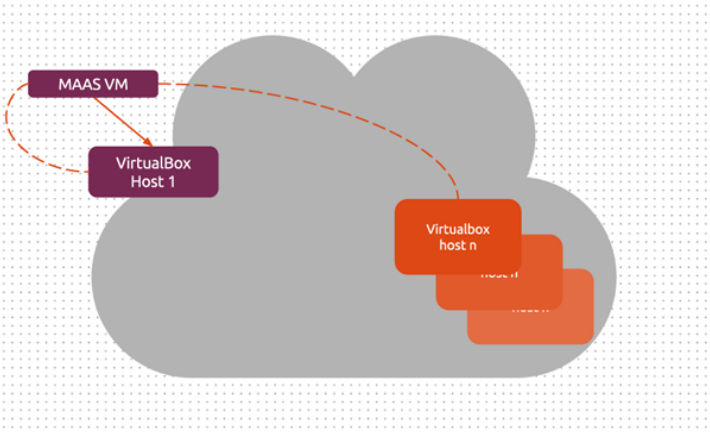
VirtualBox extensions for MAAS

By Ivan Zoratti on 15 January 2015

Share or save



During the last season's holidays, I spent some time cleaning up demos and code that I use for my daily activities at **Canonical** (http://www.canonical.com/). It's nothing really sophisticated, but for me, and I suspect for some others too, a small set of scripts makes a big difference.

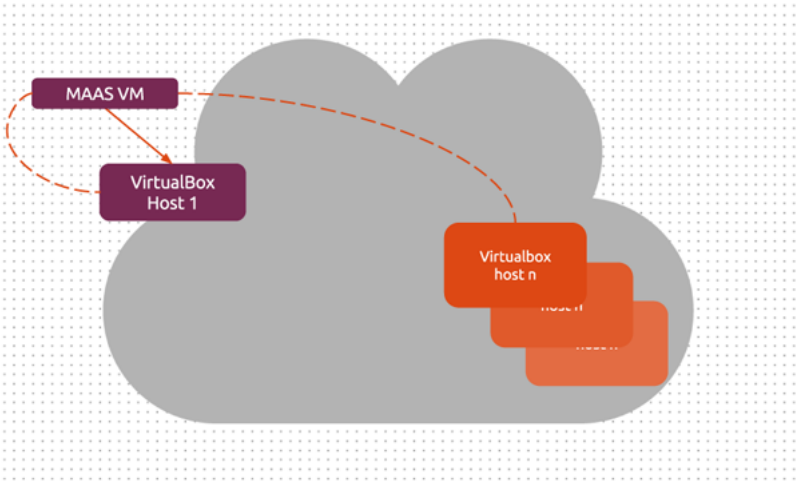
In my daily job, I like to show live demos and I need to install a large set of machines, scale workloads, monitor and administer servers and data centres. Many people I meet don't want to know only the theoretical details, they want to see the software in action, but as you can imagine, the process of spinning up 8 or 10 machines and install and run a full version of OpenStack in 10-15 minutes, while you also explain how the tools work and perhaps you even try to give suggestions on how to implement a specific solution, is not something you can handle easily without help. Yet, that is what CTOs and Chief Architects want to know in order to decide whether a technology is good or not for them.

At Canonical, workloads are orchestrated, provisioned, monitored and administered using **MAAS** (http://maas.ubuntu.com/), **Juju** (http://juju.ubuntu.com/) and **Landscape** (https://landscape.ubuntu.com/), around **Ubuntu Cloud** (http://www.ubuntu.com/cloud), which is the Canonical **OpenStack** (http://www.openstack.org/) offering. These are the products that can do the magic of what I described, but providing in minutes something that usually takes days to install, set up and run.

In addition to this long preface, I am an enthusiastic Mac user. I do love and prefer Ubuntu software and I am not entirely happy with many technical decisions around OS X, but I also found Mac laptops to be a fantastic hardware that simply fits my needs. Unfortunately, the **KVM** (http://www.linux-kvm.org/) porting to OS X **is not available yet** (https://github.com/fanwenyi0529/fvm), hence the easiest and most stable way to spin up Linux VMs in OS X is to use **VMWare Fusion** (http://www.vmware.com/uk/products/fusion), **Parallels** (http://www.parallels.com/) or **VirtualBox** (https://www.virtualbox.org/). Coming from Sun/Oracle and willing to use open source software as much as I can, VirtualBox is my favourite and natural choice.

Now, if you mix all the technologies mentioned above, you end up with a specific need: the integration of VirtualBox hosts, specifically running on OS X (but not only), with Ubuntu Server running MAAS. The current version of MAAS (1.5 GA in the Ubuntu archives and 1.7 RC in the maintainers branch), supports virsh for power management (i.e. you can use MAAS to power up, power check and power down your physical and virtual machines), but the VirtualBox integration with virsh is limited to socket communication, i.e. you cannot

connect to a remote VirtualBox host, or in other words MAAS and VirtualBox must run in the same OS environment.



(http://insights.ubuntu.com/wp-content/uploads/226b/image1_Ivan.png)
Connections to local and remote VirtualBox hosts

My first instinct was to solve the core issue, i.e. add support to remote VirtualBox hosts, but I simply did not have enough bandwidth to embark on such an adventure, and becoming accustomed to the virsh libraries would have taken a significant amount of time. So I opted for a smaller, quicker and dirtier approach: to emulate the most simple power management features in MAAS using scripts that would interact with VirtualBox.

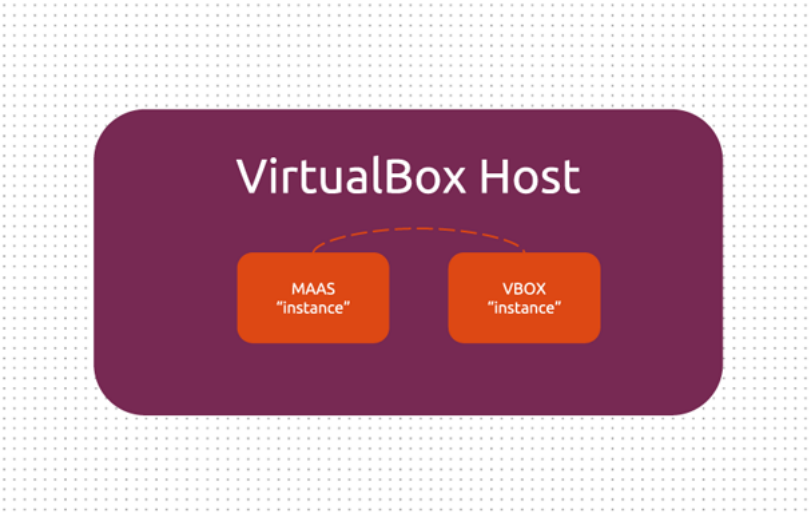
MAAS (http://maas.ubuntu.com/) – Metal As A Service, the open source product available from Canonical to provision bare metal and VMs in data centres, relies on the use of templates for power management. The templates cover all the hardware certified by Canonical and the majority of the hardware and virtualised solutions available today, but unfortunately they do not specifically cover VirtualBox. For my workaround, I modified the most basic power template provided for the Wake-On-LAN option. The template simply manages the power up of a VM, and leaves the power check and power down to other software components.

The scripts I have prepared are available on my **GitHub account** (https://github.com/izoratti/MAAS_VirtualBox), and are licensed under GPL v2, so you are absolutely free to download it, study it, use it and, even more important, provide suggestions and ideas to improve them.

The README file in GitHub is quite extensive, so I am not going to replicate here what has been written already, but I am going to give a wider architectural overview, so you may better consider whether it makes sense to use the patches or not.

MAAS, VirtualBox and OS X

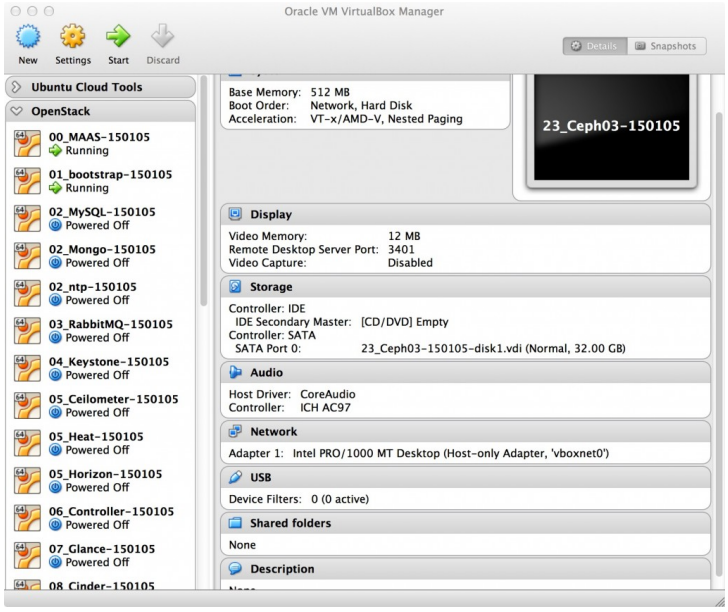
The testing scenario that I have prepared and used includes OS X (I am still on Mavericks as some of the software I need does not work well on Yosemite), VirtualBox and MAAS. What I need for my tests and demos is shown in the picture below. I can use one or more machines connected together, so I can distribute workloads on multiple physical machines. The use of a single machine makes things simpler, but of course it puts a big limitation to the scalability of the tests and demos.



(http://insights.ubuntu.com/wp-content/uploads/226b/image2_Ivan.png)

A simplified testbed with MAAS set as VM that can control other VMs, all within a single OS X VirtualBox Host machine

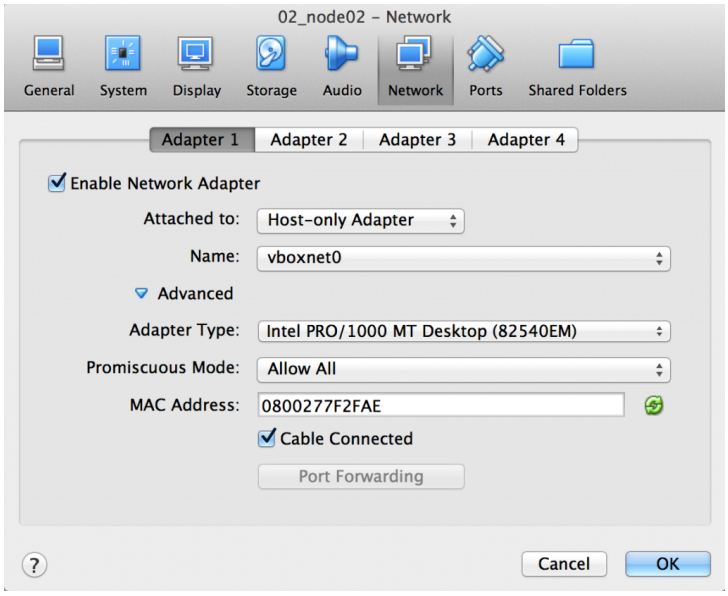
The basic testbed I need to use is formed by a set of VMs prepared to be controlled by MAAS. The VMs are visible in this screenshot of the VirtualBox console.



(http://insights.ubuntu.com/wp-content/uploads/226b/03-VirtualBox-Console.jpg)

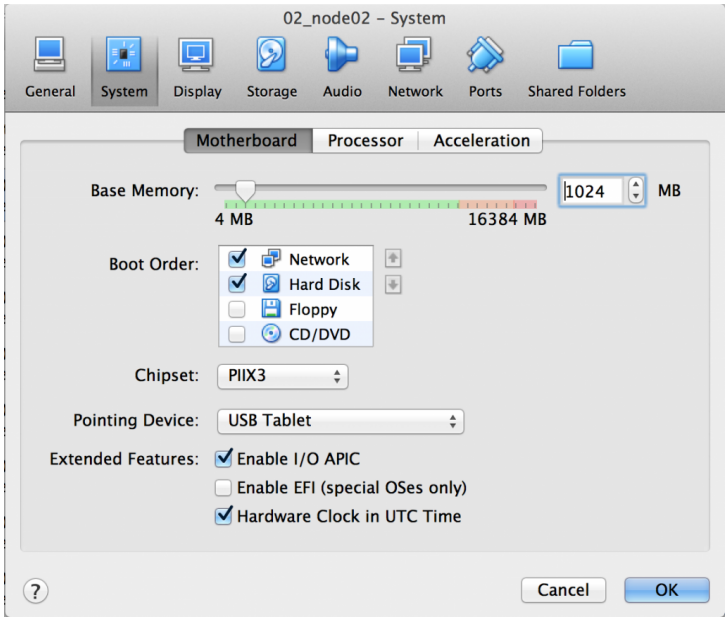
VirtualBox Console

Two aspects are extremely important here. First, the VMs must be connected using a network that allows direct communication between the MAAS node and the VMs. This can be achieved locally by using a host-only adapter where MAAS provides DNS and DHCP services and each VM has the Allow All option set in the communication mode combo.



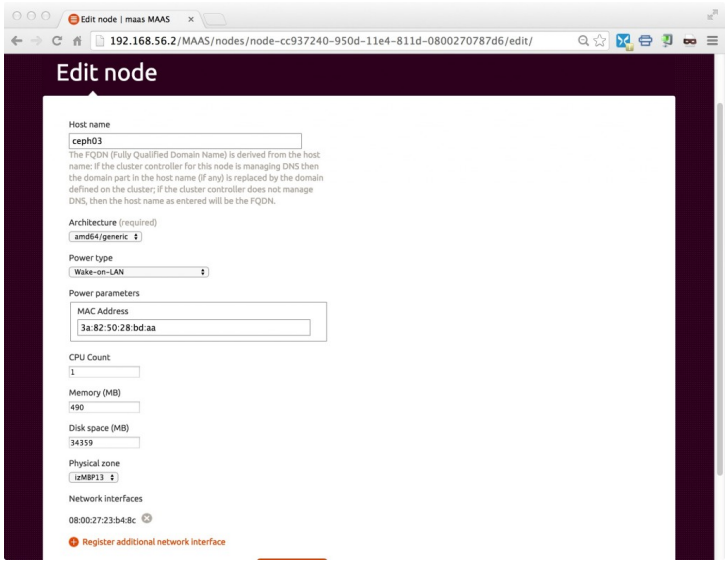
(http://insights.ubuntu.com/wp-content/uploads/226b/04-VirtualBox-Network.png)

Secondly, VMs must have PXE boot set on. In VirtualBox, this is achievable by selecting the network boot option as the first option available in the system tab.



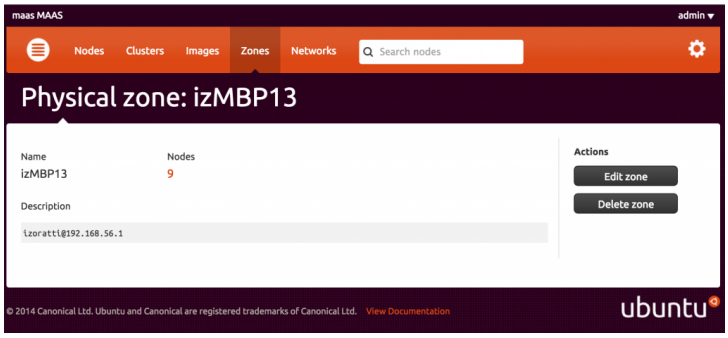
(http://insights.ubuntu.com/wp-content/uploads/226b/05-VirtualBox-Boot-Options.png)

In this way, the VMs can start the very first time and can PXE Boot using a cloud image provided by MAAS. Once MAAS has the VM enlisted as a node, administrators can edit the node via the WEB UI, the CLI app or the RESTful API. Apart from changing the name, what is really important is the setting of the Power mode and the physical zone. The power mode must be set as *Wake-On-LAN* and the MAC address is the last part of the VM id in VirtualBox (with colons). The Physical zone must be associated to the VirtualBox Host machine.



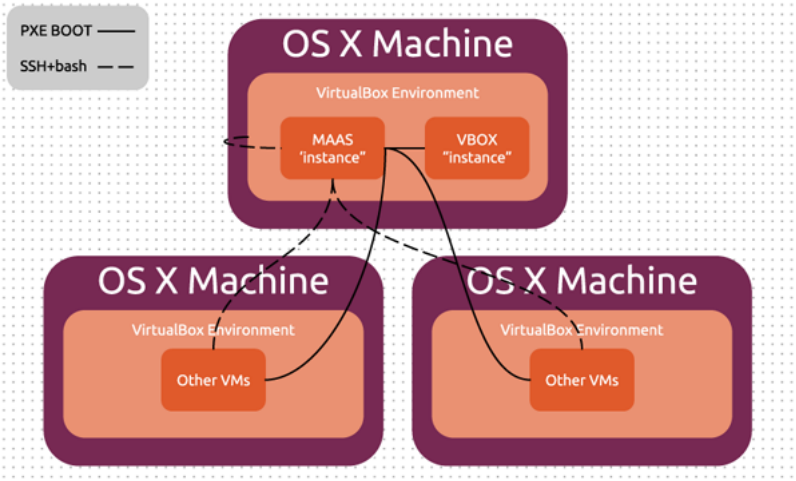
(http://insights.ubuntu.com/wp-content/uploads/226b/06-MAAS-Edit-Node.jpg)

In the picture above the *Physical zone* is set as izMBP13. The description of the *Physical zone* must contain the UID and the hostname or IP address of the host machine.



(http://insights.ubuntu.com/wp-content/uploads/226b/07-Physical-Zone.png)

Once the node has been set properly, it can be commissioned by simply clicking the *Commission node* button in the *Node* page. If the VM starts and loads the cloud image, then MAAS has been set correctly.



(http://insights.ubuntu.com/wp-content/uploads/226b/image3_Ivan.png)

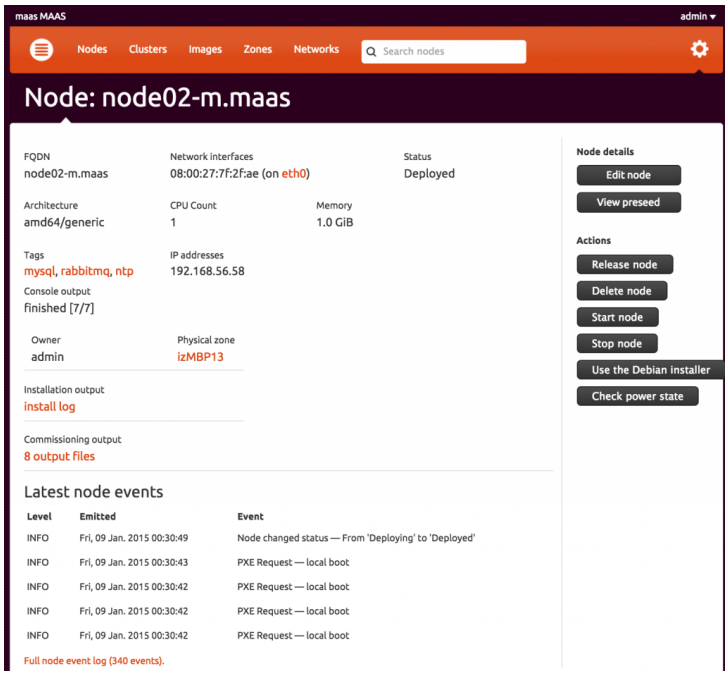
The MAAS instance interacts with the VirtualBox host via SSH and with responds to PXE Boot requests from the VMs

A quick look at the workflow

The workflow used to connect MAAS and VM is relatively simple. It is based on the components listed below.

A. MAAS control

Although I have already prepared scripts to *Power Check* and *Power Off* the VM, at the moment MAAS can only control the *Power On*. Power On is executed by many actions, such as *Commission node* or the explicit *Start node* in MAAS. You can always check the result of this action by checking the event log in the *Node* page.



(http://insights.ubuntu.com/wp-content/uploads/226b/09-MAAS-Node.png)

B. Power template

The Power On action is handled through a template, which in the case of Wake-On-LAN and of the patched version for VirtualBox is a shell script.

The small fragment of code used by the template is listed here and it is part of the file */etc/maas/templates/power/ether_wake.template*:

```
...
if [ "${power_change}" != 'on' ]
then
...
elif [ -x ${home_dir}/VBox_extensions/power_c
then
    ${home_dir}/VBox_extensions/power_on \
    $mac_address
...
fi
...
```

C. MAAS script

The script *${home_dir}/VBox_extensions/power_on* is called by the template. This is the fragment of code used to modify the MAC address and to execute a script on the VirtualBox Host machine:

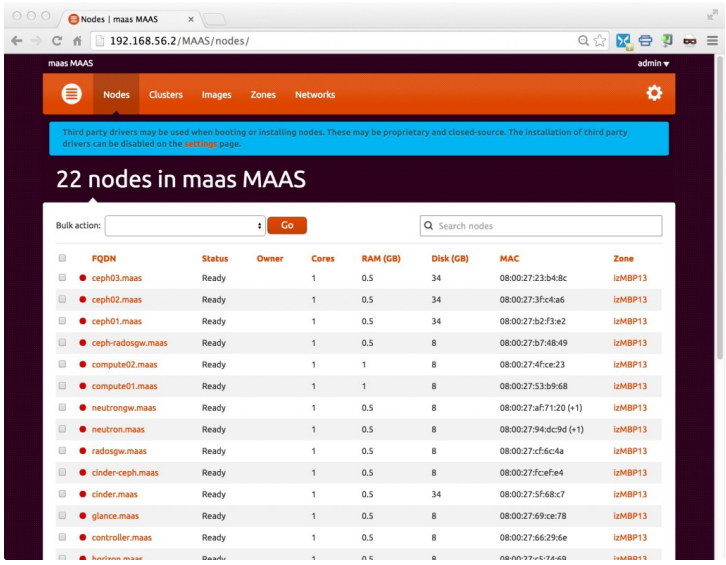
```
...
vbox_host_credentials=${zone_description//\"}

# Check if there is the @ sign, typical of ss
# user@address
if [[ ${vbox_host_credentials} == *"@"* ]]
then
  # Create the command string
  command_to_execute="ssh \
    ${vbox_host_credentials} '~/VBox_host_ext
  # Execute the command string
  eval "${command_to_execute}"
...
```

D. VirtualBox host script

The script in *~/VBox_host_extensions/startvm* is called by the MAAS script and executes the stratvm command locally:

```
...
start_this_vm=`vboxmanage list vms | grep "${
start_this_vm=${start_this_vm#*\{}
start_this_vm=${start_this_vm%\}*}
VBoxManage startvm ${start_this_vm} --type he
...
```

The final result will be a set of VMs that are then ready to be used for example by Juju to deploy Ubuntu OpenStack, as you can see in the image below.



(http://insights.ubuntu.com/wp-content/uploads/226b/10MAAS-Nodes-Ready.jpg)

Next Steps

I am not sure when I will have time to review the scripts, but they certainly have a lot of space for improvement. First of all, by adopting a richer power management option, MAAS will not only power on the VMs, but also power off and check their status. Another improvement regards the physical zones: right now, the scripts loop through all the available VirtualBox hosts. Finally, it would be ideal to use the standard virsh library to interact with VirtualBox. I can't promise when, but I am going to look into it at some point this new year.

About the author

Ivan Zoratti is a Senior Solutions Architect at Canonical. He has a background in Enterprise and carrier-grade distributed systems, specifically in transactional databases and Big Data. At Canonical, he has a strong focus on cloud technologies including OpenStack, Juju and MAAS.