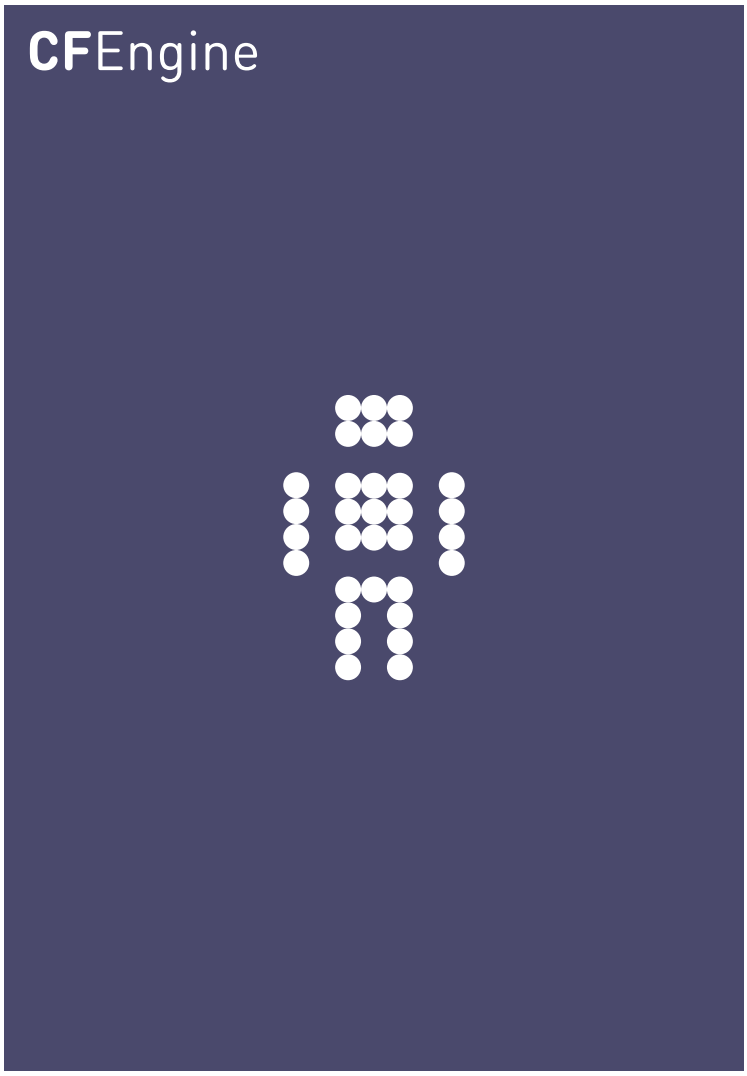

Virtualization and Cloud Support in CFEngine

A CFEngine Special Topics Handbook

CFEngine AS

CFEngine Nova integrates simply with existing frameworks for virtualization and cloud computing, allowing you to apply convergent 'self-healing' methods to the deployment and management of virtual machines running anywhere.

Virtualization

Table of Contents

Virtualization 1

What are virtualization and cloud computing?

Virtualization refers to the ability to run multiple host instances on a single physical node. Cloud computing typically refers to what is called 'platform as a service', or deployment of virtual machines on demand, often as an online service.

In this document, virtualization support refers specifically to hypervisor technologies supported by the open source library layer *libvirt* project, which includes interfaces for Xen, KVM, Vmware-ESX, and more. CFEngine thus integrates freely with other tools based on this library, such as *virsh* and the *Virtual Manager* graphical user interface.

Why build virtualization support into CFEngine?

Virtualization engines (usually called supervisors or hypervisors) are seeing an explosion of development. They exist as a number of projects in various stages of maturity. The libvirt project was designed as an integration layer based on an XML specification.

The tools for management are still quite primitive and require much manual work. CFEngine has a unique role to play in maintaining desired state in virtual machine systems.

In the cloud, virtual machines may be rented from remote commercial providers, and managed as disposable resources. Convergent or 'self-healing' maintenance is an essential method for managing machines that are geographically remote and awkward to access, e.g. machines in other time-zones that it is impractical to monitor by legacy methods.

What can CFEngine do with virtual machines?

The simple answer is: anything that *libvirt* can do, with added convergence to a desired state: that means, creating, destroying and starting and stopping machines. By starting virtual machines through CFEngine, you can be sure that a given 'virtual guest' is running on one and only one physical host, thus avoiding conflicts that are difficult to detect with centralized systems.

CFEngine does not support everything that libvirt does – it offers a simplified interface that is meant for robustness, stability and hands-free repeatability.

> CFEngine does not use libvirt's TLS based web communication layer. It manages every host as an independent entity, in typical CFEngine fashion, using CFEngine's own distributed cooperation to provide thje implicit communication. CFEngine does not currently support so-called 'live migration' of virtual machines.

Guest environments promises

A virtual machine is one example of what CFEngine calls an 'guest_environment'. You can promise to create (and host) an guest environment with certain attributes, just as you can promise to host a file or a process. Here is a simple example:

```
body common control
{
bundlesequence  => { "my_vm_cloud" };
}

########################################################

bundle agent my_vm_cloud
{
guest_environments:

   "myUbuntu" # the running instance name, defined in XML

       environment_resources => virt_xml,
       environment_type      => "xen",
       environment_host      => "my_physical_computer", # ipv4_10_1_2_3
       environment_state     => "create";
}

########################################################

body environment_resources virt_xml
{
env_spec_file => "/srv/xen/centos5-libvirt-create.xml";
}
```

- The promiser (in this case '`myUbuntu`') is the name of the virtual machine. This should be a unique identifier, as we need to be able to refer to machines uniquely.
- The guest environment host is the name of the computer that is the host for the virtual machine.
- Normally when we want to ensure something on a machine, we use classes to decide where the promise will be made. For guest environments, however, we need to make promises about the uniqueness of the machine. When you make a machine instance you normally want it to be running on one and only one host. So you want *every* machine to make a promise. On the guest environment's host, you want to promise that the guest environment is running, and on every other machine you want to promise that it is not. In CFEngine, you simply include a unique class belonging to host in the promise using `environment_host` and CFEngine assumes that rest. Unique classes might include
  - Hostname class e.g. `myhost_CFEngine_com`
  - IP address class e.g. `ipv4_123_456_789_123`

An alternative way to write this example is to quote the XML specification in CFEngine directly. This has a few advantages: you can re-use the data and use it as a template, filling in CFEngine-variables. You can thus adapt the configuration using CFEngine's classes.

```
bundle agent my_vm_cloud
{
guest_environments:

   "myUbuntu" # the running instance name, defined in XML
       environment_resources => virt_xml("$(this.promiser)"),
       environment_type      => "xen",
       environment_host      => "myphysicalcomputer";
       environment_state     => "create"
}

#########################################################

body environment_resources virt_xml(host)
{
env_spec_file =>

"<domain type='xen'>
  <name>$(host)</name>
  <os>
    <type>linux</type>
    <kernel>/var/lib/xen/install/vmlinuz-ubuntu10.4-x86_64</kernel>
    <initrd>/var/lib/xen/install/initrd-vmlinuz-ubuntu10.4-x86_64</initrd>
    <cmdline> kickstart=http://example.com/myguest.ks </cmdline>
  </os>
  <memory>131072</memory>
  <vcpu>1</vcpu>
  <devices>
    <disk type='file'>
      <source file='/var/lib/xen/images/$(host).img'/>
      <target dev='sda1'/>
    </disk>
    <interface type='bridge'>
      <source bridge='xenbr0'/>
      <mac address='aa:00:00:00:00:11'/>
      <script path='/etc/xen/scripts/vif-bridge'/>
    </interface>
    <graphics type='vnc' port='-1'/>
    <console tty='/dev/pts/5'/>
  </devices>
</domain>
";
}
```

You should consult the libvirt documentation for the details of the XML specification.

Virtualization types supported

CFEngine currently supports virtualization only through libvirt, so it supports those technologies that libvirt supports. Currently this includes most popular technologies. You must choose the type of monitor that is to be responsible for keeping the guest environment promise. In CFEngine, you should choose between a machine environment or network environment of the following types:

`xen` A Xen hypervisor virtual domain.

`kvm` A KVM hypervisor virtual domain.

`esx` A VMware hypervisor virtual domain.

`test` The libvirt test-hypervisor virtual domain.

`xen_net` A Xen hypervisor virtual network.

`kvm_net` A KVM hypervisor virtual network

`esx_net` An ESX/VMWare hypervisor virtual network.

`test_net` The test hypervisor virtual network.

`zone` A Solaris zone (future development)

`ec2` An Amazon EC2 instance (future development)

`eucalyptus`
A Eucalyptus instance (future development)

Once again, you must consult the libvirt documentation for details.

Distinct states

Libvirt recognizes a number of distinct states are transliterated into CFEngine as

`create` Build and start an guest environment.

`delete` Halt and remove runtime resources associated with an guest environment.

`running` An existing guest environment is in a running state.

`suspended`
An existing guest environment is in a ‘paused’ state.

`down` An existing guest environment is in a halted state.

The default promised state is for a machine to be running wherever the `environment_host` class is true, and suspended or down elsewhere.

Example deployment

Prerequisites: you need to make a 'disk image' for the machine, or a virtual disk of blocks that can be allocated. This image does not have to contain any data, it will simply as a block device for the VM. You can then install it by booting the machine from a network image, like a PXE/kickstart installation.

If you want to allocate disk blocks as the file grows, you can create a file with a hole. The following command will creates a file of 2048MB, but the actual data blocks are allocated in a lazy fashion:

```
# dd if=/dev/zero of=/srv/xen/my.img oflag=direct bs=1M seek=2047 count=1
```

To reserve all the data blocks right away:

```
# dd if=/dev/zero of=/srv/xen/my.img oflag=direct bs=1M count=2048
```

Libvirt uses an XML file format that cannot be circumvented. CFEngine promises to honour the promises that are expressed in this file, as in the examples above. You need to find out about this file format from the libvirt website. To get CFEngine to honour these promises, you point it to the specification that it should promise using `spec_file`.

You need to set up a network for virtual machines to communicate with the outside world. This can also be done with CFEngine, using the network promise types to build a bridge into a virtual network.

Then just run CFEngine to start, stop or manage the guest environments on each localhost. Run in verbose mode to see how CFEngine maintains the states convergently.

```
# cf-agent -v
```

Appendix A Virtualization Examples

Virtualized host examples

```
#########################################################

body common control
{
bundlesequence => {"my_vm_cloud"};
host_licenses_paid => "2";
}

#########################################################

bundle agent my_vm_cloud
{
guest_environments:

 linux::

 "bishwa-kvm1"
                 comment => "Keep this vm suspended",
   environment_resources => myresources,
        environment_type => "kvm",
       environment_state => "suspended",
        environment_host => "ubuntu";

 "bishwa-kvm2"

                 comment => "Keep this vm running",
   environment_resources => myresources,
        environment_type => "kvm",
       environment_state => "running",
        environment_host => "ubuntu";

 "bishwa-kvm3"
                 comment => "Power down this VM",
   environment_resources => myresources,
        environment_type => "kvm",
       environment_state => "down",
        environment_host => "ubuntu";

 "bishwa-kvm4"

                 comment => "Delete this VM",
```

```
   environment_resources => myresources,
        environment_type => "kvm",
       environment_state => "delete",
        environment_host => "ubuntu";

 "bishwa-kvm5"
                 comment => "Keep this vm running",
   environment_resources => myresources,
        environment_type => "kvm",
       environment_state => "running",
       environment_host => "ubuntu";

}


#############################################################

body environment_resources myresources
{
env_cpus => "2";
#env_memory => "512"; # in KB
#env_disk => "2048";  # in MB
}
```

Virtual network example

```
body common control
{
bundlesequence => {"my_vm_cloud"};
host_licenses_paid => "2";
}

#########################################################

bundle agent my_vm_cloud
{
guest_environments:

linux::

"cfvrnet1"
comment => "Create a virtual network from given xml file",
environment_resources => virt_xml("$(this.promiser)"),
environment_type      => "kvm_net",
```

```
environment_state     => "create",
environment_host      => "ubuntu";
}

#########################################################
body environment_resources virt_xml(name)
{

 env_spec_file =>"
 "
  <network>
  <name>$(name)</name>
  <bridge name='virbr1' />
  <forward mode='route' dev='eth0'/>
  <ip address='192.168.123.1' netmask='255.255.255.0'>
  <dhcp>
  <range start='192.168.123.2' end='192.168.123.254' />
  </dhcp>
  </ip>
  </network>
  ";
}
```