Repair cloud virtual machine cloning errors

Resolve the issue with Runtime Image Activation

Roberto Ragusa
Claudio Marinelli
Luigi Pichetti
Alex Donatelli

February 20, 2012

A benefit of virtualization and virtual systems is that you can clone them to be reused in different environments. External data provisioning requirements such as network configurations like IP addresses can cause problems when cloning a virtual machine to use in a new environment. If the external data is not available during the process, the reconfiguration of the VM will likely incomplete. The authors offer a way to handle this problem, even without much knowledge of the application or without a form of activation scripting to help. Runtime Image Activation (RIA) is a prototype command-line interface that lets you orchestrate networking techniques to make sure your cloned VMs are appropriately configured.

One of the benefits of exploiting virtual system images is the capability to clone them so they can be reused in different environments. This usually requires some effort in order to reconfigure the software applications the images contain. The reconfiguration issue is well known when you clone VMs with pre-installed software, particularly when you're working with applications that implement server-side services (such as DBserver, AppServer, etc), which typically may not support DHCP and are listening incoming TCP requests on a static address.

There is an existing process to resolve this image reconfiguration dilemma that is known as the Image Re-Activation process. It exploits such techniques as using an activation engine (like the IBM Activation Engine, an enablement framework used for boot-time customization of virtual images) and application-specific activation scripts. The problem with using the Image Re-Activation process in these circumstances is that it assumes that for all the misconfigured applications, there is a proper script that can be run to reconfigure everything correctly.

In this article, we'll describe an alternative way to try to resolve the issue which may come handy in cases where there is neither knowledge of the application in terms of which configuration artifacts were impacted by the original network configuration, nor availability of some pre-built activation scripting to alter such application-specific configuration artifacts. Incomplete information about the environment can happen when it is not known which applications are present in the VM or there is
no re-enablement script available for them (the application may not be a popular one, or it may be a very recent one, one that's not documented, you can't reverse-engineer it, etc.).

In such cases, when the focus is to address the networking misconfiguration of the contained software stack that occurred as a result of the VM cloning, the method we propose could have a more valuable result in re-enabling the VM. Our proposed approach is application agnostic; we call it Runtime Image Activation (RIA). We've prototyped a sample RIA command line interface that will let you orchestrate or implement the networking techniques described in this article; those allow you to solve the problem of cloning misconfiguration.

Perfect cloning is not always a good thing

Machine provisioning is a central function in cloud-oriented environments; new physical machines are brought online or new virtual machines are created on a very regular basis. It often happens that runtime images (that is, the entire content of the disks) are cloned from an existing machine or from a repository of useful templates.

A recurring problem when cloning runtime images is that they contain some information that is dependent on the external environments, such as network configuration IP addresses. The operating system and the included applications will probably not be able to work properly without the needed reconfigurations. Typical errors include:

- Failures when attempting to bind on a network socket.
- Failure when trying to contact a required network service.
- Exposing duplicate IP address on the network will certainly cause serious problems, such as random connection failures, disconnections, and general network instability.

The information about which IP addresses a machine is configured to use can be found in the configuration files of the operating system. The first step to be done after cloning a VM is to substitute the old IP address with the new one. You can automate this task with scripts and tools that work directly on the content of the runtime image in a way that is dependent on the specific operating system in use; for example, it can replace strings in /etc files for Linux®/UNIX® systems or change registry entries on Windows® systems.

It often happens that installed apps contain some internal configuration derived from the network configuration and captured at install time; for example, a web server could have saved something into its own configuration data, like the IP and/or hostname it is going to advertise when serving HTTP/HTTPS requests. One way of fixing this would be by reinstalling the application; that would basically mean going from a "clone a preconfigured machine" to an "install the machine from scratch" approach. This is annoying and inefficient — this negates all the cloning advantages such as being able to create virtual machines in a quick, reliable, and consistent way.

So what is the traditional approach to enable you to reactivate the software in VMs after they've been cloned?
A traditionalist approach: Fix all the configs

The way we normally approach this task is to surgically modify the incorrect configuration wherever it is stored. On a case-by-case, individualized basis. You can understand the work involved following this method. It implies a very precise understanding of which applications are expected to be present and which kind of operations have to be done on each of them. Said operations can be run as scripts on the mounted (not running) image or as first-boot or agent level in the running machine.

Assuming that you possess perfect knowledge of what has to be done, the final result turns out optimal: the application appears no different than one you have just "correctly" installed.

On the other hand, this method is fragile; small detail variations (like "could the user have moved the config to another path?", "is this version of the software still using the same config storage as the old one?" and so on) make this approach a potential nightmare of record-keeping.

Attempts to proceed to correct the configurations without application-specific knowledge is prone to fail. Scanning the entire image and replacing all occurrences of the old IP (in textual or binary form) with the new one should never be seriously considered by anyone who understands Murphy's Law.

There's a better strategy ... we call it "playing network tricks."

A new strategy: Network tricks

An alternative and innovative approach to fixing cloned configuration errors without specific knowledge of the software involved is the one we are now going to describe. First, let's set up the scenario:

Let's consider the frequent case of machines that are exporting services through externally reachable daemons; at install time, the daemons have unluckily saved in their configuration the IP addresses on which they are going to listen. Such daemons will probably fail to start in the cloned machine since they fail to bind themselves to a network interface.

Instead of trying to fix the configuration of the application, we can try to reshape the networking environment to trick the application to operate with the incorrect configuration. The operating system of the machine itself will assist us in this objective.

The general idea

Figure 1 describes the entire process.
Figure 1. General description of the solution

On the left is the original machine (called Source machine) that is cloned to create the machine on the right (called Target machine; see orange dot 1). There were three applications installed inside the Source machine at the moment it was cloned. Suppose that app1 is able to automatically detect the IP assigned to the machine, but app2 and app3 are using IP addresses saved in their own configuration files (textual or database).

The problem for the Target machine is that after changing the old IP address (IPS) to the new one (IPT; see orange dot 2), external clients are only able to reach app1; both app2 and app3 are unavailable because they were unable to start since the binding on an interface with address IPS failed.

What should you do?

An additional dummy network interface, perhaps

As a first step, create an additional network interface; you can create as many interface aliases as you want in Linux. For example, you can create an additional loopback interface (in addition to the usual lo one, configured as 127.0.0.1/8) and assign the old IP to it (see orange dot 3; Figure 1). The additional interface can be generated as an alias of lo and it will be identified as lo:0. As an alternative, it is also possible to generate eth0:0, an alias of the eth0, device.

After doing this, the incorrectly configured application will be able to start and bind on this fake interface. Accepting connections from external machines is still impossible; we expect external hosts will try to use the new IP, so the packets will reach our cloned machine and immediately be discarded since no process is listening on the interface with the new IP.

Figure 2 shows a new address (IPT) of 192.168.31.9 and an old address (IPS) of 10.10.9.9.
Repair cloud virtual machine cloning errors

Fig. 2. A server starts and binds to the fake address

There is a daemon that was able to start by binding to the old, “wrong” IP.

Next we’ll look at redirection of the Network Address Translation at the kernel level.

**NAT redirection at kernel level**

As a second step, ask the operating system to intercept all connections incoming to the new IP and silently, internally redirect them to the old IP (yes, this is not a typo ... redirect the NEW address to the OLD address).

This redirection is easily done at the kernel level as a Network Address Translation rule; NAT redirection has no measurable performance implications since it is just a well-known common trick used on firewalls and gateways. The connection broker (see orange star in Figure 1) plays this role.

You’ll also need to set the routing rules of the client machines and any intermediate node in a way to let the traffic directed to the old IP actually reach the machine.

Last thing: You need to redirect only specific ports.

**NAT-providing watchdog daemon**

You have to know which ports to redirect because a general newIP-to-oldIP rule would break all the applications that are correctly listening on the new IP (for example, those that were smart enough to detect IP addresses at runtime instead of reading their own outdated configs).

You still want to avoid any application-specific knowledge (we are, in fact, not even assuming to know which applications we are trying to fix), so you need to discover used ports in an automatic way. To achieve this, ask the operating system for the list of ports on which someone is listening via the interface with the old IP.
This list is obtained on Linux with the `netstat -l` option; arrange NAT redirection for those ports.

For increased robustness, repeat this scan and redirect phase every few seconds so you are able to cope with slow-starting applications or applications that dynamically bind and unbind ports.

The watchdog daemon (orange dot 4 and 5, Figure 1) has the role to perform the scan and drive the broker.

Figures 3 and 4 show the watchdog in action.

**Figure 3. The watchdog in action; watchdog creates a redirection rule**

You can see that the watchdog detected someone listening on 10.10.9.9:80 and created a redirection rule to transparently hijack the traffic directed to 192.168.31.9:80 towards 10.10.9.9:80.

**Figure 4. The client successfully establishes a connection**
An external client is now able to connect to the server by pointing at 192.168.31.9:

- From the point of view of the client, the machine has a new address (192.*).
- From the point of view of the server, the machine still has the old address (10.*) yet it can keep serving incoming network requests as it was prior the VM was cloned.

In conclusion, this approach consists of

- Setting a dummy interface with the old IP so that misconfigured daemons can start without networking issues.
- At the same time, setting up a watchdog daemon that polls the listening table against the dummy interface with the old IP; as listeners are discovered there, the watchdog daemon arranges port redirection from the new IP to the old IP.

The old IP is not visible to external clients which will expect the server to be properly configured with the new IP. The daemons will work both by binding to the new IP and by binding to the old IP, thanks to the NAT trick done performed by the operating system.

But there are a few more tricks of a reconfiguration nature you can do to make the solution more complete.

**Complementing the strategy: Configuration tricks**

While the IP aliasing mechanism described addresses most of our goals, there are additional configuration tricks that can be also orchestrated to provide a more complete solution:

- ARP filtering (ARP is the Address Resolution Protocol)
- Name resolution and DNS proxy
- `getHostName()` hook

**ARP filtering**

The configuration of fake interfaces on Linux hosts can have an unpleasant side-effect. Even if the fake interface is a local one (`lo:0` instead of `eth0:0`), ARP requests are by default replied on any interface for all IPs configured on any interface of the machine.

This means that the fake IP is in some way visible (and actually pingable) from the clients. A perfect stealth operation is desirable, especially if you want to make several clones or copies of the source machine in the same network segment.

You can do this on Linux by changing the `sysct1` network parameter `arp_ignore` to 1 in order to avoid the default behavior (0 means to reply to all ARP requests).

**Name resolution and DNS proxy**

In case app2/app3 daemons depicted in Figure 1 in the cloned VM have kept a static reference to the old hostname (say HNS), in addition to or in place of the old IP (IPS), you will also need to influence the name-resolution chain in order to return the correct IP.
This can be achieved updating the local /etc/hosts configuration file so that any inquiry for the old hostname (HNS) returns the new IP (IPT). In case the name-resolution process exploits an external name server, you can install a local DNS proxy configured to be the first-in-chain name server entitled to answer, which will reply coherently with the modified /etc/hosts file.

**Using a getHostName() hook**

An additional configuration strategy that may come in handy is the "hooking" — grabbing the function call — of the getHostName() function so that it returns modified values (in other words, the old hostname).

This has been prototyped and can be nested at different user levels (for example, root-wide or to affect processes running with a specific and lower user privilege).

**Putting it all together**

To facilitate the overall configuration process, there is a prototyped RIA command line for Linux that is able to apply the configuration techniques as we've described in this article.

Here's the syntax and sample usage in a RHEL virtual system:

```
./ria start -oh <oldhostname> -oi <old_ipv4> -ni <new_ipv4> [options]
./ria stop | status
./ria start -oh oldhostname.domain.com -oi 10.10.1.1 -ni 1.2.3.4 -ipalias -hnresolv -dnat
```

**In conclusion**

A real-world implementation of this technique proved very successful. We tested on web server (Apache) and application servers (IBM WebSphere® Application Server) hosting environments that had to be installed with a static IP configuration and which, after VM cloning, would not work anymore because the software stack was still trying to bind to, listen to, or accept the old IP configuration stored in application specific configuration artifacts.

The RIA approach circumvented the problem, letting the server software stack work in the new environment even if it continued to think it was still living in the old environment. The approach requires regression of the main product use cases.

Keep in mind that the Image Re-Activation approach is always a viable alternative to explore; it's a "surgical" and definite kind of approach since it has the goal to "remove" the old application configuration and replace it with the new one. It can be expensive and sometimes is not the best approach.

Yet, in all cases where there is no knowledge, no investment plan, or no documented way to alter the embedded configuration of software applications, you should consider the RIA application-agnostic approach.

One possible objection users could raise: **What happens during the seconds when the application has been started but the NAT daemon has not arranged the redirect yet?**
The answer is that the service simply seems unavailable to external clients; from the external point of view, the service just appears to have successfully started a few seconds later. No significant negative effect can be seen when using reasonable polling time, such as polling once per second for a few minutes (to promptly catch stuff starting up) and then relaxing the polling to once per 5-10 seconds.

The Runtime Image Activation method can be easily embedded into cloud provisioning software, and it has been successfully prototyped using the Tivoli® Cloud Management stack (Tivoli Service Automation Manager, TSAM). Knowledge of old IP (that you can specify when images get imported and registered into the cloud deployment tool) and of new IPs (which the cloud deployment tool dynamically generates and exploits in appropriate deployment workflows) is the only element you need to be aware of when applying the feature to the virtual machine deployment process.

© Copyright IBM Corporation 2012
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)