Versioned Infrastructure Templates

Authors:
Jennifer Beattie
Edmund Sutcliffe

Summary

This document discusses the motivation for using a version control system to manage all the files involved in an automatically provisioned infrastructure. It also provides a brief introductory recipe for configuring a new Subversion installation, with secure DAV access over HTTPS.

Finally, it builds on previous documents in this series to provide sample scripting methods to automatically rebuild a diskless NFS root image, in a side-by-side versioned environment.

Overview

Previous documents in this series have discussed how to build a diskless PXE environment, to enable the very rapid provisioning of new servers, with a guaranteed identical build due to automation.

When changes are needed, a diskless environment gives the advantage that the changes only have to be made in one place, and any server can automatically pick them up.

However, if you want to maintain multiple versions of an image, or roll back to previous versions of your environment, you need a mechanism to *track* changes, and a version control system is the natural choice to control this process.

Therefore:

- We assume use of Subversion as a simple and reliable centralized version control system.
- We assume that the files that make up the set of automated "build tools" are checked into Subversion on a specific managed server, so that the build machine only needs to have a Subversion client installed, in order to extract the desired version of the build scripts from the repository.
- We need a mechanism for these tools to build multiple versions of the environment side by side. We show a script fragment that simply advances a version number with each build run.
- We need a mechanism for each VM to choose a particular NFS root image to boot from. We assume that this will be mapped into the DHCP `root-path` settings.

We briefly describe each of these aspects in this document.

Prerequisites

You must have a build machine with a PXE boot environment installed as per an earlier document in this series, the Hive PXE Install document.

You must have a ESXi bootable target server to disklessly boot, and have DNS set up with records for the boot server and target server names.

Filesystem Paths and Network Addresses

Several of the pathnames used in this install process are configurable. This document assumes particular locations for these, and for ease of reading, the explanations below use these paths without qualifying them as potentially variable names.

The following paths are used as standard:

TFTP root directory	`/tftpboot`
NFS base directory	`/data/bootimage`
HTTP boot server filesystem location	`/data/bootserv`
CentOS HTTP repository root	`/data/bootserv/centos`
Subversion repository on client	`/data/buildtools`
Subversion repository on server	`/var/svn/BuildTools`

The network topology is assumed to be on a private network `10.1.4.0/24` in this example. The machines being created are intended to be part of a domain `test.example.com`.

We further assume that the build machine will be acting as the DNS nameserver for the local network, and this is reflected in the `resolv.conf` file we create. The build machine itself is named `core` in this example (ie `core.test.example.com`) and is assumed to be on `10.1.4.1`.

Installing Subversion on CentOS

In this section we briefly describe how to set up a Subversion repository on a server for our build scripts. It is crucial to remember that the build machine will primarily be a Subversion *client* in this scenario, requesting the latest available version of the build scripts.

The Subversion *server* containing the build scripts can be on any machine, and can indeed be a service in the cluster. Equally, one can start by creating a repository on the build machine itself.

When this document refers to `/var/svn/BuildTools`, and the Apache configuration steps, we are discussing the Subversion server, no matter where it is located.

When we refer to `/data/buildtools`, we are referring to the build machine, pulling the latest versions of the build template from the Subversion server in order to build new system images.

Subversion install and repository creation

Subversion v1.4.2 is provided with CentOS5, which is a sensible baseline (at the time of writing, the latest stable release is v1.6.11).

This makes installing it on the Subversion repository server, even fronted with Apache, rather easy:

```
yum -y install subversion httpd mod_dav_svn
```

The `svn` and `svnadmin` utilities will be installed into `/usr/bin` so should become immediately available on your PATH.

Next we create a directory to store the repository, and then create a new repository. Of course, it is critical to ensure that this directory is under a proper backup regime.

```
mkdir -p /var/svn
svnadmin create /var/svn/BuildTools
```

This should leave you with contents of the BuildTools directory looking like this:

```
[root@core build]# ls -alh /var/svn/BuildTools/
total 32K
drwxr-xr-x 6 root root 4.0K May 24 03:10 .
drwxr-xr-x 3 root root 4.0K May 24 03:10 ..
drwxr-xr-x 2 root root 4.0K May 24 03:10 conf
drwxr-sr-x 6 root root 4.0K May 24 03:10 db
-r--r--r-- 1 root root    2 May 24 03:10 format
drwxr-xr-x 2 root root 4.0K May 24 03:10 hooks
drwxr-xr-x 2 root root 4.0K May 24 03:10 locks
-rw-r--r-- 1 root root  229 May 24 03:10 README.txt
```

Directory permissions for Apache access

We assume that we'll create a role user called svn to own the Subversion tree, but the user that Apache runs as (by default, `www`) must also have write access to this tree to enable DAV access, so we make the Apache user a member of the svn group and set the sticky bit on the tree.

```
groupadd -r svn
useradd -g svn -r svn

chown -R svn.svn /var/svn
chmod -R g+rwxs /var/svn
chmod -R o-rwxs /var/svn

usermod -G svn www
```

It's also critical to ensure that Apache (running as `www`) creates new files in the repository with group access so that any other member of the `svn` group (eg the `svnserve` process) can still write to them.

Therefore we have to change the umask for Apache, and unfortunately the only sensible way to do that is to put a line at the top of the `/etc/init.d/httpd` script:

```
umask 002
```

Look at `http://svnbook.red-bean.com/en/1.0/svn-book.html#svn-ch-6-sect-5` for more on this topic.

Self-signed SSL certificate creation for secure HTTPS access

We ensure `openssl` is installed, then use it to generate a new private key, followed by a certificate signing request (representing the desired identity of the certificate holder).

We then sign the request ourselves to immediately produce a self-signed certificate. This has the advantage that it is free and immediate, although clients connecting will need to accept the resulting certificate. Alternatively, a certificate could be issued by an enterprise Certificate Authority or a public CA such as Verisign.

The commands below produce a private key that is not encrypted. Therefore it must be securely stored, or encrypted with a passphrase using one of the `-des*` or `-aes*` options to the openssl command.

```
cd /var/svn
openssl genrsa -out /var/svn/svn.key 1024
openssl req -new -key /var/svn/svn.key -out /var/svn/svn.csr
```

The certificate request command will prompt you to enter suitable fields for the certificate, for example GB or US for the country code. The Common Name you enter when asked, **must** match the hostname that you will eventually use to access the server.

You can then produce a self-signed certificate like this:

```
openssl x509 -req -in /var/svn/svn.csr -out /var/svn/svn.crt \
   -signkey /var/svn/svn.key -days 999
```

You can check a textual representation of the generated key, request and certificate files with the following commands:

```
openssl rsa -noout -text -in /var/svn/svn.key
openssl req -noout -text -in /var/svn/svn.csr
openssl x509 -noout -text -in /var/svn/svn.crt
```

The `svn.key` and `svn.crt` files can then be used in the Apache config to enable SSL access.

Apache configuration for Subversion access

The following aspects must be present in your Apache configuration.

The `mod_dav`, `mod_dav_fs`, `mod_dav_lock`, `mod_dav_svn` and `mod_authz_svn` modules must be loaded, plus of course the SSL module, so the following config lines are needed:

```
LoadModule ssl_module          modules/mod_ssl.so
LoadModule dav_module          modules/mod_dav.so
LoadModule dav_fs_module       modules/mod_dav_fs.so
LoadModule dav_lock_module     modules/mod_dav_lock.so
LoadModule dav_svn_module      modules/mod_dav_svn.so
LoadModule authz_svn_module    modules/mod_authz_svn.so
```

The basic SSL configuration directives must be present; in CentOS these can be configured in `/etc/httpd/conf.d/ssl.conf` .

Inside the SSL virtual host stanza, configure the certificate:

```
SSLCertificateFile      /var/svn/svn.crt
SSLCertificateKeyFile   /var/svn/svn.key
```

You can then configure a Subversion managed URL path, for example `/svn` , like this:

```
<Location /svn>
DAV                     svn
SVNPath                 "/var/svn/BuildTools"

AuthType                Basic
AuthName                "BuildTools"
AuthUserFile            /var/svn/svnusers.db
Require                 valid-user
</Location>
LimitXMLRequestBody 0
```

The `svnusers.db` file is then expected to be an Apache htpasswd format file which you can configure as you wish, or change the auth directives to integrate with other systems.

At this point you should be able to restart Apache, and visit your server (for example, `buildtools.test.example.com`), at a URL like
`https://buildtools.test.example.com/svn`
either in a browser, a Subversion GUI client, or from the command line.
In each case you will need to accept the self-signed certificate, but can then proceed to check in managed build files.

In the remainder of this document, we assume that you create a project named BuildTools inside this repository, so that the eventual code URL to check out from is
`https://buildtools.test.example.com/svn/BuildTools/`

We also assume that you create an automated build user named `build` to gain access to the repository, using the Apache `htpasswd` utility.

Automated Build Scripting

We assume that the build machine has the Subversion command line client installed via `yum install subversion`.

There are many methods and tools that could be used to run automated builds (and indeed we recommend using a continuous integration tool such as Hudson to control this process). In this document, therefore, we will merely show the core mechanics of this process.

Automated checkouts

We also assume that we will maintain an up to date copy of the build tools in `/data/buildtools/current`
from the repository in
`https://buildtools.test.example.com/svn/BuildTools`
using a Subversion automated userID called `build`.

This can be achieved by periodically running:

```
SVNUSER=build
SVNPASS=b0gus
if [ -d /data/buildtools/current ]; then
    svn update --non-interactive --trust-server-cert     \
               --username $SVNUSER --password $SVNPASS   \
               /data/buildtools/current
else
    svn co --non-interactive --trust-server-cert              \
           --username $SVNUSER --password $SVNPASS            \
           https://buildtools.test.example.com/svn/BuildTools \
           /data/buildtools/current
fi
```

which will ensure that we always run our builds from the latest official source.

Side by side build images

Whereas previously we built our NFS root image directly under `/data/bootimage/` , now we wish to have multiple root images side by side so that we can roll forwards and backwards in each environment.

To this end, we will adopt a new convention and move the images down one level, inserting a build number into the path. For example, build number 437 will be found under: `/data/bootimage/437/`

Our automated build script can generate a new build number by running:

```
MAXBUILD=`ls /data/bootimage | grep -E -e '[0-9]+' | sort -r | head -1`
BUILDNUM=`expr $MAXBUILD + 1`
BUILDROOT="/data/bootimage/$BUILDNUM"
mkdir -p $BUILDROOT
echo "Creating new build number $BUILDNUM in directory $BUILDROOT"
```

This simply finds the maximum existing numeric format directory name in `/data/bootimage` and adds one to it to form the new name.

At this point, our automatic build script can run through the steps that were outlined in our previous **Hive Diskless CentOS Install document**, to produce a fresh root image in an automated fashion.

Of course, it is also possible to use a variety of build systems to assist in achieving this end, such as Puppet (http://www.puppetlabs.com/) and Chef (http://www.opscode.com/chef/).

Depending on your environment, you might of course have a human-controlled procedure for naming particular builds to be promoted, but here we assume that the build machine will emit new build images directly into the `/data/bootimage` directory.

Versioned Image NFS Booting

Finally, we need a mechanism to ensure that different hosts in our environment can use different build images.

We assume that the provisioning process for new machines will control their identities and IP addresses via the PXE and DHCP servers for this environment, and in both cases we can control the NFS root path on a per-server basis, depending on which mechanism we use to pass the root path variable:

In `/tftpbootpxelinux.cfg/default` :

```
label centos-diskless
  kernel vmlinuz
  append initrd=initrd.img load_ramdisk=1 network root=/dev/nfs \
      NFSROOT=10.1.4.1:/data/bootimage/437
```

In a PXE server, we can create additional PXE menu files matching the MAC address or IP address of each server, and individually change their boot paths if needed.

In a DHCP environment, we can update the `root-path` option, and if needed we can divide the hosts into different groups depending on the build they require:

```
deny unknown-clients;
not authoritative;

   ... other default options here ...

group {
  next-server 10.1.4.1;          # Name of your TFTP server
  filename "pxelinux.0";         # Name of the bootloader program
  option root-path "10.1.4.1:/data/bootimage/437";

  host vapp1 {
        option host-name "vapp1.test.example.com";
        hardware ethernet 00:0c:29:01:02:03;
        fixed-address 10.1.4.101;
  }

  host vapp2 {
        option host-name "vapp2.test.example.com ";
        hardware ethernet 00:0c:29:04:05:06;
        fixed-address 10.1.4.102;
  }
}
```

In either case, we can partition the set of nodes in our environment to use the required build, and rollback of a given node is as simple as changing a single text line and rebooting the affected node.