TASTE TEST

Puppet - Chef - SaltStack - Ansible

THIRD EDITION

Bonus Chapters

Communities

Security

Docker

MATT JAYNES

The sample chapter is on the following pages. It shows the mini example project being done with simple shell commands. This same example is then used in the other chapters of the book to show each of the CM Tools in action: Puppet, Chef, SaltStack, and Ansible. It's a super simple example, but you'll probably be surprised how the different tools handle this trivial setup.

Shell Script

Wait, why am I starting with a shell script?!

Well, before we dive into the CM tools it's important to show you how the example system would be set up manually. Then you'll have a clear idea of what parts the system has and what the tools do when they perform the setup for us.

The obvious limitations

Just having a shell script for server setups is more than a lot of systems have. Still, it has some big limitations.

Can't run multiple times safely

A shell script works fine for the initial setup, but is pretty useless for ongoing maintenance and system changes. It generally can only be run once safely, then it's a liability and only good for documenting what happened once-upon-a-time on the system.

Using a shell script to set up a server generally indicates that it will then be managed manually afterwards (which leads to sadness and despair!).

A huge advantage of using a CM tool is that they're "idempotent". Idempotency basically means that you can run the directives over and over again safely.

An idempotent command will verify that the system is how you defined it and will only make changes to bring the system back into alignment with what you defined. That means you can define your system in the language of the CM tool and use it not only for initial system setup, but also for monitoring, updating, and correcting a server's configuration over the life of the server.

The CM tool can ultimately act like a self-healing test suite for your systems - neat!

Disparate command interfaces

Each system command (`ls`, `cat`, `tail`, etc) has a different group of developers behind it. That leads to slightly different interfaces for using each command. There are conventions that are generally followed, but the commands still differ from each other in subtle ways and sometimes differ depending on which Unix/Linux/BSD distribution they're on (ex: usage for the `ps` command on OS X is different from `ps` on Red Hat).

Another advantage of CM tools is that they abstract away some of these differences and give you a more unified interface for interacting with your system. For example, rather than looking up whether your system uses `groupadd` or the `addgroup` command, you can just use the standardized CM tool command to create the group. This command/directive happens to simply be called `group` in all of the four CM tools we're covering.

Similarly, CM tools standardize the reporting output from the different commands it runs.

Scenarios

I want to show you how the CM tools work for some typical scenarios. In order to do that quickly, I've created a fairly arbitrary system that isn't very realistic, but will give you a good sense of how each tool presents some key features.

The main scenarios I'll be covering are:

- **master/children node setup**
- **installing packages**

- **user/group setup**
- **deploying a static file**
- **deploying a templated file**
- **running a service**

In order to show those, I'll be setting up two simple websites.

Why two?

Well, there are several basic features I want to highlight which require more than one.

Frivolous story

Need a story for why this system exists?

Well, in 1965 a secret underground cult dedicated to puppy worship started an invisible war with a secret kitten cult. That war has raged for decades now and many strongly-worded memos have been exchanged between the cults.

In the efforts of peace and more civil memos, you've devised a way to appease the cults. You've observed both groups and they both desperately want a browser home page that presents a simple idyllic picture of their favorite baby animal.

So, you've searched the Creative Commons images for suitable puppy and kitten photos and come up with these:

Now, the cults are very sensitive about where the electrons serving the photo come from, so the puppy and kitten images must never be served from the same server, lest their electrons be contaminated by the other baby animal photo!

They also insist that the user/group that owns the puppy/kitty image be named 'puppy' or 'kitty' respectively.

Yes, it makes no sense - but what cult was ever very reasonable? ;-)

Launch servers

First we'll launch a puppy and a kitty server on Digital Ocean and use Ubuntu 14.04 x64 as the OS for both of them.

Note:
You don't have to use Digital Ocean, but each server is less than $0.01 per hour, so for each demo of a CM tool, you'll spend less than 5¢. Just remember to destroy your servers (or "droplets" as Digital Ocean calls them) when you're done with them so you don't get charged while they're idle.

I've personally tested the walkthrough for the shell script and each of the CM tools with this

setup, so for the smoothest experience, I recommend following the exact setup I used.

If you are already an experienced Vagrant user, then it should be pretty straight-forward to set this up for the walkthroughs. If you don't have experience with Vagrant yet, I recommend finishing this book first with the recommended Digital Ocean setup, then tackling Vagrant as a separate learning project. There's a learning curve and several very large downloads involved, so you don't want to get distracted with that right now.

Set their hostnames as `puppy.dev` and `kitty.dev`, then when the server is created and you get their IP addresses, add them to your `/etc/hosts` file like this (replacing the IPs below with the actual IPs Digital Ocean creates):

```
999.999.999.2  puppy.dev
999.999.999.3  kitty.dev
```

Let's build this thing!

First, let's build the puppy server, then when we've finished `puppy.dev` we'll build `kitty.dev` with the respective commands.

The steps will be pretty simple:

- **install nginx**
- **add the photo**
- **create user/group**
- **change photo's ownership/permissions**
- **add the html page**
- **run nginx**

Install nginx

Since we're doing this on the Ubuntu linux distribution we'll use `apt-get` to install nginx.

First we'll update the package lists so we get the most up-to-date packages:

```
root@puppy:~# apt-get update
```

Then we'll install the nginx package:

```
root@puppy:~# apt-get install nginx --assume-yes
```

Great, that was easy. We use the `--assumeyes` (same as `-y`) to avoid having to answer the "Are you sure?" type of prompts.

The web root for nginx is now at `/usr/share/nginx/html`, so that's where we'll be putting the puppy photo.

Add the photo

I've already created a Github repository with the resources for this project, including the images, so let's just download the image to the web server document root.

```
root@puppy:~# wget https://raw.github.com/nanobeep/tt/master/puppy.jpg \
>                 --output-document=/usr/share/nginx/html/puppy.jpg
```

Create user/group

Now we'll create the user/group:

```
root@puppy:~# useradd --user-group puppy
```

The `--user-group` flag tells `useradd` to also create a 'puppy' group and add the newly created puppy user to it.

Change photo's ownership and permissions

```
root@puppy:~# chown puppy:puppy /usr/share/nginx/html/puppy.jpg
root@puppy:~# chmod 664 /usr/share/nginx/html/puppy.jpg
```

Add the html page

The html page is super simple and looks like this:

```
<html>
    <body bgcolor="gray">
        <center>
            <img src="/baby.jpg">
        </center>
    </body>
</html>
```

Let's download it to the right place:

```
root@puppy:~# wget https://raw.github.com/nanobeep/tt/master/index.html \
>                  --output-document=/usr/share/nginx/html/index.html
```

To have the right image source, we'll just replace 'baby' with puppy:

```
root@puppy:~# sed --in-place 's/baby/puppy/g' /usr/share/nginx/html/index.html
```

You can probably guess that this is the page we'll be demonstrating CM tool templating with later :)

Run nginx

Now all we have to do is run the web server and we should be done.

Ubuntu keeps its service management scripts in `/etc/init.d/`, so let's look there:

```
root@puppy:~# ls -l /etc/init.d/ | grep nginx
-rwxr-xr-x 1 root root 2235 Jul 12  2012 /etc/init.d/nginx
```

Great, it's there and ready, so let's start it:

```
root@puppy:~# /etc/init.d/nginx start
Starting nginx: nginx.
```

Verify

Now, we can verify everything works by checking in the browser:

http://puppy.dev/

Putting it all together

So, our shell script for setting up a puppy server looks like this:

```
apt-get update
apt-get install nginx --assume-yes
wget https://raw.github.com/nanobeep/tt/master/puppy.jpg \
     --output-document=/usr/share/nginx/html/puppy.jpg
useradd --user-group puppy
chmod 664 /usr/share/nginx/html/puppy.jpg
chown puppy:puppy /usr/share/nginx/html/puppy.jpg
wget https://raw.github.com/nanobeep/tt/master/index.html \
     --output-document=/usr/share/nginx/html/index.html
sed --in-place 's/baby/puppy/' /usr/share/nginx/html/index.html
/etc/init.d/nginx start
```

Kitty

So, now to set up the kitty server, we'll just make a few substitutions:

```
apt-get update
apt-get install nginx --assume-yes
wget https://raw.github.com/nanobeep/tt/master/kitty.jpg \
     --output-document=/usr/share/nginx/html/kitty.jpg
useradd --user-group kitty
chmod 664 /usr/share/nginx/html/kitty.jpg
chown kitty:kitty /usr/share/nginx/html/kitty.jpg
wget https://raw.github.com/nanobeep/tt/master/index.html \
     --output-document=/usr/share/nginx/html/index.html
sed --in-place 's/baby/kitty/' /usr/share/nginx/html/index.html
/etc/init.d/nginx start
```

Run that on the kitty server, then verify that it worked:

http://kitty.dev/

Up next

Now you're familiar with the sample system we'll be using. The CM tool setups you're about to see will make a lot more sense now that you've seen exactly how the systems would be created by hand.