The Software Practitioner

Generated by Doxygen 1.8.8

Sun Jun 14 2015 23:14:41

Contents

Chapter 1

Main Page

List of pages

- Page Five Basic Tips for Teamwork with Git
- Page Using Cloud-Init Outside of Cloud
- Page Dynamic DNS with BIND and dhclient
- Page Java Logging Quick Reference

Chapter 2

Five Basic Tips for Teamwork with Git

Do you care about how your Git commits look like? A great software practitioner does, indeed. Let's review a couple of basic tips for developers that make the Git commit log look good and teamwork with Git source control more fun.

2.1 1) Introduce yourself to Git

When searching through the commit history your fellow developers would like to recognize that this particular awesome commit was created by you. Before your first commit, please, introduce yourself to Git. Tell Git your name and email address:

```
git config --global user.name "Ales Nosek"
git config --global user.email "anosek@verimatrix.com"
```

2.2 2) Commit message formatting

Respect the conventions for Git commit message formatting. The first line of the commit message is a short (50 chars or less) summary. The first letter of the summary is capitalized and there's no dot at the end of the summary. The summary begins with a reference to the issue in your bug tracking system if available. Use imperative in your summary line as you'd be commanding your code to do something, i.e. write "Remove obsolete code" instead of "Removed obsolete code" or "Removes obsolete code". More detailed explanatory text comes after a blank line. Here is a model Git commit message:

```
PROJXREF-27 Remove obsolete code in component X

The logging functionality in component X is not needed anymore.
Component Y took over the logging responsibility.
```

See also this `article` for more detailed explanation and examples.

2.3 3) One commit per unit of work

Create one commit per unit of work. A combined commit like "Add method X, correct indention, clean up whitespaces" is harder to review. Break your changes down into multiple commits, e.g. three commits "Add method X", "Correct indention" and "Clean up whitespaces".

2.4 4) git diff --cached

Before commiting *always* check your code changes. Make sure that your commit includes only the changes you intended to commit. Let your debug code and test modifications not flow into the production code base. Before committing double-check your changes with:

```
git diff --cached
```

2.5 5) Trailing whitespaces

Whitespace changes in the commit diffs decrease the readability of the commit diffs and make code review less fun. You can configure your editor to remove the trailing whitespaces for you on file save. Perhaps a better option though is to instruct Git to clean up the trailing white spaces automatically before comitting. You can use the commit hook `here` to do exactly that. Save the commit hook as file named `pre-commit`. Install the `pre-commit` script into your Git repository:

```
cp pre-commit my_existing_repo/.git/hooks
chmod 755 my_existing_repo/.git/hooks/pre-commit
```

The pre-commit hook will run only before the commit in this particular Git repository. If you'd like Git to install the pre-commit hook to every newly created/cloned repository you'll need to add the hook file into your template directory. First tell Git where is your template directory located by adding the following into your `~/.gitconfig` file:

```
[init]
        templatedir = ~/.gittemplate
```

Now you can create your Git template directory and copy the `pre-commit` hook file into it:

```
mkdir -p ~/.gittemplate/hooks
cp pre-commit ~/.gittemplate/hooks
chmod 755 ~/.gittemplate/hooks/pre-commit
```

From now on whenever you create/clone a Git repository you should find the `pre-commit` hook file installed at `.git/hooks/pre-commit`. Git will clean up trailing whitespaces for you before you commit.

Chapter 3

Using Cloud-Init Outside of Cloud

In EC2 and OpenStack cloud environments *user data* can be passed to the cloud instance to customize the cloud instance on the first boot. But what if your virtual machine doesn't run in the cloud environment? In this article we're going to configure our virtual machines with user data regardless if they're running in the cloud or not.

3.1 Introducing Cloud-Init

`Cloud-Init` is a tool that handles early initialization of a cloud instance. The `cloud-init` RPM package should be installed on the disk image which the cloud instance is going to boot up from. The package installs init scripts into `/etc/rc.d/init.d` that makes Cloud-Init run early during the system initialization. Cloud-Init obtains user data passed to it by the cloud software and executes them. User data contains a set of configuration tasks for the cloud instance. For example, Cloud-Init can update machine's hostname, configure `/etc/hosts`, create users, configure SSH authorized keys, resize filesystems, manage disk mounts, run user-defined scripts and `much more`.

Even if you're not running your virtual machines in the cloud environment it's worth it to deploy Clout-Init.

Every cloud software comes with its own mechanism of how to pass the user data to the cloud instance. For example, EC2 provides a *magic IP* from which the instance can download its user data. OpenStack cloud attaches a special *config drive* to the cloud instance containing the user data to be consumed by Clout-Init. In order to pass the user data to our virtual machine let's go the OpenStack way and assemble a minimum config drive.

3.2 Config drive assembly

First, we're going to prepare the following file structure for our config drive:

```
config_drive
config_drive/openstack
config_drive/openstack/latest
config_drive/openstack/2012-08-10
config_drive/openstack/2012-08-10/meta_data.json
config_drive/openstack/2012-08-10/user_data
```

Start by creating directories and the `latest` symbolic link like this:

```
mkdir config_drive
mkdir -p config_drive/openstack/2012-08-10
ln -s 2012-08-10 config_drive/openstack/latest
```

Next create a minimum metadata file required by Cloud-Init. I'm using a fully qualified domain name of the virtual machine as its UUID:

```
cat > config_drive/openstack/latest/meta_data.json << EOF
{
    "uuid": "myinstance.mydomain.com"
}
EOF
```

Cloud-Init supports many `formats` for scripts within user data. One of the most popular formats is the *cloud-config* file format. Let's create a cloud-config script that adds our SSH public key to the authorized keys for the user `root` on the virtual machine. We can then login into the virtual machine as user root without using a password. If you don't have a public-private SSH key pair you can quickly generate it using `ssh-keygen` command:

```
ssh-keygen -f mykey
```

Now create a `user_data` file with the configuration instructions for Cloud-Init. In the following code block replace the value of the `ssh-authorized-keys` field with the content of your generated `mykey.pub` file:

```
cat > config_drive/openstack/latest/user_data << EOF
#cloud-config
fqdn: myinstance.mydomain.com
users:
  - name: root
    ssh-authorized-keys:
      - ssh-rsa
       AAAAB3NzaC1yc2EAAAADAQABAAABAQDNH8Qwn4raGR1f9fvjbZe/GXM2N9Mh+eWlsFoYpcU4H5qf5YxT5CUo7BaTOgeE
    5geHyzxJQmCQlvoxcW3qkcjBJvVgEsTrrnX7KYS8BszvT4AMIuG2Za8f7myubXd6zYfj74XYhutUsPz7x2TEp9ZqbVkWcaElrQFxF2AzF7dV
    1RGntpPKyISqem70En8RYpGY514OLZ9TQDBYjbw8tfPuDd9mznXnWOZ34fPtP7+QDvOMFuA4tXsBpHj99/cbC0ViwzZtvb1QtY7dv9OFDgCRadw81+SKtzX
EOF
```

The file structure for our config drive is ready. Let's generate an ext2 filesystem and copy the files to it. The `virt-make-fs` utility from the `libguestfs-tools` package can help us with that:

```
virt-make-fs config_drive disk.config
```

In order for Cloud-Init to detect the attached drive as config drive the filesystem on the config drive needs to be labeled `config-2`. You can use `e2label` command from the `e2fsprogs` package to label your config drive:

```
e2label disk.config config-2
```

3.3 Cloud-Init in action

On my Linux host I'm running `libvirt` to ease the management of virtual machines. You can install it by running `sudo yum install libvirt`. There is a handy command-line utility `virsh` which comes with libvirt in the extra package `libvirt-client`.

Let's create a virtual machine with the config drive attached. As a virtual machine boot image I'm using a CentOS-6 image from `cloud.centos.org` which comes with Cloud-Init built in. Make sure that your virtual machine boot image has Cloud-Init installed. Following is a virtual machine definition file for the CentoOS-6 virtual machine. You might need to change the location of the disk image files and save it as `CentOS-6.xml`:

```
<domain type='kvm'>
  <name>CentOS-6</name>
  <memory unit='KiB'>2097152</memory>
  <os>
    <type>hvm</type>
  </os>
  <devices>
    <disk type='file' device='disk'>
      <driver name="qemu" type="qcow2"/>
       <source file='/tmp/CentOS-6-x86_64-GenericCloud.qcow2'/>
       <target bus="virtio" dev="vda"/>
     </disk>
     <disk type='file' device='disk'>
       <driver name="qemu" type="raw"/>
       <source file='/tmp/disk.config'/>
       <target bus="virtio" dev="vdb"/>
     </disk>
     <interface type='network'>
```

```
      <source network='default'/>
    </interface>
    <serial type="file">
      <source path="/tmp/CentOS-6.log"/>
    </serial>
  </devices>
</domain>
```

Okay, everything is ready, let's launch our Cloud-Init enabled CentOS-6 virtual machine:

```
sudo virsh define CentOS-6.xml
sudo virsh start CentOS-6
```

If everything went fine you can watch the console output of the booting virtual machine at `/tmp/CentOS-6.log`. Cloud-Init will print out the IP address obtained by the virtual machine (192.168.122.165 in my case) where we can login as root using the generated private key:

```
ssh -i testkey root@192.168.122.165
```

Congratulations, your virtual machine has just been configured by Cloud-Init the same way as any other virtual machine running in the cloud environment.

Chapter 4

Dynamic DNS with BIND and dhclient

In this blogpost we're going to configure the BIND server to accept dynamic updates. Client machines themselves will send the updates to the DNS server instead of letting DHCP server update the DNS. A great setup for situations where the DHCP server is not in your control.

Examples in this article work on RHEL6 that comes with BIND 9. You'll need to have `bind` and `bind-utils` RPM packages installed. In the following, the BIND server with host name `ns.somedomain.com` is an authoritative DNS server for the fictive zone `somedomain.com`.

4.1 Dynamic DNS with BIND

In our example we're going to configure the BIND server to accept DNS updates for `somedomain.com` zone from any client. In production environment you'd use encryption keys to secure the access to the DNS server. You can read more on the secure configuration in `this` excellent article. To allow any client to update the `somedomain.com` zone add the `allow-update { 0/0; };` option into your `/etc/named.conf` file:

```
zone "somedomain.com" in {
        type master;
        file "db.somedomain.com";
        allow-update { 0/0; };
};
```

After restarting the DNS server with `sudo /etc/init.d/named restart` we can test that the DNS updates are working. Let's ask the DNS server `ns.somedomain.com` to register a host `somehost.somedomain.com` with IP address `192.168.100.200`:

```
nsupdate -d << EOF
server ns.somedomain.com
update add somehost.somedomain.com 300 A 192.168.100.200
send
EOF
```

If everything worked fine you should see `status: NOERROR` in the reply from update query. The DNS server created a new record in its database pairing the `somehost.somedomain.com` host name with the IP address `192.168.100.200`. Let's check that the host name resolution works by issuing:

```
host somehost.somedomain.com ns.somedomain.com
```

You should see the IP address `192.168.100.200` in the command output. When on the DNS server you can dump the zone data into `/var/named/data/cache_dump.db` file for inspection:

```
rndc dumpdb -all
```

4.2 Updating DNS after IP acquisition

Our virtual machines obtain their IP addresses via DHCP. Whenever the virtual machine obtains a new IP address or renews the lease we'd like it to update the DNS accordingly. This way the DNS is always kept up to date and we're able to access the virtual machine using its host name.

The IP address acquisition is managed by the DHCP client `dhclient` running on the virtual machine. The `dhclient` can be extended by custom hooks. We are going to prepare a script that updates the DN↩ S database whenever the virtual machine acquires an IP address. Our DNS update hook must be saved at `/etc/dhcp/dhclient-eth0-up-hooks`. The `/sbin/dhclient-script` shell script that comes with the `dhclient` package will execute the hook. Upon execution the hook is passed a `reason` variable describing the event.

To install the update hook on the virtual machine let's make use of Cloud-Init tool that I talked about in the `previous blogpost`. The cloud-config script to be consumed by Cloud-Init looks as follows:

```
#cloud-config
fqdn: somehost.somedomain.com
write_files:
  - path: /etc/dhcp/dhclient-eth0-up-hooks
    permissions: '0755'
    content: |
      #!/bin/bash
      INTERFACE=eth0
      LEASE_FILE=/var/lib/dhclient/dhclient-$INTERFACE.leases
      HOST_ADDR=$(sed -n -e 's/.*fixed-address \([0-9]\+\.[0-9]\+\.[0-9]\+\.[0-9]\+\).*/\1/p' $LEASE_FILE | tail -1)
      HOST_NAME=$(hostname)
      NAMESERVER=ns.somedomain.com
      TTL=300

      if host $NAMESERVER 1>/dev/null 2>&1; then
        case $reason in
          BOUND|RENEW|REBIND|REBOOT)
            nsupdate << EOF
              server $NAMESERVER
              update delete $HOST_NAME A
              update add $HOST_NAME $TTL A $HOST_ADDR
              send
      EOF
          ;;
        esac
      fi
runcmd:
  - hostname somehost.somedomain.com # fix the hostname incorrectly set up by cloud-init
  - reason=BOUND /etc/dhcp/dhclient-eth0-up-hooks # DNS registration on first boot
```

Upon the very first execution of the hook the machine's network setup is not complete yet. There's no `/etc/resolv.conf` file written yet and the default route is not configured. The condition `if host $NA↩ MESERVER; then ...` skips the DNS update in this case. Later in the initialization process the `runcmd` part of the cloud-config script gets executed. At this time the network configuration is complete and so we execute the update hook manually. This is the first time that the virtual machine registers itself with DNS. Cloud-Init executes the `runcmd` section only on the very first boot. Subsequent boots won't execute the `runcmd` code.

Note that we're parsing the `/var/lib/dhclient/dhclient-eth0.leases` file to obtain the acquired IP address. Should the virtual machine obtain different IP address in the future the DNS entry gets updated accordingly.

Chapter 5

Java Logging Quick Reference

The Simple Logging Facade for Java (SLF4J) serves as a simple abstraction for various logging frameworks. Let's look at how to configure SLF4J to work with SLF4J Simple logger, JDK 1.4 logger, Log4j, Logback and Log4j2 framework.

Following is a Java code of our logging application `LogApp`. Note that it uses SLF4J API classes to do the logging. The jar file `slf4j-api-1.7.12.jar` is the only compile time dependency.

```java
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class LogApp {

    private static final Logger log = LoggerFactory.getLogger(LogApp.class);

    public static void main(String[] args) {
        log.trace("Trace message");
        log.debug("Debug message");
        log.info("Info message");
        log.warn("Warning message");
        log.error("Error message");
    }
}
```

You can compile the LogApp code with:

```
javac -cp slf4j-api-1.7.12.jar LogApp.java
```

And run the LogApp with:

```
java -cp .:slf4j-api-1.7.12.jar LogApp
```

The output shows that SLF4J is missing a logger implementation. In the following we'll demonstrate how to plug in different logging backends. The principle is always the same: include the logging framework implementation jars on the classpath, include the respective SLF4J adaptor jar on the classpath and provide a configuration file specific to the logging framework.

```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
```

5.1 SLF4J Simple logger

The SLF4J comes with a Simple logger implemenation. Simple logger provides only basic functionality. It read its configuration from a `simplelogger.properties` file that must be included on the classpath. There's no way to specify a different location of the configuration file.

```
org.slf4j.simpleLogger.logFile=System.out
org.slf4j.simpleLogger.defaultLogLevel=debug
org.slf4j.simpleLogger.showDateTime=true
org.slf4j.simpleLogger.dateTimeFormat=HH:mm:ss.SSS
```

```
org.slf4j.simpleLogger.logFile=/tmp/logger.out
org.slf4j.simpleLogger.defaultLogLevel=debug
org.slf4j.simpleLogger.showDateTime=true
org.slf4j.simpleLogger.dateTimeFormat=HH:mm:ss.SSS
```

Run the application with:

```
java -cp .:slf4j-api-1.7.12.jar:slf4j-simple-1.7.12.jar LogApp
```

5.2 JDK 1.4 logger (java.util.logging)

The java.util.logging package was introduced in JDK 1.4. The default `logging.properties` configuration file that comes with JRE (`jre/lib/logging.properties`) specifies a ConsoleHandler that routes logging to System.err. There's no way how to configure the JDK 1.4 logger to log to standard output instead of standard error output unless you do the configuration programmaticaly. You can specify the location of your JDK 1.4 logging configuration file in `java.util.logging.config.file` Java property.

```
handlers=java.util.logging.ConsoleHandler
.level=FINE
java.util.logging.ConsoleHandler.level=FINE
java.util.logging.ConsoleHandler.formatter=java.util.logging.SimpleFormatter
# message pattern works since Java 7
java.util.logging.SimpleFormatter.format=%1$tT [%2$s] %4$s - %5$s %6$s%n
```

```
handlers=java.util.logging.FileHandler
.level=FINE
java.util.logging.FileHandler.level=FINE
java.util.logging.FileHandler.pattern=/tmp/logger.out
java.util.logging.FileHandler.append=true
java.util.logging.FileHandler.formatter=java.util.logging.SimpleFormatter
# message pattern works since Java 7
java.util.logging.SimpleFormatter.format=%1$tT [%2$s] %4$s - %5$s %6$s%n
```

Run the application with:

```
java -cp .:slf4j-api-1.7.12.jar:slf4j-jdk14-1.7.12.jar
     -Djava.util.logging.config.file=/tmp/jdk14.stderr.properties LogApp
```

5.3 Log4j

The Log4j doesn't log a single message for you unless you provide it with a proper configuration:

```
log4j:WARN No appenders could be found for logger (LogApp).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
```

The following are the sample Log4j configuration files. Note that the `log4j.configuration` Java property that specifies the location of the configuration file must be a URL. In the example below the `/tmp/log4j.stdout.↩ properties` location has to be prepended with `file:` to form a URL.

```
log4j.rootLogger=DEBUG, stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{HH:mm:ss} [%t] %-5p %c{1}:%L - %m%n
```

```
log4j.rootLogger=DEBUG, file
log4j.appender.file=org.apache.log4j.FileAppender
log4j.appender.file.File=/tmp/logger.out
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{HH:mm:ss} [%t] %-5p %c{1}:%L - %m%n
```

Run the application with:

```
java -cp .:slf4j-api-1.7.12.jar:slf4j-log4j12-1.7.12.jar:log4j-1.2.17.jar
     -Dlog4j.configuration=file:/tmp/log4j.stdout.properties LogApp
```

5.4 Logback

With no configuration provided Logback defaults to printing all log messages on the console standard output. The `logback-classic` jar package that comes with Logback includes the `org.slf4j.impl.Static↩ LoggerBinder` class that serves as an adaptor to SLF4J framework. Therefore no extra SLF4J adaptor jar is needed on the runtime classpath. You can specify the location of your Logback configuration file in the `logback.↩ configurationFile` Java property.

```
<configuration>
  <appender name="stdout" class="ch.qos.logback.core.ConsoleAppender">
    <Target>System.out</Target>
    <encoder>
      <pattern>%d{HH:mm:ss} [%t] %-5p %c{1}:%L - %m%n</pattern>
    </encoder>
  </appender>
  <root level="DEBUG">
    <appender-ref ref="stdout"/>
  </root>
</configuration>
```

```
<configuration>
  <appender name="file" class="ch.qos.logback.core.FileAppender">
    <File>/tmp/logger.out</File>
    <encoder>
      <pattern>%d{HH:mm:ss} [%t] %-5p %c{1}:%L - %m%n</pattern>
    </encoder>
  </appender>
  <root level="DEBUG">
    <appender-ref ref="file"/>
  </root>
</configuration>
```

Run the application with:

```
java -cp .:slf4j-api-1.7.12.jar:logback-core-1.1.3.jar:logback-classic-1.1.3.jar
     -Dlogback.configurationFile=/tmp/logback.stdout.xml LogApp
```

5.5 Log4j2

With no configuration provided Log4j2 informs you that it logs only error messages to stdout. You can provide the location of your Log4j2 configuration file in the `log4j.configurationFile` Java property.

```
ERROR StatusLogger No log4j2 configuration file found. Using default configuration: logging only errors to
     the console.
22:09:13.052 [main] ERROR LogApp - Error message
```

```
<Configuration>
    <Appenders>
        <Console name="console" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss} [%t] %-5p %c{1}:%L - %m%n" />
        </Console>
    </Appenders>
    <Loggers>
        <Root level="debug">
            <AppenderRef ref="console" />
        </Root>
    </Loggers>
</Configuration>
```

```
<Configuration>
    <Appenders>
        <File name="file" fileName="/tmp/logger.out" append="true">
            <PatternLayout pattern="%d{HH:mm:ss} [%t] %-5p %c{1}:%L - %m%n"/>
        </File>
    </Appenders>
    <Loggers>
        <Root level="debug">
            <AppenderRef ref="file" />
        </Root>
    </Loggers>
</Configuration>
```

Run the application with:

```
java -cp .:slf4j-api-1.7.12.jar:log4j-core-2.2.jar:log4j-api-2.2.jar:log4j-slf4j-impl-2.2.jar
      -Dlog4j.configurationFile=/tmp/log4j2.stdout.xml LogApp
```