Parallel and Distributed Systems

Secure and Scalable deployments

Author: Mihai CARABAȘ

Editura POLITEHNICA PRESS

Reviewers:

Darius MIHAI

Elena MIHĂILESCU

Mihai BĂRBULESCU

ISBN: 978-606-515- 956-3

Contents

Security: PKI, X.509, SSL, TLS 4
Experimental Setup 4
Inspecting and Verifying a Certificate 4
Remotely Inspecting a Certificate 6
Generating and Inspecting a Certificate 9
Unencrypted Client/Server Communication 11
Client/Server Communication over SSL/TLS 11
Configuration Management: Puppet, Ansible 13
Experimental Setup 13
Puppet Resources 13
Puppet Manifests 17
Resource Dependency 19
Design Patterns: Package / File / Service 22
Variables and Conditional Statements 23
Ansible Install & Configuration 26
Ansible Facter 28
Two-factor Authentication for SSH 29

Security: PKI, X.509, SSL, TLS

TLS (Transport Layer Security) is a cryptographic protocol that provides communication security between a client and a server. Usually, the identity of the server is verified through a certificate. This certificate contains a public key, the identity of the server and a signature which verifies that the key belongs to the entity in the certificate.

A certificate is valid if it is signed by a Certificate Authority (CA). The CA is considered trustworthy by the communication client. The client has access to the certificate of the CA, with which the signature in the certificate belonging to the server can be verified and, consequently, the identity of the server can be verified.

Experimental Setup

- We will be using a virtual machine in the faculty's cloud.
- When creating a virtual machine follow the steps in this tutorial.
- When creating a virtual machine in the Launch Instance window:
  - Select **Boot from image** in **Instance Boot Source** section
  - Select **SCGC Template** in **Image Name** section
  - Select a flavor that is at least **m1.medium**.
- The username for connecting to the VM is `student`
- For the following exercises, the resources can be found in the laboratory archive:

```
[student@scgc ~] $ cd
[student@scgc ~] $ wget --user=<username> --ask-password https://repository.grid.pub.ro/cs/scgc/laboratoare/lab-08.zip
```

```
[student@scgc ~] $ unzip lab-08.zip
```

Inspecting and Verifying a Certificate

Begin by inspecting the certificate found in the `houdini.cs.pub.ro.crt-roedunet` file.

```
$ openssl x509 -in houdini.cs.pub.ro.crt-roedunet -noout -text
```

In the output you can find information about:

- the issuer
- the validity
  - start date
  - end date

- the public key
    - algorithm
    - modulus
    - exponent
- certificate extensions
- signature

Specific information regarding the certificate can be printed by replacing the `-text` argument with the one or more of the following:

```
$ openssl x509 -in houdini.cs.pub.ro.crt-roedunet -noout -pubkey

$ openssl x509 -in houdini.cs.pub.ro.crt-roedunet -noout -startdate

$ openssl x509 -in houdini.cs.pub.ro.crt-roedunet -noout -enddate

$ openssl x509 -in houdini.cs.pub.ro.crt-roedunet -noout -dates

$ openssl x509 -in houdini.cs.pub.ro.crt-roedunet -noout -issuer

$ openssl x509 -in houdini.cs.pub.ro.crt-roedunet -noout -subject

$ openssl x509 -in houdini.cs.pub.ro.crt-roedunet -noout -modulus
```

To verify the certificate using a certificate chain, use the following command:

```
$ openssl verify -CAfile terena-ca-chain.pem houdini.cs.pub.ro.crt-roedunet

houdini.cs.pub.ro.crt-roedunet: OU = Domain Control Validated, CN = houdini.cs.pub.ro

error 10 at 0 depth lookup:certificate has expired

OK
```

The certificate is expired, but has otherwise been verified.

Check the information in certificate chain:

```
$ cat terena-ca-chain.pem

-----BEGIN CERTIFICATE-----

...

-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----

...
```

```
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----

...

-----END CERTIFICATE-----
```

Notice there are multiple certificates in the file. Although `openssl` does not provide direct support for printing information about each certificate in the chain, the following workaround can be used:

```
$ openssl crl2pkcs7 -nocrl -certfile terena-ca-chain.pem | openssl pkcs7 -print_certs -noout

subject=/C=NL/ST=Noord-Holland/L=Amsterdam/O=TERENA/CN=TERENA SSL CA 2

issuer=/C=US/ST=New Jersey/L=Jersey City/O=The USERTRUST Network/CN=USERTrust RSA Certification Authority



subject=/C=US/ST=New Jersey/L=Jersey City/O=The USERTRUST Network/CN=USERTrust RSA Certification Authority

issuer=/C=SE/O=AddTrust AB/OU=AddTrust External TTP Network/CN=AddTrust External CA Root



subject=/C=SE/O=AddTrust AB/OU=AddTrust External TTP Network/CN=AddTrust External CA Root

issuer=/C=SE/O=AddTrust AB/OU=AddTrust External TTP Network/CN=AddTrust External CA Root
```

Verify `open-source.cs.pub.ro.crt-roedunet` and `security.cs.pub.ro.crt-roedunet` using the two certificate chains present in the resources archive.
Find the `issuer` for each of the certificates and use the appropriate certificate chain.

Remotely Inspecting a Certificate

Connect to `aero.curs.pub.ro` using a secure connection to obtain its certificate.

```
$ echo | openssl s_client -connect aero.curs.pub.ro:443

CONNECTED(00000005)

depth=2 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert High Assurance EV Root CA

verify return:1
```

```
depth=1 C = NL, ST = Noord-Holland, L = Amsterdam, O = TERENA, CN = TERENA SSL
High Assurance CA 3

verify return:1

depth=0 businessCategory = Government Entity, jurisdictionC = RO, serialNumber
= Government Entity, C = RO, L = Bucure\C8\99ti, O = Universitatea POLITEHNICA
din Bucuresti, OU = NCIT Cluster, CN = acs.curs.pub.ro

verify return:1

---

Certificate chain

 0 s:businessCategory = Government Entity, jurisdictionC = RO, serialNumber =
Government Entity, C = RO, L = Bucure\C8\99ti, O = Universitatea POLITEHNICA
din Bucuresti, OU = NCIT Cluster, CN = acs.curs.pub.ro

   i:C = NL, ST = Noord-Holland, L = Amsterdam, O = TERENA, CN = TERENA SSL
High Assurance CA 3

 1 s:C = NL, ST = Noord-Holland, L = Amsterdam, O = TERENA, CN = TERENA SSL
High Assurance CA 3

   i:C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert High
Assurance EV Root CA

...
```

The received certificate appears to be for `acs.curs.pub.ro`. This is because both servers have same certificate (issued to acs.curs.pub.ro) and `aero.cs.pub.ro` is a subject alternative name (SAN) for the domain. Let's inspect the certificate:

```
$ echo | openssl s_client -connect aero.curs.pub.ro:443 2>/dev/null | sed -ne
'/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' | openssl x509 -noout -text

Certificate:

    Data:

        Version: 3 (0x2)

        Serial Number:

            0d:34:0a:2f:41:fa:35:0e:5b:29:85:4c:1e:c1:51:23

        Signature Algorithm: sha256WithRSAEncryption

        Issuer: C = NL, ST = Noord-Holland, L = Amsterdam, O = TERENA, CN =
TERENA SSL High Assurance CA 3

        Validity

            Not Before: Sep 17 00:00:00 2019 GMT
```

```
            Not After : Sep 21 12:00:00 2020 GMT

        Subject: businessCategory = Government Entity, jurisdictionC = RO,
serialNumber = Government Entity, C = RO, L = Bucure\C8\99ti, O = Universitatea
POLITEHNICA din Bucuresti, OU = NCIT Cluster, CN = acs.curs.pub.ro

        Subject Public Key Info:

            Public Key Algorithm: rsaEncryption

                RSA Public-Key: (2048 bit)

                Modulus:

                    00:bd:8f:eb:51:6d:52:af:25:30:c7:d2:92:34:a7:

                    7e:8f:b5:44:9c:4f:2c:0c:71:33:72:83:e9:53:cc:

                    7e:e3:9b:e2:81:95:48:a2:bd:9e:0c:de:d1:e0:56:

                    9f:f5:54:ea:70:9e:be:32:13:8e:6f:59:0b:57:45:

                    c5:ca:f8:4b:5a:66:da:89:48:f2:fb:32:2c:0d:75:

                    76:e1:e7:8b:57:2b:01:61:1c:a8:71:42:a5:6b:35:

                    7f:3e:a5:5b:dd:8d:85:8a:bf:ba:f2:0a:db:ed:eb:

                    c8:2a:9c:af:4b:2b:c2:28:80:3b:38:47:f3:64:80:

                    7f:7d:75:8c:9c:34:d2:63:ef:cd:d9:37:88:57:e0:

                    49:54:df:fc:11:e1:e7:80:3b:74:95:f2:71:05:0d:

                    13:6a:fa:ba:eb:43:62:f9:dd:80:b7:f1:ee:36:5d:

                    8e:9e:f6:7e:5a:cb:da:a0:ad:2b:17:ce:36:70:a1:

                    24:92:e3:60:f4:c5:a4:8d:da:53:e7:42:0a:e0:9d:

                    4b:64:8e:86:37:31:fe:53:b8:23:4b:71:75:48:c6:

                    af:97:fe:e5:26:05:54:5c:6b:b6:40:f2:98:8c:13:

                    05:b4:43:b7:aa:c6:76:06:85:fb:71:73:29:37:2d:

                    00:12:b3:63:5d:13:f1:4a:06:06:c0:6b:e6:d1:01:

                    8d:f5

                Exponent: 65537 (0x10001)

        X509v3 extensions:

            X509v3 Authority Key Identifier:
```

```
keyid:C2:B8:85:D7:E1:B9:13:BD:D1:48:BC:FD:5E:DC:7D:90:42:7A:8A:A9



            X509v3 Subject Key Identifier:

                84:AD:71:69:54:FA:D1:44:BC:74:1A:9F:C8:93:25:D7:A3:62:80:9D

            X509v3 Subject Alternative Name:

                DNS:acs.curs.pub.ro,                      DNS:aero.curs.pub.ro,
DNS:aracis.curs.pub.ro,        DNS:chim.curs.pub.ro,        DNS:cs.curs.pub.ro,
DNS:dmkm.curs.pub.ro,      DNS:dppd.curs.pub.ro,      DNS:electro.curs.pub.ro,
DNS:electronica.curs.pub.ro,   DNS:energ.curs.pub.ro,   DNS:faima.curs.pub.ro,
DNS:fils.curs.pub.ro,         DNS:fim.curs.pub.ro,         DNS:fsa.curs.pub.ro,
DNS:hub.curs.pub.ro,         DNS:imst.curs.pub.ro,         DNS:isb.curs.pub.ro,
DNS:mecanica.curs.pub.ro,   DNS:nt.curs.pub.ro,   DNS:posdru62485.curs.pub.ro,
DNS:postdoc.curs.pub.ro,       DNS:sas.curs.pub.ro,       DNS:sim.curs.pub.ro,
DNS:tet.curs.pub.ro,     DNS:transporturi.curs.pub.ro,     DNS:www.curs.pub.ro,
DNS:fiir.curs.pub.ro

...
```

As we can see, all the Subject Alternative Names (SAN) can be found under in the certificate, under `DNS` entries.
Within a browser, inspect the certificate for `aero.curs.pub.ro` and find the field that specifies the Subject Alternative Names for the certificate.

Generating and Inspecting a Certificate

The steps required when generating a certificate are as follows:

- generate a private key
- generate a certificate signing request (CSR) with the key and identification data
- send the CSR to a CA in order to have it signed

We will generate a CSR for `server.scgc`. First, generate a private key:

```
$ openssl genrsa -out server.scgc.key 2048

Generating RSA private key, 2048 bit long modulus

.............................................+++

.....................................................................
........+++

e is 65537 (0x10001)
```

Then, generate the signing request:

```
$ openssl req -new -key server.scgc.key -out server.scgc.csr

...
```

Supply the following information in the request:

- `Organization Name: SCGC`
- `Organizational Unit: Development`
- `Common Name: server.scgc`

The other fields can be completed as desired.

Usually, at this point, the request would be sent to a trusted CA in order to be signed. Instead, we will sign the certificate using the `scgc-ca.crt` certificate from the resource archive.

```
$ openssl ca -config scgc-ca.cnf -policy signing_policy -extensions signing_req
-in server.scgc.csr -out server.scgc.crt

Using configuration from scgc-ca.cnf

Check that the request matches the signature

Signature ok

...

Sign the certificate? [y/n]:y




1 out of 1 certificate requests certified, commit? [y/n]y

Write out database with 1 new entries

Data Base Updated
```

Inspect the `scgc-ca.cnf` file, in particular the `signing_policy` section.
A more complex openssl configuration file can be found at `/etc/ssl/openssl.cnf`.

Verify that the signed certificate matches the generated key.

```
$ openssl x509 -in server.scgc.crt -noout -modulus | md5sum

d80db122c02c6ef6eabb3b4cbd8b8f40  -

$ openssl rsa -in server.scgc.key -noout -modulus | md5sum

d80db122c02c6ef6eabb3b4cbd8b8f40  -
```

Furthermore, verify the certificate using the `scgc-ca.crt` certificate.

```
$ openssl verify -CAfile scgc-ca/scgc-ca.crt server.scgc.crt
```

```
server.scgc.crt: OK
```

Currently, the `scgc-ca.crt` certificate is expired, so the last command will fail. If you want to solve this issue, you can regenerate the CA certificate by running the following commands (and resign the newly created CSR):

```
$ openssl req -new -key scgc-ca/scgc-ca.key -out scgc-ca/scgc-ca.csr
$ openssl x509 -req -in scgc-ca/scgc-ca.csr -signkey scgc-ca/scgc-ca.key -out
scgc-ca/scgc-ca.crt
```

Unencrypted Client/Server Communication

First, in a separate terminal, start a `tcpdump` session to dump traffic on the loopback interface. We will also use this for the next exercise.

```
$ sudo tcpdump -A -i lo port 12345
```

Now, start a simple server listening on the port `tcpdump` is monitoring.

```
$ nc -l 12345
```

To connect to the server, run the following in another terminal.

```
$ nc localhost 12345
```

Notice that any text typed into the client shows in the server and vice-versa. Also, the messages can be seen in plaintext in the `tcpdump` log.

Client/Server Communication over SSL/TLS

Use `openssl s_server` to start a server listening on the same port as the previous exercise. Use the `server.scgc` certificate previously generated.

```
$ openssl s_server -key server.scgc.key -cert server.scgc.crt -accept 12345

Using default temp DH parameters

ACCEPT
```

Connect to the server using `openssl s_client`.

```
$ openssl s_client -connect localhost:12345

CONNECTED(00000003)

depth=0 C = RO, ST = Bucharest, L = Bucharest, O = SCGC, OU = Development, CN
= server.scgc

verify error:num=20:unable to get local issuer certificate

verify return:1
```

```
depth=0 C = RO, ST = Bucharest, L = Bucharest, O = SCGC, OU = Development, CN
= server.scgc

verify error:num=21:unable to verify the first certificate

verify return:1

---

Certificate chain

 0 s:/C=RO/ST=Bucharest/L=Bucharest/O=SCGC/OU=Development/CN=server.scgc

   i:/C=RO/O=SCGC/OU=Development/CN=SCGC CA

---

...

    Verify return code: 21 (unable to verify the first certificate)
```

The validation of the server certificate has failed.

Attempt the connection again, this time specifying the CA certificate.

```
$ openssl s_client -CAfile scgc-ca/scgc-ca.crt -connect localhost:12345

CONNECTED(00000003)

depth=1 C = RO, O = SCGC, OU = Development, CN = SCGC CA

verify return:1

depth=0 C = RO, ST = Bucharest, L = Bucharest, O = SCGC, OU = Development, CN
= server.scgc

verify return:1

---

Certificate chain

 0 s:/C=RO/ST=Bucharest/L=Bucharest/O=SCGC/OU=Development/CN=server.scgc

   i:/C=RO/O=SCGC/OU=Development/CN=SCGC CA

---

...

    Verify return code: 0 (ok)
```

The connection has been successfully established. Verify that messages exchanged between server and client are no longer displayed in the `tcpdump` log.

Configuration Management: Puppet, Ansible

Experimental Setup

- We will be using a virtual machine in the faculty's cloud.
- When creating a virtual machine follow the steps in this tutorial.
- When creating a virtual machine in the Launch Instance window:
  - Select **Boot from image** in **Instance Boot Source** section
  - Select **SCGC Template** in **Image Name** section
  - Select a flavor that is at least **m1.medium**.
- The username for connecting to the VM is `student`
- First, download the laboratory archive:

```
[student@scgc ~] $ cd scgc
[student@scgc ~/scgc] $ wget --user=<username> --ask-password
https://repository.grid.pub.ro/cs/scgc/laboratoare/lab-07.zip
```

```
[student@scgc ~/scgc] $ unzip lab-07.zip
```

After unzipping you should have a KVM image file (`puppet.qcow2`) and a script used to start the VM (`lab07-start-kvm`).
To start the VM, run the startup script:

```
student@scgc:~/scgc$ ./lab07-start-kvm
```

The KVM virtual machine for the lab will boot (can take up to 2-3 minutes).

In order to access the VM, use the following IP address (the password is `student`):

```
student@scgc:~/scgc$ ssh student@10.0.0.2
```

Puppet Resources

Puppet is a configuration management tool. In order to describe the necessary configurations, Puppet uses its own declarative language. Puppet can manage both Linux and Windows systems.

Puppet resources

Puppet uses a **resource** as an abstraction for most entities and operations to be performed on a system. As an example, the state of a service (running/stopped) is defined in Puppet as a resource.

Use the **puppet resource service** command to see the system services from Puppet's perspective.

```
[root@puppet ~]# puppet resource service

service { 'apparmor':

  ensure => 'running',

  enable => 'true',

}

service { 'apparmor.service':

  ensure => 'running',

  enable => 'true',

}

service { 'autovt@':

  ensure => 'stopped',

  enable => 'true',

}

service { 'autovt@.service':

  ensure => 'stopped',

  enable => 'true',

}

service { 'bootlogd.service':

  ensure => 'stopped',

  enable => 'false',

}

service { 'bootlogs.service':

  ensure => 'stopped',

  enable => 'false',

}

...
```

The previous command syntax is:

- **puppet** command - used for accessing most of Puppet's features
- **resource** subcommand - interacts with available Puppet resources
- **service** parameter - type of the resources to be shown

Besides services, other Puppet resource examples are:

- users
- files or directories
- packages (software)

Resource structure

Show the resource representing the **root** user account, using the command: **puppet resource user root**

```
[root@puppet ~]# puppet resource user root

user { 'root':

  ensure             => 'present',

  comment            => 'root',

  gid                => 0,

  home               => '/root',

  password           => '$6$SWpfJK2ozbQ.bFA9$/[...]',

  password_max_age   => 99999,

  password_min_age   => 0,

  password_warn_days => 7,

  shell              => '/bin/bash',

  uid                => 0,

}
```

The resource structure contains the following elements:

- **Type** of resource: for this example, **user**
- **Name** of the resource: **'root'**
- **Attributes** of the resource: **ensure, comment, gid, home** etc.
- Each attribute has a certain **value**

The previous syntax forms the "resource declaration".

Types of resources

Besides services and users, Puppet implements a lot of other types of resources. In order to show them, use the command **puppet describe –list**

```
[root@puppet ~]# puppet describe --list

These are the types known to puppet:

augeas          - Apply a change or an array of changes to the  ...

computer        - Computer object management using DirectorySer ...

cron            - Installs and manages cron jobs

exec            - Executes external commands

file            - Manages files, including their content, owner ...

filebucket      - A repository for storing and retrieving file  ...

group           - Manage groups

...
```

Creating / removing a resource

Using the **puppet resource** command, we can create new resources. Generic syntax is: puppet resource type name attr1=val1 attr2=val2

If we want to create the user **gigel** so that:

- the user home directory is **/home/gigel**
- the default shell is **/bin/sh**

The Puppet command for this is:

```
[root@puppet ~]# puppet resource user gigel ensure=present shell="/bin/sh"
home="/home/gigel"

Notice: /User[gigel]/ensure: created

user { 'gigel':

  ensure => 'present',

  home   => '/home/gigel',

  shell  => '/bin/sh',

}
```

Open the **/etc/passwd** file and check if the user has been created.

In order to remove a resource, the **ensure** attribute must be set to **absent**.
As an example, to remove the user **gigel** that we have previously created:

```
[root@puppet ~]# puppet resource user gigel ensure=absent

Notice: /User[gigel]/ensure: removed

user { 'gigel':

  ensure => 'absent',

}
```

Chech the **/etc/passwd** file to see if the user was actually removed.

Puppet Manifests

Even though we can create, modify or remove resources from the command line, using **puppet resource** commands, this is not a scalable approach and not appropriate for complex scenarios.
A better solution would be:

- declaring resources in a (text) file
- applying the described modifications using Puppet

Files containing Puppet resource declarations are called **manifests** and usually have the **.pp** file extension.

Creating a manifest

We are going to write a manifest that describes a (text) file resource. The file is going to have the following properties:

- name and path: **/tmp/my_file**
- access rights: **0604**
- file content: "File created using Puppet"

Resource declaration has the following syntax:

```
file {'my_file':

  path    => '/tmp/my_file',

  ensure  => present,

  mode    => '0640',

  content => "File created using Puppet.",

}
```

Save the previously described code in a manifest file called **my_file_manif.pp**

Applying a manifest

Applying a manifest is done with the command: **puppet apply**

```
[root@puppet ~]# puppet apply my_file_manif.pp


Notice: Compiled catalog for puppet in environment production in 0.18 seconds

Notice:    /Stage[main]/Main/File[my_file]/ensure:    defined    content    as
'{md5}b4fdf30d694de5a5d7fe7a50cda27851'

Notice: Finished catalog run in 0.38 seconds
```

Check that the file has been created and the content and access rights are correct.

Try to apply the same manifest one more time:

```
[root@puppet ~]# puppet apply my_file_manif.pp


Notice: Compiled catalog for puppet in environment production in 0.16 seconds

Notice: Finished catalog run in 0.38 seconds
```

Notice that if the resource is already in the state described by the manifest, Puppet does not execute any action.

Change the access rights of the file to **755** and then apply the manifest again.

```
[root@puppet ~]# chmod 755 /tmp/my_file

[root@puppet ~]# puppet apply my_file_manif.pp


Notice: Compiled catalog for puppet in environment production in 0.18 seconds

Notice: /Stage[main]/Main/File[my_file]/mode: mode changed '0755' to '0640'

Notice: Finished catalog run in 0.38 seconds
```

Change the content of the file and then apply the manifest again.

```
[root@puppet ~]# echo "This is not my file" > /tmp/my_file

[root@puppet ~]# puppet apply my_file_manif.pp


Notice: Compiled catalog for puppet in environment production in 0.18 seconds
```

```
Notice:      /Stage[main]/Main/File[my_file]/content:      content      changed
'{md5}7225302b0d15d4a2562c2ab55e45d4cc'                                   to
'{md5}b4fdf30d694de5a5d7fe7a50cda27851'

Notice: Finished catalog run in 0.41 seconds
```

Notice that if the attributes of the resource are different from the ones described in the manifest, applying the manifest brings the resource back to the desired state.

States (ensure)

The **ensure** attribute usually specifies if the resource:

- must exist (ensure ⇒ present)
- must NOT exist (ensure ⇒ absent)

Some types of resources define additional states for this attribute. **File** resources can have, in addition, the following values for **ensure**:

- directory
- link
- file

Define a manifest that creates a symbolic link to the **/tmp/my_file** file.
The resource must also have the **target** attribute.

Use the Puppet documentation for the file type resource.

Authorized SSH key

In a manifest, define a resource with the type **ssh_authorized_key**.
The resource must allow the user **student** from the physical machine to authenticate as the **student** user on the VM, without a password.
If it doesn't already exist, the key pair for the **student** user must be generated beforehand.
Then, run the command `ssh-add ~/.ssh/id_rsa`

Use the Puppet documentation for the resource type ssh_authorized_key.

Resource Dependency

A Puppet manifest can contain declarations for multiple resources, but the order in which they are applied is not strictly enforced.

There are cases in which we have to make sure that a resource is applied before another (as an example, a package is installed before starting the service).

In these situations, we have to define resource dependencies.

Before / Require

We modify the previously created manifest:

```
file {'my_file':
```

```
  path    => '/tmp/my_file',
  ensure  => present,
  mode    => '0640',
  content => "File created using Puppet.",
}

notify {'my_notify':
  message => "File /tmp/my_file has been synced",
  require => File['my_file'],
}
```

- The **notify** resource defines a message that will be shown when its declaration is evaluated
- The dependency between resources is defined with the **require** attribute. In the previous example, the **my_file** resource is evaluated before the **my_notify** resource.

Modify the **/tmp/my_file** file and then apply the manifest described above. Notice the order in which the resources are evaluated.
An equivalent syntax would be to use the **before** attribute in the **my_file** resource:

```
file {'my_file':
  path    => '/tmp/my_file',
  ensure  => present,
  mode    => '0640',
  content => "File created using Puppet.",
  before  => Notify['my_notify'],
}

notify {'my_notify':
  message => "File /tmp/my_file has been synced",
}
```

Notify / Subscribe

For some resources we need a "refresh" action (as an example, a service that has to be restarted).

If in addition to resource dependency, we want to "refresh" the second resource when the first one is changed, we must:

- use **notify** instead of **before**, or
- use **subscribe** instead of **require**

An example would be restarting the SSH service when its configuration file has been changed:

```
file { '/etc/ssh/sshd_config':

  ensure => file,

  mode   => '0600',

  source => '/root/config-files/sshd_config',

}

service { 'sshd':

  ensure    => running,

  enable    => true,

  subscribe => File['/etc/ssh/sshd_config'],

}
```

Create a Puppet manifest with the previous code, then modify the **/etc/ssh/sshd_config** file and apply the manifest.

Equivalent syntax

Instead of **before / require** or **notify / subscribe**, we can use the operators: "->" or "~>" Example:

```
file {'my_file':

  path    => '/tmp/my_file',

  ensure  => present,

  mode    => '0640',

  content => "File created using Puppet.",

}
```

```
->

notify {'my_notify':

  message => "File /tmp/my_file has been synced",

}
```

Be careful when typing `~>` on a new line, as the sequence `<enter>~.` - i.e., pressing enter, followed by tilde (`~`) and period (`.`) - will immediately terminate the ssh connection.

Design Patterns: Package / File / Service

Package / File / Service

In many situations, Puppet is used to make sure that a certain system service is installed, started and with the appropriate configuration.

The above use case can be implemented with 3 resources:

- package
- file
- service

Between the first 2 we have a "**before / require**" relation, and between the last 2 there is a "**notify / subscribe**".
Create the following manifest which implements this design pattern for the SSH service, and then apply the manifest:

```
package { 'openssh-server':

  ensure => present,

}

->

file { '/etc/ssh/sshd_config':

  ensure => file,

  mode   => '600',

  source => '/root/config-files/sshd_config',

}

~>

service { 'sshd':

  ensure     => running,
```

```
  enable     => true,

}
```

Modify various states of the "package / file / service" triplet and reapply the manifest. Examples:

- uninstall/purge the packet
- change the configuration file
- stop the service

Task - Apache

Create a "package / file / service" manifest for the Apache service.

The configuration file must have a copy of the current file as a source. An example configuration is **/root/config-files/apache2.conf**

In Debian, the package for the Apache server is called **apache2**, and the configuration file is **/etc/apache2/apache2.conf**

Variables and Conditional Statements

Variables

In order to define a variable in Puppet, we use the syntax `$variable`, both for assignment and referencing.
We change the manifest for the **my_file** file, defining the contents of the file as a variable.

```
$my_content = "File created using Puppet."


file {'my_file':

  path    => '/tmp/my_file',

  ensure  => present,

  mode    => '0640',

  content => $my_content,

}
```

Facts

In addition to user-defined variables, Puppet defines some system variables. These are called **facts**. In order to see all of these variables, we use the command **facter**.

```
[root@puppet ~]# facter
```

```
disks => {
  fd0 => {
    size => "4.00 KiB",
    size_bytes => 4096
  },
  sda => {
    model => "QEMU HARDDISK",
    size => "8.00 GiB",
    size_bytes => 8589934592,
    vendor => "ATA"
  },
  sr0 => {
    model => "QEMU DVD-ROM",
    size => "1.00 GiB",
    size_bytes => 1073741312,
    vendor => "QEMU"
  }
}
dmi => {
  bios => {
    release_date => "04/01/2014",
    vendor => "SeaBIOS",
    version => "1.10.2-1ubuntu1"
  },
...
```

If

An example of using system variables is when taking decisions based on the value of some of them.

The following manifest ensures that the NTP service:

- is started if the system is a physical machine
- is stopped if the system is a virtual machine

The decision is taken based on the value of the `$is_virtual` system variable.

```
if str2bool("$is_virtual") {

  service {'ntp':

    ensure => stopped,

    enable => false,

  }

}

else {

  service { 'ntp':

    name       => 'ntp',

    ensure     => running,

    enable     => true,

    hasrestart => true,

    require => Package['ntp'],

  }

}
```

Puppet has a modular implementation, and some functionality is provided through classes, some of which may be provided by certain modules. To use the `str2bool` function, you must install the `puppet-module-puppetlabs-stdlib` module using the `apt` package manager.

Apply the manifest and notice the state of the NTP service.

Manifest for installing NTP

First, uninstall the NTP server from the virtual machine.

Then, write a manifest that:

- installs the NTP server packet
- ensures that the NTP server is started (the service name depends on the Linux distro)

Use the **case** conditional statement.
Check the documentation for the case statement. Depending on the version of puppet you use, the way the facter distributes the information in dictionaries may differ. For example, in older versions, the 'architecture' was a top-level variable, while in others it was moved under the **os** dictionary (i.e., it was accessed using `$os['architecture']`)

For Ubuntu/Debian, the service is called **ntp** and for RedHat/Fedora it is **ntpd**
Download the configuration file:

- Debian/Ubuntu: ntp.conf.debian
- RedHat/Fedora: ntp.conf.el

Ansible Install & Configuration

Ansible is a configuration management and provisioning tool, similar to Puppet. It uses SSH to connect to servers and run the configured tasks.

As opposed to Puppet, where each host manages its own data (services, users, files, etc.), and can optionally connect to a remote host to retrieve manifest files for configuration, Ansible is used to push the configuration from a central system to other hosts. An advantage of Ansible is that it does not require a specific service daemon to be installed before being able to configure the hosts. Operation is achieved through Python scripts for remote Linux hosts, or Powershell scripts for remote Windows hosts.

On the **SCGC VM** we are going to install and configure Ansible.

```
student@scgc:~$ sudo apt update
student@scgc:~$ sudo apt install -y ansible
# Required to use password authentication. By default, ansible requires authentication through SSH keys
student@scgc:~$ sudo apt install -y sshpass
```

Check that the package was successfully installed by running the command:

```
student@scgc:~/scgc$ ansible --version

ansible 2.5.1

  config file = /etc/ansible/ansible.cfg

  configured module search path = [u'/home/student/.ansible/plugins/modules',
u'/usr/share/ansible/plugins/modules']

  ansible python module location = /usr/lib/python2.7/dist-packages/ansible

  executable location = /usr/bin/ansible

  python version = 2.7.17 (default, Nov  7 2019, 10:07:09) [GCC 7.4.0]
```

Configuring host labels

Ansible has a default inventory file used to define which servers it will be managing. After installation, there's an example one you can reference at **/etc/ansible/hosts** (initially fully commented out).
We are going to add 2 **labels** to the hosts file. One containing the local address, named **thishost** and another one of the previously used VM, named **remote**.

```
student@scgc:~$ cat /etc/ansible/hosts | grep -A 5 thishost

--

--

[thishost]

127.0.0.1



[remote]

10.0.0.2

--

--
```

Testing hosts availability

Let's start running Tasks against a server.

Running against localhost:

```
student@scgc:~$ ansible thishost --connection=local -m ping

127.0.0.1 | SUCCESS => {

    "changed": false,

    "ping": "pong"

}
```

Running against the remote VM:

```
student@scgc:~$ ansible --ask-pass --user=student remote -m ping

SSH password:

10.0.0.2 | SUCCESS => {

    "changed": false,
```

```
    "ping": "pong"
}
```

In either case, we can see the output we get from Ansible is some JSON which tells us if the Task (our call to the ping module) made any changes and the result.

Let's cover these commands:

- `thishost`, `remote` - Use the servers defined under this label in the hosts inventory file. The `all` argument can be used to run the ruleset on all defined hosts.
- `--connection=local` - Run commands on the local server, not over SSH
- `-m ping` - Use the "ping" module, which checks if the host can be accessed. Using `ping` with `--connection=local` does not make sense, as the option is used when running commands on the host that is issuing commands. It normally attempts to connect to the host via SSH.
- `--ask-pass --user=student` - SSH connection parameters: interactive password input, login as **student** user

Ansible Facter

Ansible has a fact gathering system similar to Puppet. To extract facts about the remote host we can use the `setup` module. The information is returned as Python dictionaries, where values can be strings, arrays, or other dictionaries.

```
student@scgc:~$ ansible --ask-pass --user=student remote -m setup
SSH password:
10.0.0.2 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "10.0.0.2"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::5054:ff:fe12:3451"
        ],
```

```
        "ansible_apparmor": {

            "status": "enabled"

        },

        "ansible_architecture": "x86_64",

        "ansible_bios_date": "04/01/2014",

        "ansible_bios_version": "1.10.2-1ubuntu1",

...
```

The information in the facter, can be used in playbooks - configuration files written in YAML that act as scripts for ansible. The syntax used to expand all variables - including those created by the facter - is `"{{ variable }}"`. For example, `"{{ ansible_facts.hostname }}"` will be expanded to the hostname, as identified by the facter.

Two-factor Authentication for SSH

We plan to enable the use of two-factor authentication for SSH through the use of Google's Authenticator mobile application. To do this, we need to create a Google Authenticator configuration file on the host. To create one with sensible defaults, you can use the following commands:

```
student@scgc:~$ sudo apt install libpam-google-authenticator qrencode
student@scgc:~$ echo -e "y\ny\ny\nn\ny" | google-authenticator
```

The commands above will create a configuration file for the authenticator, that will generate time-based codes, will update the `~/.google-authenticator` file, disallow multiple users and enable rate limiting. The fourth option (the **n** in the string passed to the `google-authenticator` binary) disables longer-lasting codes (this option is only useful when the phone and/or the server's time sync protocols are not working properly). For more details, consult DigitalOcean's tutorial on how to set it up here.
After running the command, the terminal will display the secret key as both a large QR code, and text. Please open the Google Authenticator app on your phone, and scan the QR code or enter it manually.

We will copy the Google Authenticator's configuration file, the configuration for the SSH daemon, and the PAM configuration file for the SSH service:

```
student@scgc:~$ mkdir config-files
student@scgc:~$ cd config-files
student@scgc:~/config-files$ cp /home/student/.google_authenticator .
student@scgc:~/config-files$ cp /etc/pam.d/sshd .
student@scgc:~/config-files$ cp /etc/ssh/sshd_config .
```

The Authenticator configuration file is sensitive information! It MUST have `0600` permissions (only the user must be able to access it), and it is usually not a good idea to copy it to another server. From a security point of view, it is similar to copying a private SSH key to another server. Make sure you copy the configuration only to servers you trust.

We will use the files created above as templates to replicate on the server(s). This example will only use the `10.0.0.2` VM as a target machine. We must set up the configuration files to use password + the a One Time Password (OTP) generated by the authenticator. Make sure the configuration files for sshd and PAM look as below:

```
student@scgc:~/config-files$ grep -B 5 -A 3 'pam_google_authenticator.so' sshd
# PAM configuration for the Secure Shell service

# Standard Un*x authentication.
@include common-auth
# 2-FA authentication with Google Authenticator
auth       required     pam_google_authenticator.so

# Disallow non-root logins when /etc/nologin exists.
account    required     pam_nologin.so
```

The changes to the PAM configuration file above make using the Google Authenticator module mandatory. It is placed after 'common-auth', so the code will be required **after** entering the password.

```
student@scgc:~/config-files$ grep -B 5 -A 3 '^ChallengeResponseAuthentication'
sshd_config
#PasswordAuthentication yes
#PermitEmptyPasswords no

# Change to yes to enable challenge-response passwords (beware issues with
# some PAM modules and threads)
ChallengeResponseAuthentication yes
AuthenticationMethods publickey keyboard-interactive

# Kerberos options
```

The changes to the configuration file above make using the challenge response to allow PAM to use multiple modules with challenge responses (i.e., password and authentication code in our case); the use of keyboard-interactive authentication is mandatory if more than just the password is required.

Ansible can use privilege escalation using the **become** keyword at certain tasks, or all tasks. If the user cannot run sudo with a password, the `ansible_become_password` variable must be set. To do this we will use a vault - a type of file that encrypts strings through a password - to store the password, instead of adding it as plain text to the playbook. To create a vault, use the following command, and write the key-value pair for the password in the file it opens using the default editor:

```
student@scgc:~/config-files$ ansible-vault create puppet.vault
New Vault password: # Enter vault password
Confirm New Vault password: # Confirm vault password

# In the opened file
ansible_become_password: student
```

After closing the vault, you can see that the information in it is encrypted.

To install the SSH daemon on the remote machine, and set it up for use with the Google Authenticator we will use the following playbook, saved as `sshd.yaml`:

```
---
- hosts: remote
```

```
  remote_user: student

  tasks:
  # include sudo password vault
  - name: Set host variables
    include_vars: "{{ ansible_facts.hostname }}.vault"

  # Install sshd and make sure it is at the latest version using the package
  # manager identified by ansible
  - name: Ensure sshd is at the latest version
    package:
      name: openssh-server
      state: latest
    become: yes

  # Install google authenticator module and make sure it is at the latest
version
  - name: Ensure google-authenticator is at the latest version
    package:
      name: libpam-google-authenticator
      state: latest
    become: yes

  # Copy the google authenticator configuration file
  # The file MUST be located in the user's home directory with permissions 0600
  - name: Copy Google Authenticator config file
    copy:
      src: /home/student/config-files/.google_authenticator
      dest: /home/student/.google_authenticator
      mode: 0600
      owner: student
      group: student

  # Overwrite sshd configuration file. Make sure the challenge response setting
  # is enabled, and keyboard-interactive is a valid authentication method
  - name: Write the sshd configuration file
    template:
      src: /home/student/config-files/sshd_config
      dest: /etc/ssh/sshd_config
    become: yes
    notify:
      - restart sshd

  # Overwrite the PAM configuration file. Make sure that authentication through
  # google authenticator is required
  - name: Write the PAM configuration file
    template:
      src: /home/student/config-files/sshd
      dest: /etc/pam.d/sshd
    become: yes
    notify:
      - restart sshd

  handlers:
  # Handlers that are invoked when the configuration files change -
  # restart the sshd service
  - name: restart sshd
```

```
    service:
      name: sshd
      state: restarted
    become: yes
```

The playbook attempts to include the file named `{{ ansible_facts.hostname }}.vault` - which resolves to `puppet.vault` for the VM. To run it, we use the `ansible-playbook` command, with the `--ask-pass`, to ask for the SSH authentication password, and the `--ask-vault-pass` to provide the decryption password for the vault.

```
student@scgc:~/config-files$ ansible-playbook --ask-vault-pass --ask-pass
sshd.yml
SSH password:
Vault password:

PLAY [remote]
*********************************************************************

TASK [Gathering Facts]
*********************************************************
ok: [10.0.0.2]

TASK [Set host variables]
*********************************************************
ok: [10.0.0.2]

TASK [Ensure sshd is at the latest version]
***************************************
ok: [10.0.0.2]

TASK [Ensure google-authenticator is at the latest version]
**********************
changed: [10.0.0.2]

TASK [Copy Google Authenticator config file]
***********************************
changed: [10.0.0.2]

TASK [Write the sshd configuration file]
*****************************************
changed: [10.0.0.2]

TASK [Write the PAM configuration file]
*****************************************
changed: [10.0.0.2]

RUNNING HANDLER [restart sshd]
***************************************************
changed: [10.0.0.2]

PLAY RECAP
*********************************************************************
10.0.0.2                   : ok=8    changed=5    unreachable=0    failed=0
```

You should now be able to login using the password and the Google Authenticator.

```
student@scgc:~$ ssh student@10.0.0.2
```

```
Password:
Verification code:
Password:
Verification code:
Linux puppet 4.19.0-8-amd64 #1 SMP Debian 4.19.98-1 (2020-01-26) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
student@puppet:~$
```

After adding two-factor authentication, Ansible will no longer be able to access the VM using password authentication, since the password is read by ansible before actually attempting to access the server, and `sshpass` is not aware it is required.

SPD

parallel crunch

ISBN: 978-606-515- 956-3