PUPPET

SOFTWARE AVANZADO – DICIEMBRE 2020

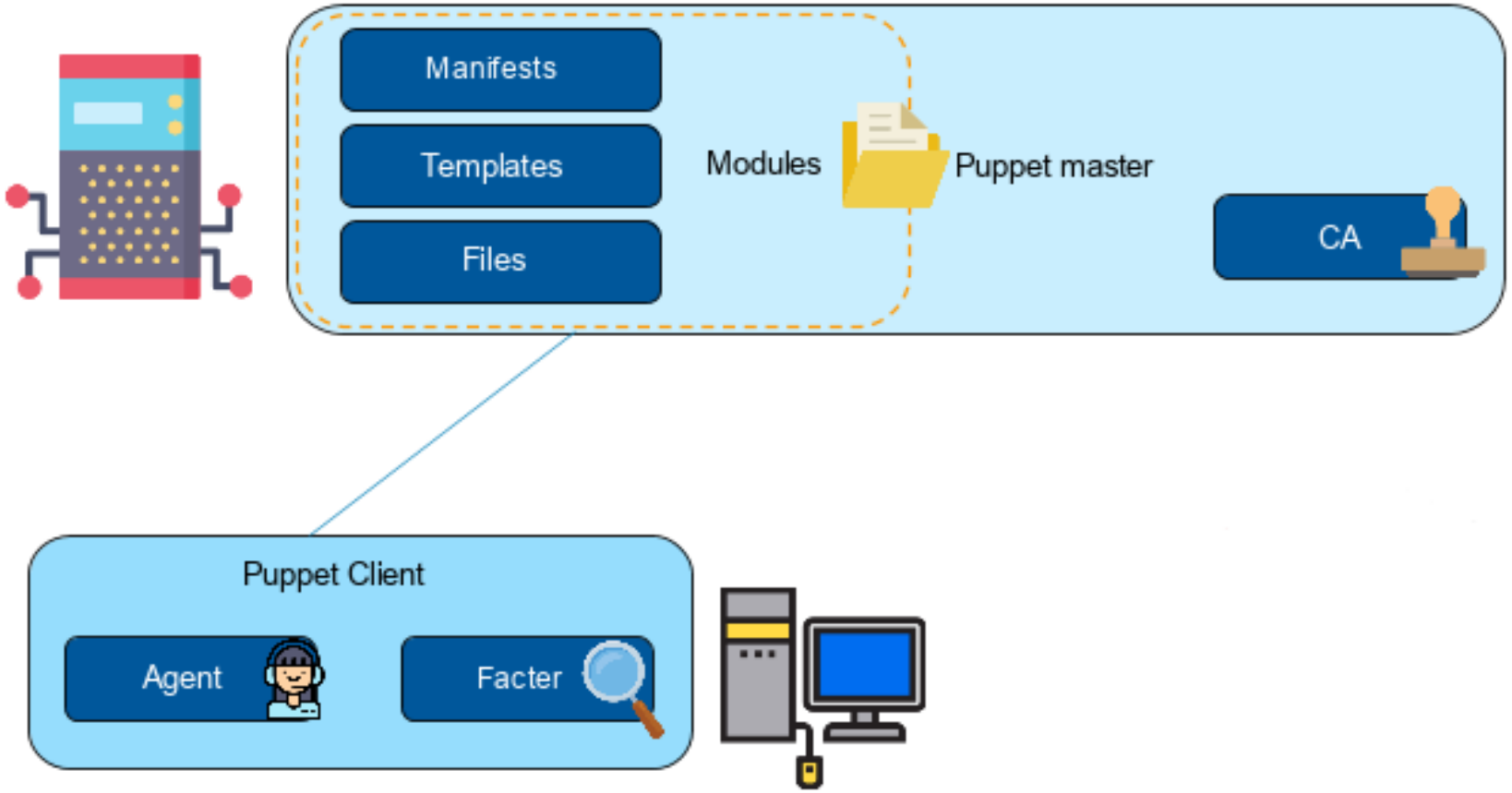
QUE ES PUPPET

Puppet es una herramienta de gestión de la configuración que garantiza que todos los sistemas estén configurados en un estado deseado y predecible.

También puede utilizar Puppet como herramienta de implementación, ya que puede implementar software automáticamente en el sistema. Puppet implementa la infraestructura como código, lo que significa que puede probar el entorno y asegurarse de que se implemente con precisión.

COMPONENTES DE PUPPET

El entorno de Puppet se puede dividir en dos partes, en el entorno del servidor principal (Master) y el entorno del cliente (AGENT).



COMPONENTES SERVIDOR PRINCIPAL (MASTER)

- **Los manifiestos:** son los códigos reales para configurar los clientes.
- **Las plantillas:** combinan código y datos para representar un documento final.
- **Los archivos:** son el contenido estático que pueden descargar los clientes.
- **Los módulos:** son una colección de manifiestos, plantillas y archivos.
- **La autoridad certificadora:** permite al maestro firmar los certificados enviados por el cliente.

COMPONENTES SERVIDOR CLIENTE

Servidor consta de dos componentes Agent y Facter.

- **Agent:** Medio por el cual se interactúa continuamente con el servidor maestro para garantizar que los certificados se actualicen de manera adecuada.
- **Facter:** Medio por el cual se recopila el estado actual del cliente que se utiliza y lo comunica a través del agente.

ELEMENTOS BASICOS EN PUPPET

Resources

Un recurso describe algo sobre el estado del sistema, como que debería existir un determinado usuario o archivo, o debería instalarse un paquete.

El código de puppet se compone principalmente de declaraciones de recursos.

```
user { 'mitchell':
  ensure     => present,
  uid        => '1000',
  gid        => '1000',
  shell      => '/bin/bash',
  home       => '/home/mitchell'
}
```

DECLARACIONES DE RECURSOS

- **Exec:** para ejecutar comandos, como **apt-get**
- **Package:** para instalar paquetes a través de apt
- **File:** para asegurarse de que existen ciertos archivos
- **Fact:** variables globales que contienen información sobre el sistema, como interfaces de red y sistema operativo.
- **Service:** se utiliza para activar cambios en el estado del servicio, como reiniciar o detener un servicio.

Manifests

Los programas de puppet se llaman manifiestos. Los manifiestos son básicamente una colección de declaraciones de recursos. Los manifiestos se componen de código de puppet y sus nombres de archivo usan la extensión **.pp**. El manifiesto principal predeterminado en Puppet instalado a través de apt es **/etc/puppet/manifests/sites.pp**

```
exec { 'apt-get update':
    command => '/usr/bin/apt-get update'
}

package { 'vim':
    ensure => 'installed'
    require => Exec['apt-get update']
}
```

DEPENDENCIA DE LOS RECURSOS

Al escribir manifiestos, es importante tener en cuenta que Puppet no evalúa los recursos en el mismo orden en que están definidos. Esta es una fuente común de confusión para quienes están comenzando con Puppet. Los recursos deben definir explícitamente la dependencia entre ellos; de lo contrario, no hay garantía de qué recurso se evaluará y, en consecuencia, se ejecutará primero.

Digamos que desea ejecutar un comando, pero primero debe asegurarse de que se instale una dependencia:

```
package { 'python-software-properties':
    ensure => 'installed'
}

exec { 'add-repository':
    command => '/usr/bin/add-apt-repository ppa:ondrej/php5 -y'
    require => Package['python-software-properties']
}
```

Digamos que necesita asegurarse de que una tarea se ejecute antes que otra. Para un caso como este, podemos usar la opción **before**

```
package { 'curl':
    ensure => 'installed'
    before => Exec['install script']
}

exec { 'install script':
    command => '/usr/bin/curl http://example.com/some-script.sh'
```

Classes

En Puppet, las clases son bloques de código que se pueden llamar en un código en otro lugar. El uso de clases le permite reutilizar el código de Puppet y puede facilitar la lectura de manifiestos.

Definición de clase

Una definición de clase es donde vive el código que compone una clase. La definición de una clase hace que la clase esté disponible para usarse en manifiestos, pero en realidad no evalúa nada.

```
class example_class {
  ...
  code
  ...
}
```

Declaración de clase

Una declaración de clase ocurre cuando se llama a una clase en un manifiesto. Una declaración de clase le dice a Puppet que evalúe el código dentro de la clase. Las declaraciones de clase vienen en dos tipos diferentes: normal y similar a un recurso.

Una declaración de clase normal ocurre cuando la palabra clave **INCLUDE** se usa en el código de Puppet

```
include example_class
```

Modules

Un módulo es una colección de manifiestos y datos (como facts, archivos y plantillas) y tienen una estructura de directorio específica.

Los módulos son útiles para organizar su código Puppet, porque le permiten dividir su código en múltiples manifiestos. Se considera una buena práctica utilizar módulos para organizar casi todos los manifiestos de Puppet.

Para agregar un módulo a Puppet, se debe colocar en el directorio **/etc/puppet/module**

RESUMEN

- **Puppet Master :** el servidor maestro que controla la configuración en los nodos
- **Nodo Puppet Agent :** un nodo controlado por un Puppet Master
- **Manifiesto :** un archivo que contiene un conjunto de instrucciones a ejecutar.
- **Recurso:** una porción de código que declara un elemento del sistema y cómo se debe cambiar su estado. Por ejemplo, para instalar un paquete, necesitamos definir un recurso de paquete y asegurarnos de que su estado esté establecido en "instalado"
- **Módulo:** una colección de manifiestos y otros archivos relacionados organizados de una manera predefinida para facilitar el intercambio y la reutilización de partes de un aprovisionamiento.
- **Clase:** al igual que con los lenguajes de programación normales, las clases se utilizan en Puppet para organizar mejor el aprovisionamiento y facilitar la reutilización de partes del código.

ENLACES

https://www.digitalocean.com/community/tutorials/configuration-management-101-writing-puppet-manifests

https://www.digitalocean.com/community/tutorials/getting-started-with-puppet-code-manifests-and-modules