PICA: Perl Installation and Configuration Agent

Una solución inteligente para administración de sistemas

Esteban Manchado Velázquez

Miguel Armas del Río

1. Introducción

PICA es una pequeña herramienta escrita en Perl para ayudar a los administradores de sistemas a mantener coherente la configuración de sus máquinas. Además de en el lenguaje Perl, PICA se basa en muchas ideas de la herramienta PIKT, y en SSH, para enviar todos los datos de unas máquinas a otras de forma segura.

1.1. ¿Por qué escribimos PICA?

Necesitábamos una forma de mantener una copia de la configuración de los servidores en una de las máquinas, para luego poder copiar los ficheros necesarios a las que hiciera falta. Hasta el momento estábamos usando PIKT, pero había muchas cosas que no nos gustaban. La idea encajaba perfectamente con nosotros, pero necesitábamos más potencia, más flexibilidad, más seguridad y más comodidad. Las alternativas tampoco nos gustaban por diferentes razones, como veremos más adelante.

1.2. ¿Por qué no PIKT?

PIKT es un programa escrito en C cuyo cometido es copiar ficheros a máquinas remotas, instalar alarmas y ejecutar órdenes remotamente. Es una herramienta bastante útil para administradores de sistemas, y nosotros la usamos durante un tiempo, pero

decidimos escribir PICA y dejar de usar PIKT por varias razones, entre las que destacan:

- Uso de *portmapper* para las comunicaciones, por el historial de problemas de seguridad y la apertura de un servicio que no usábamos para nada más.
- Imposibilidad de enviar ficheros binarios.
- Sistema de condiciones muy pobre.
- Imposibilidad de generar partes de los ficheros dinámicamente.
- Varios fallos graves en el propio programa, debidos en parte al perfil de su creador (administrador de sistemas y no programador).
- Falta de una forma sencilla de distribuir ficheros sin tener que adaptarlos a una sintaxis determinada.
- Necesidad de decidir de antemano qué servidores iban a tener ficheros distribuidos automáticamente, ya que hay que instalar un cliente en cada una de estas máquinas.
- En general, un enfoque muy complejo, difícil de entender y un programa difícil de modificar para resolver un problema muy simple.

1.3. Alternativas a PIKT

Algunas de las alternativas a PIKT (y PICA) son cfengine, arusha, big sister, y gap. Ninguna de ellas satisfacía todas nuestras necesidades de administración, o resultaron demasiado complicadas para el problema que queríamos resolver.

Sin duda, y obviando PIKT, cfengine es la herramienta más parecida a PICA. Sus inconvenientes: tener que aprender muchos conceptos nuevos, compuesto por varios programas (instalación más compleja) y necesidad de aprender un lenguaje de configuración específico para usar la herramienta. Además, aparentemente no permite copiar los ficheros variando su contenido según la máquina o cualquier otra condición.

Arusha no termina de resolver nuestros problemas, por ser un «marco» en el que trabajar y construir las soluciones de administración. Por si fuera poco, necesita también conocer un lenguaje de configuración específico (en XML) para poder usarlo. PICA da una solución mucho más simple, más sencilla de aprender y que puede usarse al poco tiempo de instalar el programa.

Big sister es una aplicación de monitorización, que nada tiene que ver con PICA: ambas herramientas son complementarias y podrían usarse conjuntamente.

Por último, gap es un proyecto demasiado ambicioso para nuestros problemas diarios. Al igual que algunas de las otras alternativas, necesita aprender su forma de trabajo, y parece más enfocado hacia la monitorización y ejecución remota que a la simple copia de ficheros.

1.4. ¿Dónde conseguir PICA?

La página oficial de PICA está en SourceForge, en la URL http://pica.sf.net, y se distribuye por la licencia GPL. Hay disponibles paquetes Debian y RPM, además del paquete original *.tar.gz*.

2. Fundamentos de PICA

Teniendo en cuenta nuestra experiencia usando PIKT, queríamos intentar evitar los errores de éste en nuestro nuevo proyecto. Por esta razón, aprovechamos ideas del PIKT, pero lo diseñamos todo desde cero. Usamos un lenguaje de programación diferente (Perl), y unos ficheros de configuración y de entrada con una sintaxis completamente distinta, diseñada acorde con nuestros objetivos. Queríamos mantener la filosofía del PIKT, pero cambiar el enfoque y la mentalidad en cuanto a las herramientas.

2.1. Principios de diseño

Los principios de diseño de PICA son:

1. Deben usarse herramientas existentes que funcionen bien. De ahí que se use SSH para todas las transferencias.
2. La seguridad tiene que tenerse en cuenta. Toda la información va cifrada, y no se requiere que PICA tenga permisos especiales para hacer su trabajo.
3. Todo debe ser lo más flexible posible, para no tener que estar haciendo extensiones de la sintaxis o de la funcionalidad una y otra vez.
4. La simplicidad y linealidad son buenas a largo plazo. Por ello, todos los ficheros de configuración se preprocesan antes de leerse, y todas las condiciones del preprocesador de PICA no son más que expresiones en Perl genéricas. Los casos

particulares comunes se tratan con funciones en Perl declaradas visiblemente en un paquete para tal efecto.

2.2. Conceptos importantes en PICA

A la hora de trabajar, PICA se apoya en los siguientes conceptos:

- Máquinas. Para PICA, conceptualmente las máquinas siempre son remotas, es decir, todo se copia siempre mediante órdenes **ssh**.
- Objetos. Son los ficheros que PICA distribuye a las máquinas. Estos ficheros pueden ser ficheros normales o *alarmas*, que tienen un tratamiento especial en algunos casos. Hablaremos de éstas más tarde.
- Grupos. Podemos agrupar tanto las máquinas como los objetos con nombres fáciles de manejar. Además, las alarmas son grupos implícitos, que pueden contener otros objetos (ficheros normales, nunca otras alarmas). Estos objetos se tratan como dependencias, es decir, siempre que se instale esa alarma deben instalarse también las dependencias.

3. Posibilidades de PICA

PICA se diseñó para resolver los siguientes problemas:

- Copia de ficheros de una máquina central al resto. Los ficheros debían tener condicionales y alguna forma genérica de decidir qué iba a formar parte del contenido, dependiendo de la máquina donde fuera a instalarse y de otros parámetros no decididos de antemano por nosotros.
- Ejecución remota de órdenes, tanto distribuidas con el programa como arbitrarias, instaladas en la máquina destino.
- Gestión de alarmas en cada máquina. Entendemos por alarmas pequeños programas que comprueban el estado del sistema y avisan de cualquier anomalía, y que optativamente lo resuelven.

Todo esto se hace a petición explícita del administrador. Es decir, que cada vez que se quiere copiar, ejecutar, borrar o listar un objeto, el administrador llama desde la línea de órdenes a PICA con los parámetros apropiados (veremos la sintaxis de llamada más

adelante). Para funcionar, PICA *no* necesita instalar ningún programa en las máquinas a las que se va a copiar ficheros, o sobre las que se va a actuar. El único requisito es tener un servidor SSH instalado (y, a estas alturas, ¿qué servidor que nos preocupe no tiene ya el SSH?).

3.1. Opciones de llamada

Las operaciones que PICA puede ejecutar son:

- *Instalación* de cualquier objeto o grupo de objetos (ficheros o alarmas). Los ficheros a copiar se llaman *ficheros de distribución*, y pueden contener directivas para ser preprocesados. Por defecto, los ficheros de distribución se buscan en el directorio `$picasrc` especificado en el fichero de configuración `pica.conf`.
- *Ejecución* de un objeto remotamente, como si fuera una orden del sistema (PICA busca la ruta correcta en la definición del objeto), o una orden arbitraria que esté instalada en la máquina remota (muy útil para actualizar la configuración de un servicio que se está ejecutando).
- *Borrado* de objetos.
- *Cálculo de diferencias* de un objeto (diferencia entre la versión que se instalaría y la que hay realmente en la máquina).
- *Listado* de objetos. PICA puede listar los objetos que hay instalados en una máquina determinada, para saber si están instalados y para comprobar sus permisos y propietarios.

Por otro lado, hay tres opciones generales de PICA, aplicables a todas las operaciones:

- Modo de depuración. PICA informará de todo lo que hace a medida que lo va haciendo, y deja los ficheros de configuración preprocesados en el directorio temporal. Además, *no ejecuta ninguna orden, ni instala ni borra ningún fichero*, sólo imprime en pantalla qué debería ejecutar en cada momento.
- Simulación. Esta opción permite simular, sin producir realmente ningún resultado, cualquier acción. Puede informar de errores, e imprime las órdenes que ejecutaría en condiciones normales. Es similar a la opción `-n` del programa make.
- Verbosidad. Imprime en pantalla mucha más información que de costumbre. Conjugando esta opción con la anterior se puede conseguir algo parecido a la de depuración, pero sin dejar ficheros en el directorio temporal ni cargar *tanto* la pantalla.

4. Configuración de PICA

PICA tiene actualmente tres ficheros de configuración. Su sintaxis es parecida a la de los ficheros de DNS, por ser clara y legible y resultar familiar a los administradores de sistemas. En uno especificamos algunos datos necesarios para el propio ejecutable de PICA, y en los otros dos especificamos las máquinas y los objetos que vamos a manejar, respectivamente. Los ficheros se llaman `pica.conf`, `hosts.conf` y `objects.conf`. Aunque no es importante para entender el funcionamiento general de PICA, sí es importante a la hora de usarlo saber que los ficheros se leen en ese orden.

4.1. `pica.conf`

En este fichero se especifican las rutas de varios ejecutables que PICA necesita, algunos directorios locales importantes para el programa (dónde van los ficheros temporales, por ejemplo) y una lista de directorios que no deben borrarse:

Ejemplo 1. Ejemplo de `pica.conf`

```
# pica.conf

defaults {
   # Directorios (locales) importantes para el ejecutable
   picaroot    = /var/lib/pica;  # Raíz de los ficheros para PICA
   picatmp     = /var/lib/pica/tmp; # Ficheros temporales
   # Donde están los ficheros a distribuir
   picasrc     = /var/lib/pica/src;
   # Ficheros de inclusión del preprocesador
   picainclude = /var/lib/pica/include;

   # Rutas de algunos programas usados por PICA
   sshpath   = '/usr/bin/ssh -q';  # Ruta al ssh (y opciones)
   diffpath  = '/usr/bin/diff';
   tarpath   = '/bin/tar';
   rsyncpath = '/usr/bin/rsync';

   # Directorios protegidos (nunca se borrarán ni se cogerán
   # como directorios temporales)
   protecteddirs {
      /,
      /bin,
      /usr/bin,
      /lib,
```

```
      /usr/lib
   }
}
```

4.2. `hosts.conf`

El fichero de descripción de máquinas permite especificar los nombres de las que vamos a manejar y sus propiedades, junto con la definición de los posibles grupos que queramos (para facilitarnos el referirnos a éstas al llamar a PICA). Por defecto, se intentará conectar con las máquinas por el nombre que le demos en el fichero. Si se quiere poner un nombre «lógico» diferente del nombre real de la máquina, se puede usar la propiedad `fqdn`.

Ejemplo 2. Ejemplo de `hosts.conf`

```
# hosts.conf

# Valores por defecto
defaults {
   method      =  'tar'; # método de copia (tar, rsync or ssh)

   # Variables globales
   vars {
      docdir  = '/var/www/html/sysadm';
   }
}

# Definición de máquina
host mimaquina;
host miotramaquina {
   method = ssh;
   fqdn   = nombre.completo.net;
}

# Definición de grupo
hostgroup migrupo {
    members { mimaquina, miotramaquina }
}

hostgroup miotrogrupo {
```

```
    members { miotramaquina }
}
```

4.3. `objects.conf`

El fichero de objetos probablemente será el que modifiquemos con más frecuencia, y desde luego es el que tiene la sintaxis más completa. Al igual que en el fichero de máquinas, especificamos la lista de objetos que podemos distribuir con sus propiedades y los grupos que queramos crear (aunque se declaran de forma diferente, ver ejemplo).

Ejemplo 3. Ejemplo de `objects.conf`

```
# objects.conf

# Inclusión de ficheros
#include <pifia.conf>

# Valores por defecto
defaults {
    uid = 0;
    gid = 0;
    perms = 644;
    verbatim = 0;  # Si vale 1 no se aplica el preprocesador
                   # antes de copiar
}

# Ficheros de prueba
file pica-rules {
    path = '/var/lib/pica/rules'; # Donde se instalan en la máquina
                                  # remota
    source = 'pica-rules'; # Donde está el "fuente" localmente

    # Variables locales para el fichero (se podrán sustituir
    # en el fichero de distribución, ver ejemplos del preprocesador
    # más adelante)
    vars {
       myvar         = 'foo';
       limit         = '2';
       mythirdvar    = 'enough';
    }
}
```

```
# Ficheros de autentificación RSA con SSH
group sshauth {
   file auth_keys {
      source = 'SSHAuth/authorized_keys';
      path   = '/root/.ssh/authorized_keys';
      perms  = '600';
   }
   # SSH v.2
   file auth_keys2 {
      source = 'SSHAuth/authorized_keys2';
      path   = '/root/.ssh/authorized_keys2';
      perms  = '600';
   }
}
```

5. Usos comunes de PICA

A continuación de presentan algunos de los casos comunes de uso de PICA, no sin antes explicar brevemente el formato de la llamada.

5.1. Formato general de la llamada a PICA

En general, el formato de una llamada a PICA se compone de: una operación y opciones generales, una lista de objetos sobre los que operar y una lista de máquinas sobre las que operar. La sintaxis es muy similar a la del PIKT, por aquello de no tener que aprender una nueva sintaxis, y porque nos pareció apropiada (al fin y al cabo, *queríamos* aprovechar todo lo posible del PIKT).

Las opciones y la operación van precedidas de un guión, al estilo de las opciones normales de UNIX. Pueden agruparse o separarse, por legibilidad.

Para especificar las listas de máquinas y objetos hay toda una sintaxis de aritmética, también tomada prestada del PIKT. Debido a esta aritmética, podemos ir sumando y restando máquinas, objetos y grupos de ambos. Para sumar o restar máquinas, precedemos la lista con `+H` o `-H`, y para sumar o restar objetos (ficheros o alarmas), `+F` o `-F`.

5.2. Ejemplos de llamadas a PICA

Algunos ejemplos terminarán de aclarar el uso de la aritmética de objetos y máquinas. Empecemos por el caso más común de todos, la instalación (copia) de un objeto (`pica-rules`) en una máquina (`demiurgo`):

Ejemplo 4. Llamada a PICA (instalación de `pica-rules` en `demiurgo`)

```
pica -i +F pica-rules +H demiurgo
```

Supongamos ahora que tenemos definido un grupo de máquinas con los servidores DNS (`dnsservers`) a las que queremos copiar todos los ficheros definidos para esas máquinas. El problema es que hay dos máquinas pertenecientes al grupo que están en pruebas o apagadas y por esta vez no vamos a procesarlas (`w2k` y `w98`). En ese caso, podríamos perfectamente «restarlas» del grupo con algo como:

Ejemplo 5. Resta de dos máquinas a un grupo

```
pica -i +F all +H dnsservers -H w2k w98
```

Nótese que en este ejemplo acabamos de introducir un concepto nuevo: el de grupos implícitos. Para hacernos la vida más fácil, PICA define algunos grupos implícitamente. Tanto para máquinas como para objetos está definido el grupo *`all`*, que se refiere a todas las máquinas/objetos (depende del contexto). También vale la pena comentar que los espacios de nombres de los objetos y las máquinas son completamente independientes, así que podemos tener *tanto* una máquina (o grupo de máquinas) como un objeto (o grupo de objetos) llamado «`dns`». Sin embargo, para evitar confusiones suele ser una buena costumbre no repetir nombres.

Si, en vez de copiar un objeto, queremos comparar la versión que se instalaría y la que está realmente instalada en todas las máquinas definidas, podríamos usar lo siguiente:

Ejemplo 6. Comparación de un objeto

```
pica -f +F fichero-importante +H all
```

La salida será un diff unificado entre ambas versiones.

El borrado de objetos es igual de simple, pero usando la opción `-t`:

Ejemplo 7. Borrado de objetos

```
pica -t +F fichero-menos-importante +H all
```

En la ejecución hay dos casos: por un lado, podemos ejecutar un objeto de PICA que sea una orden (si tiene permiso de ejecución y está instalado); por otro lado, podemos ejecutar órdenes instaladas en la máquina remota aunque no estén declaradas en `objects.conf`. Por ejemplo, supongamos que uno de los objetos que distribuimos es un programa llamado postupdate que actualiza la configuración del postfix. Si lo tuviéramos declarado como objeto de PICA, aunque no se instalara en un directorio dentro de la ruta de búsqueda de ejecutables, podríamos ejecutarlo en todas las máquinas del grupo `mailservers` escribiendo

Ejemplo 8. Ejecución del objeto `postupdate` en los componentes del grupo `mailservers`

```
pica -x +F postupdate +H mailservers
```

Si, por el contrario, lo único que queremos es ejecutar una orden cualquiera de la máquina (incluso con parámetros), podemos escribir algo como lo siguiente:

Ejemplo 9. Ejecución de órdenes remotas con parámetros

```
pica -x +F "killall netscape" +H alumnos-3
```

La razón de poner las comillas es que PICA interpreta diferentes parámetros como distintas órdenes a ejecutar. Si no pusiéramos comillas, intentaría ejecutar killall sin parámetros y la aplicación netscape en remoto.

Todo el manejo de PICA se basa en estos ejemplos. Podemos añadir objetos que nos solucionen problemas u objetos que sean órdenes para ejecutar remotamente, pero todo lo haremos con la instalación y ejecución remota. Incluso las alarmas, que veremos en el siguiente apartado, se basan en estos principios.

6. Gestión de alarmas: PIFIA

Llamamos «alarmas» a pequeños programas que comprueban el estado de los servicios del sistema y avisan de las posibles anomalías, intentando quizás recobrar el estado

normal. Todos los ficheros y convenciones de PICA en torno a las alarmas reciben el nombre de PIFIA (PICA Framework for Integrated Alarms).

Las alarmas se instalan como ficheros normales en las máquinas remotas, aunque tienen algunas propiedades adicionales y la forma de declararlas es diferente a la de los objetos corrientes. El trato homogéneo de todos los objetos tiene varias ventajas:

1. No añadimos conceptos nuevos ni el administrador tiene que recordar otra sintaxis.
2. Pueden instalarse diferentes versiones en distintas máquinas, si fuera necesario o conveniente.
3. Evitamos un trato especial a las alarmas, que de esta forma pueden entenderse como simples ficheros. Esto facilita la creación de programas de gestión de alarmas (por nosotros y por terceros) y facilita que, en caso de problemas, los administradores puedan cambiarlas, moverlas, borrarlas, o manipularlas de cualquier forma manualmente.

Una vez copiadas en la máquina destino se ejecutan periódicamente (como un trabajo *cron* normal y corriente), evitando así la dependencia de la conexión con el servidor «principal», del que se copiaron originalmente las alarmas.

6.1. Declaración de las alarmas

En el apartado de configuración ya hemos visto cómo declarar un objeto. Las diferencias entre los objetos normales y las alarmas son:

- Las alarmas tienen una propiedad obligatoria adicional, `priority`, que indica la frecuencia con la que se ejecuta. La prioridad puede ser «`Emergency`», «`Urgent`» y «`Warning`».
- En vez de la palabra `file`, las alarmas usan la palabra `alarm`.
- No puede especificarse el atributo `path`, dado que las alarmas se copian siempre a un directorio especial (que, hasta cierto punto, puede personalizar el administrador).
- Las alarmas añaden una construcción optativa llamada `uses`, para especificar *dependencias* de la alarma. Los ficheros especificados aquí, que se instalan siempre que se instala la alarma, son normalmente ficheros de configuración de las alarmas. Por supuesto, estos ficheros pueden declararse y copiarse aparte, pero se corre el peligro de copiar las alarmas y no los ficheros de configuración.
- Para no mezclar los ficheros de alarmas con los demás objetos, por defecto las alarmas se buscan en `$picasrc/alarms` en vez de a partir de `$picasrc`.

Con todo esto, tenemos que un ejemplo de declaración de alarma podría ser:

Ejemplo 10. Ejemplo de declaración de alarma

```
alarm alarmaPrueba {
   source   = prueba;
   priority = Urgent;
   vars {
      mivariable = mivalor;
      limite     = 80;
      servidor   = central;
      modo       = simple;
      rutabin    = /usr/local/bin;
   }
   uses {
      file alarmaPruebaConfig {
         source = alarms/config/prueba.conf;
 path   = <# $picaalarms #>/config/prueba.conf;
      }
   }
}
```

6.2. Componentes de PIFIA

Actualmente podemos decir que PIFIA se compone de:

- Un módulo de Perl con definiciones útiles para escribir alarmas, como persistencia en forma de extensión de la sintaxis.
- Tres programas para gestionar las alarmas de forma local (aunque están pensados para instalarse y ejecutarse remotamente con PICA).
- Un programa que ejecuta periódicamente todas las alarmas que haya instaladas en un momento dado.
- Un fichero para el cron, que ejecuta las alarmas en los intervalos apropiados.
- Un fichero para incluir en `objects.conf` que define todos los ficheros necesarios para ejecutar las alarmas. Estos ficheros deben copiarse a cualquier máquina donde queramos ejecutarlas.

7. El PPP (Perl PreProcessor)

Como dijimos al principio, todos los ficheros de configuración y de distribución de PICA se preprocesan antes de pasar por el analizador sintáctico. Este componente es uno de los puntos más importantes de PICA, porque es el que le da gran parte de su flexibilidad. Es fácil de usar y entender y da unas posibilidades a la hora de tratar los ficheros a distribuir que, al menos de forma tan sencilla, no hemos encontrado en ninguna otra herramienta.

El preprocesador es una función escrita en Perl que hace las sustituciones necesarias en el fichero, generando uno nuevo ya preprocesado. Es muy parecido al preprocesador de C, pero con algunas diferencias importantes:

- Hay dos primitivas para incluir programas en Perl, para generar dinámicamente partes de los ficheros.
- Las expresiones que ponemos en las condiciones y similares son Perl puro (que se `eval`úan), que pueden consultar el espacio de nombres de la aplicación para consultar ciertas variables del estado interno, como el fichero o la máquina actual que se están procesando, etc.

La lista completa de opciones que nos da el preprocesador es:

- Una directiva `#if/#elsif/#else/#fi` para condiciones simples. En estas condiciones podemos poner expresiones genéricas en Perl.
- Una directiva `#include` para incluir ficheros (que serán procesados recursivamente), como en el caso del C.
- Una directiva `#perl/#lrep` para generar dinámicamente partes del fichero a distribuir. Lo que se *devuelva* en el entorno `#perl/#lrep` quedará en el fichero final a copiar.
- Una directiva `<#/#>` que funciona igual que la anterior, pero es una versión reducida, más cómoda, para usar en una sola línea.

Veamos un ejemplo del preprocesador de PICA en acción, con uno de los ficheros de ejemplo que vienen en la distribución oficial:

Ejemplo 11. Ejemplo de fichero de distribución de PICA

```
Well, this is just a test file to see if it worked...

The value of the (local) variable myvar is '<# $myvar #>'
```

```
The value of the (local) variable mythirdvar is '<# $mythirdvar #>'


$myvar is <# ($myvar lt $mythirdvar)?'less':'greater' #> than
$mythirdvar...

The crypt'ed version of $mythirdvar is '<# crypt $mythirdvar,
'aa' #>'.


Now, a little list from 0 to <# $limit #>:
#perl
my @result;
for (my $i = 0; $i < $limit; ++$i) {
        push @result, "$i\n";
}
@result;
#lrep
```

Suponiendo una definición como la siguiente:

Ejemplo 12. Ejemplo de definición del fichero pica-rules en `objects.conf`

```
file pica-rules {
    path = '<#$picaroot#>/rules';   # Esto también se
                                          # preprocesará
    source = 'pica-rules';

    vars {
       myvar         = 'foo';
       limit         = '2';
       mythirdvar    = 'enough';
    }
}
```

el resultado del fichero, ya preprocesado, sería:

Ejemplo 13. Fichero preprocesado por PPP

Well, this is just a test file to see if it worked...

The value of the (local) variable myvar is ’foo’
The value of the (local) variable mythirdvar is ’enough’

$myvar is greater than $mythirdvar...

The crypt’ed version of $mythirdvar is ’aaVxUBNI9d3lI’.

Now, a little list from 0 to 2:
0
1

Esta versión es, naturalmente, la que se instalaría en las máquinas. Nótese además que el fichero de configuración de definición de objetos (`objects.conf`) y los ficheros de distribución vuelven a leerse y preprocesarse por cada máquina, así que podemos hacer que éstos dependan del objetivo donde se instalarán (ver ejemplo "Creación de varios ficheros a partir de un solo fuente" en el siguiente apartado).

8. Ejemplos de usos reales de PICA

En este apartado se dan ejemplos reales de usos que le hemos dado a PICA, que muestran diversas características del programa.

8.1. Distribución de claves RSA

Para no estar continuamente escribiendo claves a la hora de acceder a los servidores por SSH, hemos distribuido nuestras claves públicas en varios servidores y activado los agentes RSA. La propia distribución de las claves puede hacerse de forma automática con PICA. A continuación presentamos el extracto de `objects.conf` donde se definen los ficheros a distribuir, y su contenido.

Ejemplo 14. Distribución de claves RSA: extracto de `objects.conf`

```
group RSAAuth {
    # Fichero de autorización de SSHv2
    file ssh_auth {
```

```
          path = '/root/.ssh2/authorization';
          source = "SSH/authorization.cfg";
     }
     file sysadm1.pub {
          path = '/root/.ssh2/sysadm1.pub';
          source = "SSH/sysadm1.pub";
     }
     file sysadm2.pub {
          path = '/root/.ssh2/sysadm2.pub';
          source = "SSH/sysadm2.pub";
     }
     # ...
     # Resto de las definiciones de ficheros de clave pública
}
```

Por supuesto, podríamos haber generado dinámicamente las entradas para los ficheros de clave pública, pero no creemos que valga la pena, por seguridad y porque probablemente no serán muchas.

Ejemplo 15. Distribución de claves RSA: fichero `authorization.cfg`

```
#perl
my @return;
# Obtenemos los ficheros de clave leyendo los miembros del grupo
# SSHAuth, pero saltándonos el fichero 'ssh_auth'
my @keys=grep(/\.pub$/,members('SSHAuth'));
foreach my $key (@keys) {
  push @return,"Key $key\n";
}
# Devolvemos la lista (se imprimirá en la versión final del
# fichero)
@return;
#lrep
```

Después de preprocesarse, el fichero contendrá los nombres de los ficheros «lógicos» (es decir, lo que va después de `file`) que terminen en `.pub`, uno por línea, precedidos de `Key`, es decir, algo como esto:

Ejemplo 16. Distribución de claves RSA: ejemplo de fichero `authorization` creado automáticamente

```
Key sysadm1.pub
Key sysadm2.pub
...
```

8.2. Creación de varios ficheros a partir de un solo fuente

Otra de las posibilidades que nos brinca PICA es concentrar en un solo fuente todas las variaciones necesarias de un fichero, mediante condicionales. Por ejemplo, podemos querer instalar en la misma máquina dos ficheros iguales a partir del mismo fuente. Este caso se nos dio en los servidores primarios de nombres, en los que además queríamos guardar ficheros de ejemplo de los secundarios para publicarlos en web. Para ello, no nos hizo falta nada especial: pudimos aprovechar el mismo fuente para ambas versiones.

Ejemplo 17. Declaración de los servidores DNS

```
hostgroup dnsservers {
    members { fobos, deimos, mercurio, ulpnet, ulpnet2 }
}
hostgroup dnsmaster {
    members { ulpnet, ulpnet2 }
}
hostgroup doc {
    members { ulpnet, ulpnet2 }
}
```

Como vemos, tenemos declarados los grupos de máquinas que dan servicio DNS, los que son servidores principales, y los que hacen de servidor de documentación (que coinciden con los servidores principales de nombres). Los objetos se declararon de la siguiente manera:

Ejemplo 18. Declaración de los objetos a instalar

```
#if (ingroup('dnsservers'))
group DNS {
    file named.conf {
```

```
          path = '/etc/named.conf';
          source = "DNS/named_conf.cfg";
     }

     ## Documentación del servicio
#  if (ingroup('doc'))
     # ...
     file named.conf.sample {
        path = '<#$docdir#>/Servicios/DNS/named.conf.sample';
        source = 'DNS/named_conf.cfg';
     }
#  fi
}
#fi
```

De esta forma, sabemos que ambos ficheros se van a sacar del mismo fuente (`DNS/named_conf.cfg`), *y* que los ficheros de ejemplo siempre terminan en `.sample`. Con esta información tenemos suficiente para poder distinguir con el preprocesador. El contenido del fichero `named.conf.cfg` será algo como:

Ejemplo 19. Contenido parcial del fichero `named.conf.cfg`

```
# ...
zone "ulpgc.es" {
#if ((ingroup('dnsmaster')) && ($picaobject !~ /\.sample$/))
   type master;
   file "mydb.db";
   also-notify {
      # ...
   };
#else
   type slave;
   file "mydb.db.bak";
   masters {
      # ...
   };
#fi
};
# ...
```

Así, cada vez que se vaya a instalar cualquier fichero en los servidores de nombre primarios, se comprobará el nombre del fichero (el nombre del objeto en el fichero de configuración, no del fuente). Si termina en `.sample`, el contenido será el de un

servidor secundario de DNS; si no, será la configuración de un DNS primario. Si la máquina donde va a instalarse es un secundario, no hay posible ambigüedad, siempre se instala la versión de servidor secundario.

9. Conclusiones

PICA es una herramienta sencilla, muy fácil de aprender y sin grandes pretensiones. Consideramos que, excepto en instalaciones muy grandes, no vale la pena aprender complicados conceptos y nuevos lenguajes para resolver problemas; PICA resuelve problemas sin exigir que el administrador aprenda nada que no le sea familiar.

10. Licencia