Icinga

Sumario

Instalación de Icinga Core

Veremos a continuación los pasos de instalación y puesta en marcha de Icinga2 sobre Debian 9 (stretch)

Actualizamos caché de paquetes

```
apt update
```

Descarga de clave de repositorio y registro del mismo en la lista de repositorios del sistema

```
wget -O - https://packages.icinga.com/icinga.key | apt-key add -
echo 'deb https://packages.icinga.com/debian icinga-stretch main' >/etc/apt/sources.list.d/icinga.list
```

Instalación de Icinga Core

```
apt install -y apt-transport-https
apt update
apt install -y icinga2
```

Instalación de plugins

```
apt install -y monitoring-plugins
```

Instalación de Icingaweb

El proyecto dispone de una interesante interfaz web que permite visualizar la información de monitorización recopilada por Icinga. Vamos a realizar los pasos de instalación necesarios

Instalando icingaweb2 mariadb-server y módulos ido-mysql

```
apt install -y icingaweb2 icingacli mariadb-server icinga2-ido-mysql
```

Responder ?No? a las preguntas del asistente de instalación del módulo ido-mysql

Creación de bases de datos

Accedemos a la consola de administración del cliente de mysql

```
mysql -u root
```

Ejecutamos desde el cliente mysql las siguientes sentencias de creación de las bases de datos icingaweb2, para almacenar los usuarios y grupos de icinga, e icingaido en el que se almacenará la información de monitorización. En ambos casos concederemos permisos al usuario icinga de mysql

```
CREATE DATABASE icingaweb2;
GRANT ALL ON icingaweb2.* TO icinga@localhost IDENTIFIED by 'abc123.';
CREATE DATABASE icingaido;
GRANT ALL ON icingaido.* TO icinga@localhost IDENTIFIED by 'abc123.';
```

Una vez ejecutados los comandos de creación de la base de datos anteriores, salimos del cliente mysql con

```
quit
```

Vamos a incorporar el **schema IDO-Mysql** en la base de datos icingaido que acabamos de crear. Este schema IDO (Icinga Data Output) será utilizado por Icinga a la hora de escribir en Mysql los datos de monitorización recopilados

```
mysql -u root icingaido < /usr/share/icinga2-ido-mysql/schema/mysql.sql
```

Web Setup para Icingaweb

Accedemos a través del navegador web a icingaweb para la realización del setup

http://IP_ICINGA/icingaweb2/setup

Creación del token

Lo primero que nos pide es un setup token, el cual podremos generar con el comando

```
icingacli setup token create
```

Copiamos el token generado y lo pegamos en la caja de texto del primer paso de configuración del Web Setup.

Si en el siguiente paso aparece la alerta de que no está definido el **default.timezone** de php lo hacemos editando el archivo correspondiente

Configuración del date.timezone de php

```
vi /etc/php/7.0/apache2/php.ini
```

Descomentando y estableciendo la directiva

```
date.timezone = Europe/Madrid
```

Tras lo cual reiniciamos apache y ya podremos continuar

```
service apache2 restart
```

Usuario admin y password

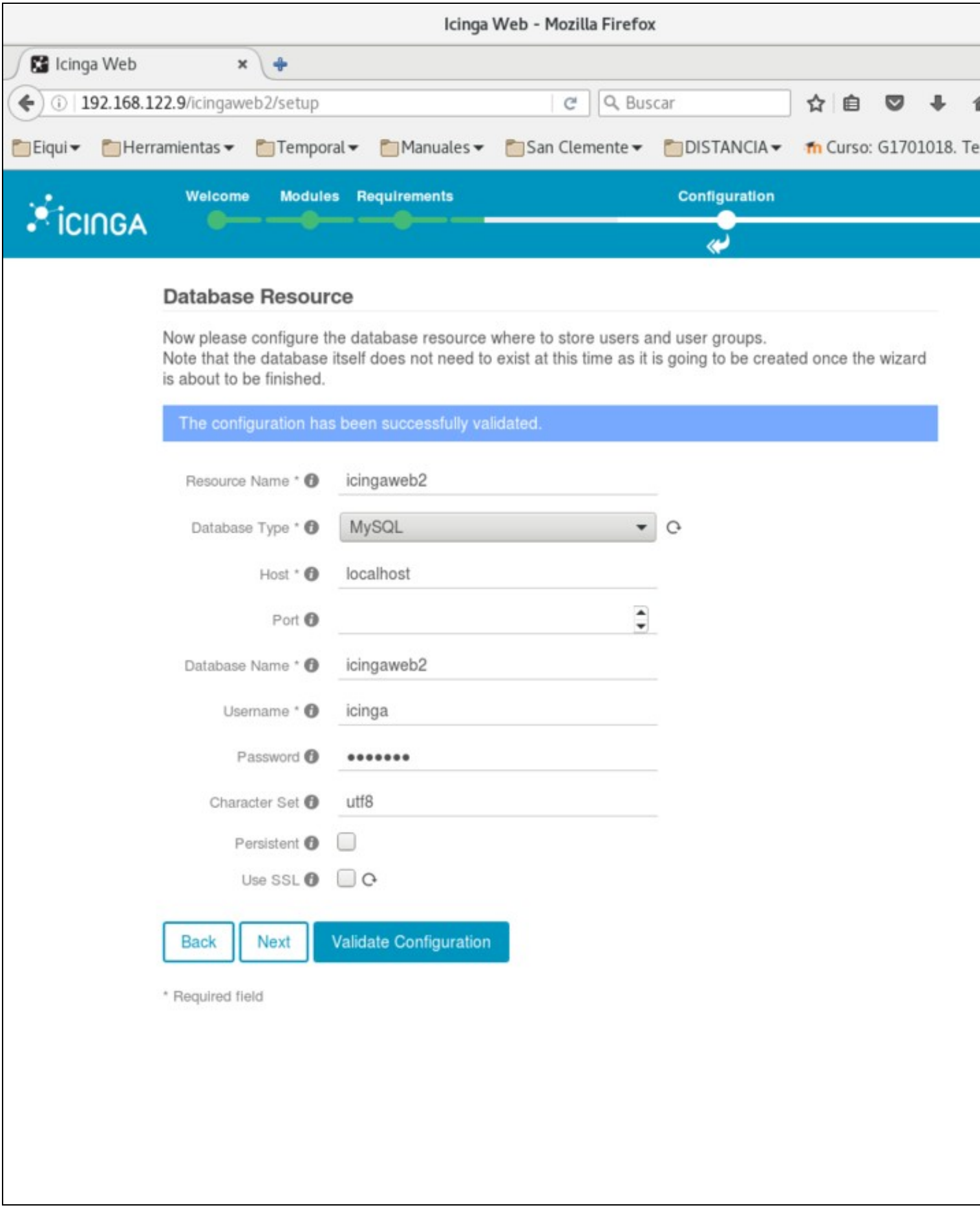
Establecemos el usuario **admin y su password**

user: admin **password**: abc123.

Definición de recurso de bases de datos

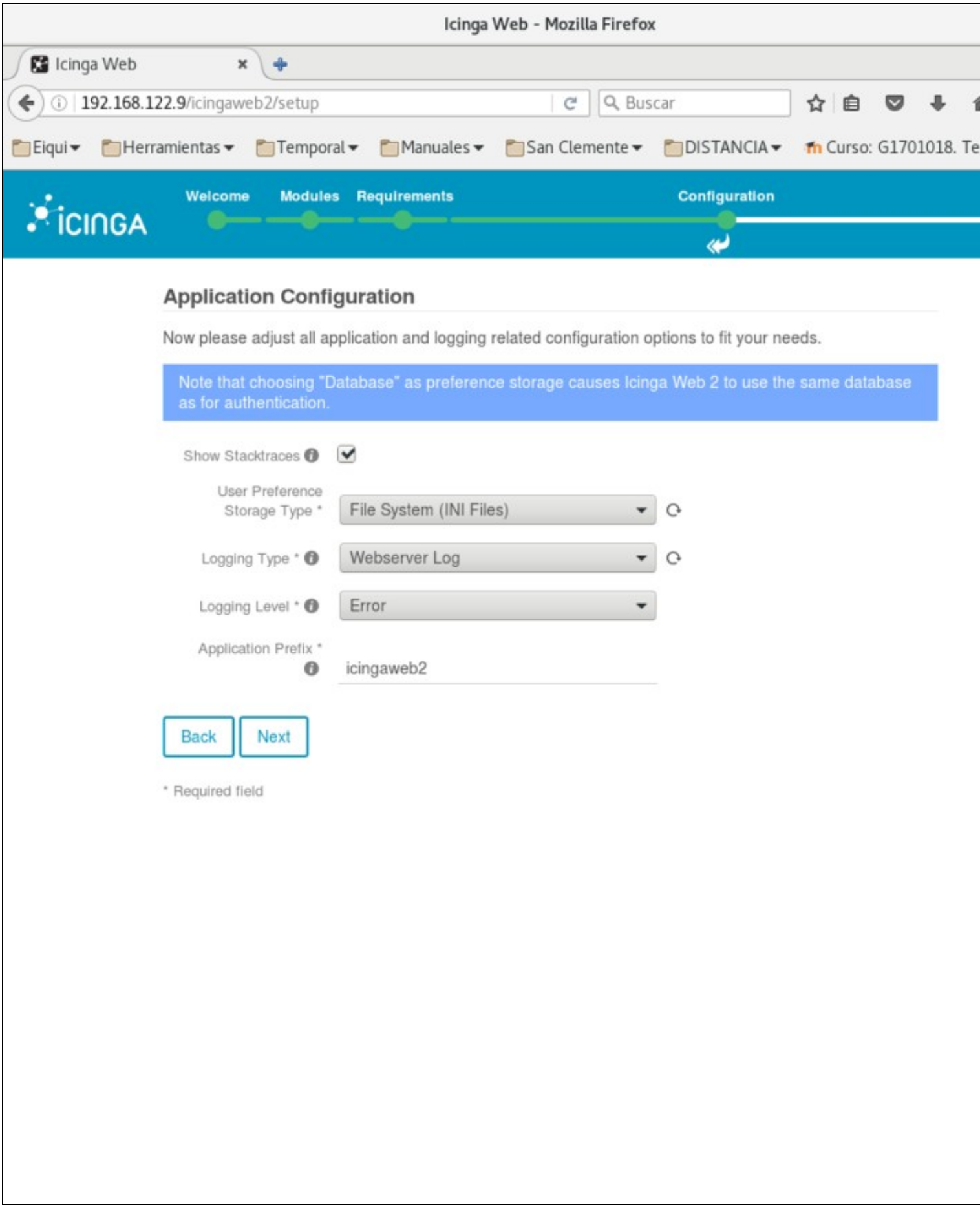
NOTA: A la hora de establecer credenciales para MySql, utilizar las creadas anteriormente y para la codificación **utf8**

El primer elemento a configurar es la base de datos en la que se almacenará la **configuración de usuarios y grupos** de icinga, introducimos las credenciales para la base de datos **icingaweb2**.


Icinga Web - Mozilla Firefox
Icinga Web
192.168.122.9/icingaweb2/setup
Buscar
Eiqui
Herramientas
Temporal
Manuales
San Clemente
DISTANCIA
Curso: G1701018. Te
Welcome
Modules
Requirements
Configuration
Database Resource
Now please configure the database resource where to store users and user groups.
Note that the database itself does not need to exist at this time as it is going to be created once the wizard is about to be finished.
The configuration has been successfully validated.
Resource Name *
icingaweb2
Database Type *
MySQL
Host *
localhost
Port
Database Name *
icingaweb2
Username *
icinga
Password
Character Set
utf8
Persistent
Use SSL
Back
Next
Validate Configuration
* Required field

Configuración de datos de aplicación

En la sección ?**Application Configuration**? seleccionamos la opción ?File System?


Icinga Web - Mozilla Firefox
Icinga Web
192.168.122.9/icingaweb2/setup
Buscar
Eiqui
Herramientas
Temporal
Manuales
San Clemente
DISTANCIA
Curso: G1701018. Te
Welcome
Modules
Requirements
Configuration
Application Configuration
Now please adjust all application and logging related configuration options to fit your needs.
Note that choosing "Database" as preference storage causes Icinga Web 2 to use the same database as for authentication.
Show Stacktraces
User Preference Storage Type *
File System (INI Files)
Logging Type *
Webserver Log
Logging Level *
Error
Application Prefix *
icingaweb2
Back
Next
* Required field

Recurso de base de datos de monitorización IDO

A continuación vamos a editar el archivo /etc/icinga2/features-available/ido-mysql.conf para habilitar el módulo IDO de MySql

```
vi /etc/icinga2/features-available/ido-mysql.conf
```

Establecemos los valores según lo indicado en la creación de la base de datos

```
object IdoMysqlConnection "ido-mysql" {
user = "icinga",
password = "abc123.",
host = "localhost",
database = "icingaido"
}
```

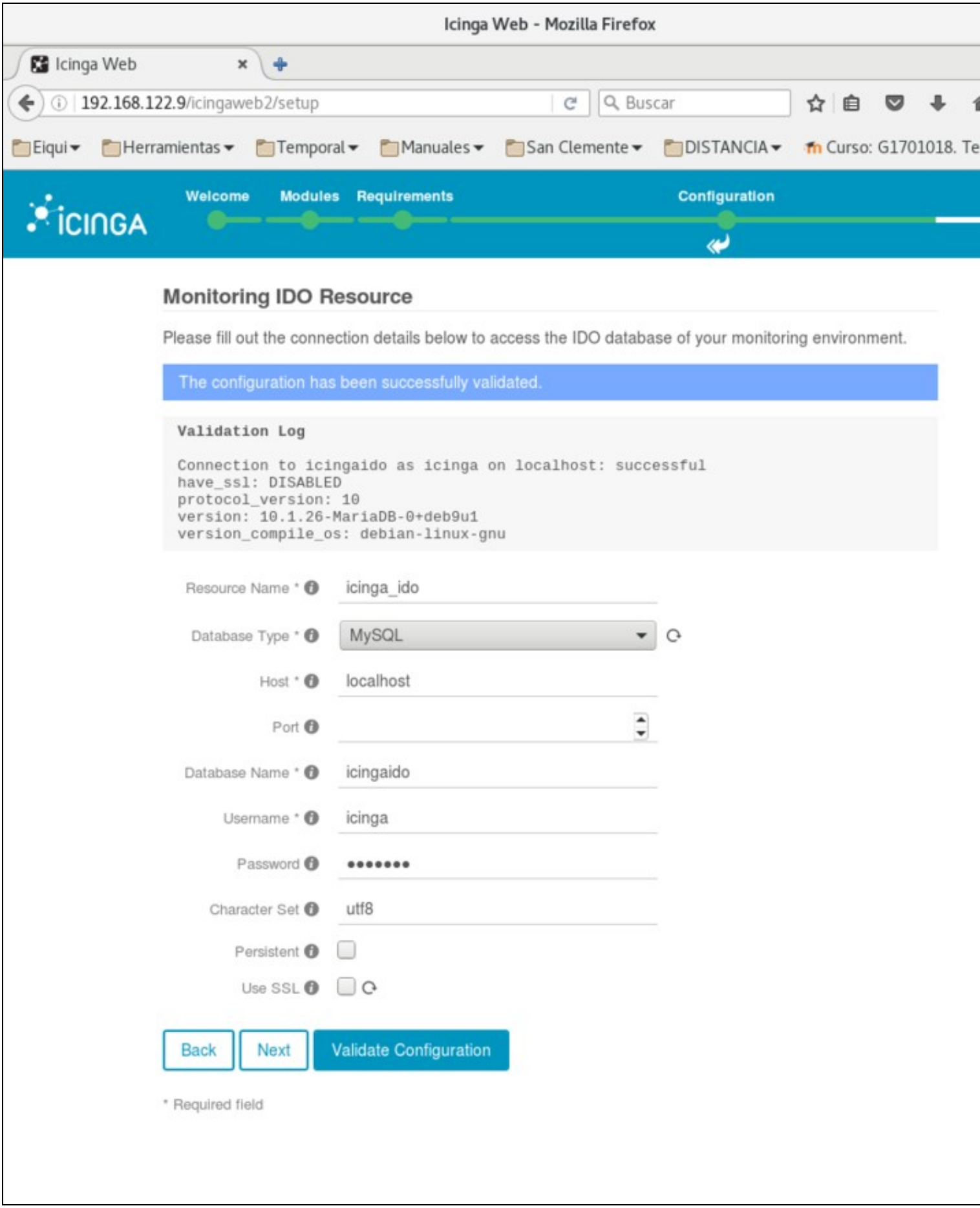
Ejecutamos el comando para **habilitar ido-mysql**

```
icinga2 feature enable ido-mysql
```

Reiniciamos el servicio de icinga

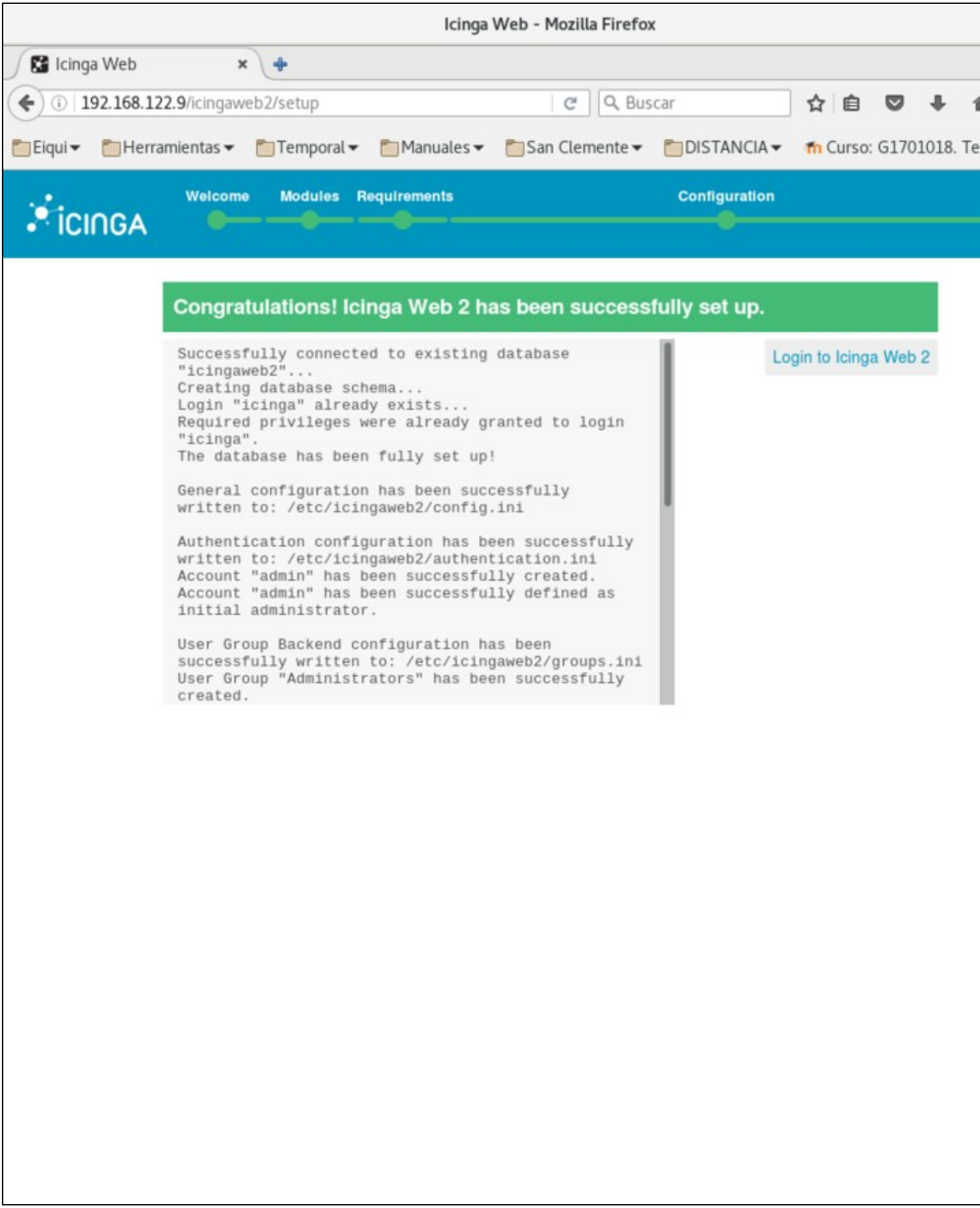
```
systemctl restart icinga2.service
```

Establecemos el Recurso de bases de datos para la información de monitorización, en este caso usaremos la base de datos **icingaido**, la cual incorpora el esquema ido-myql

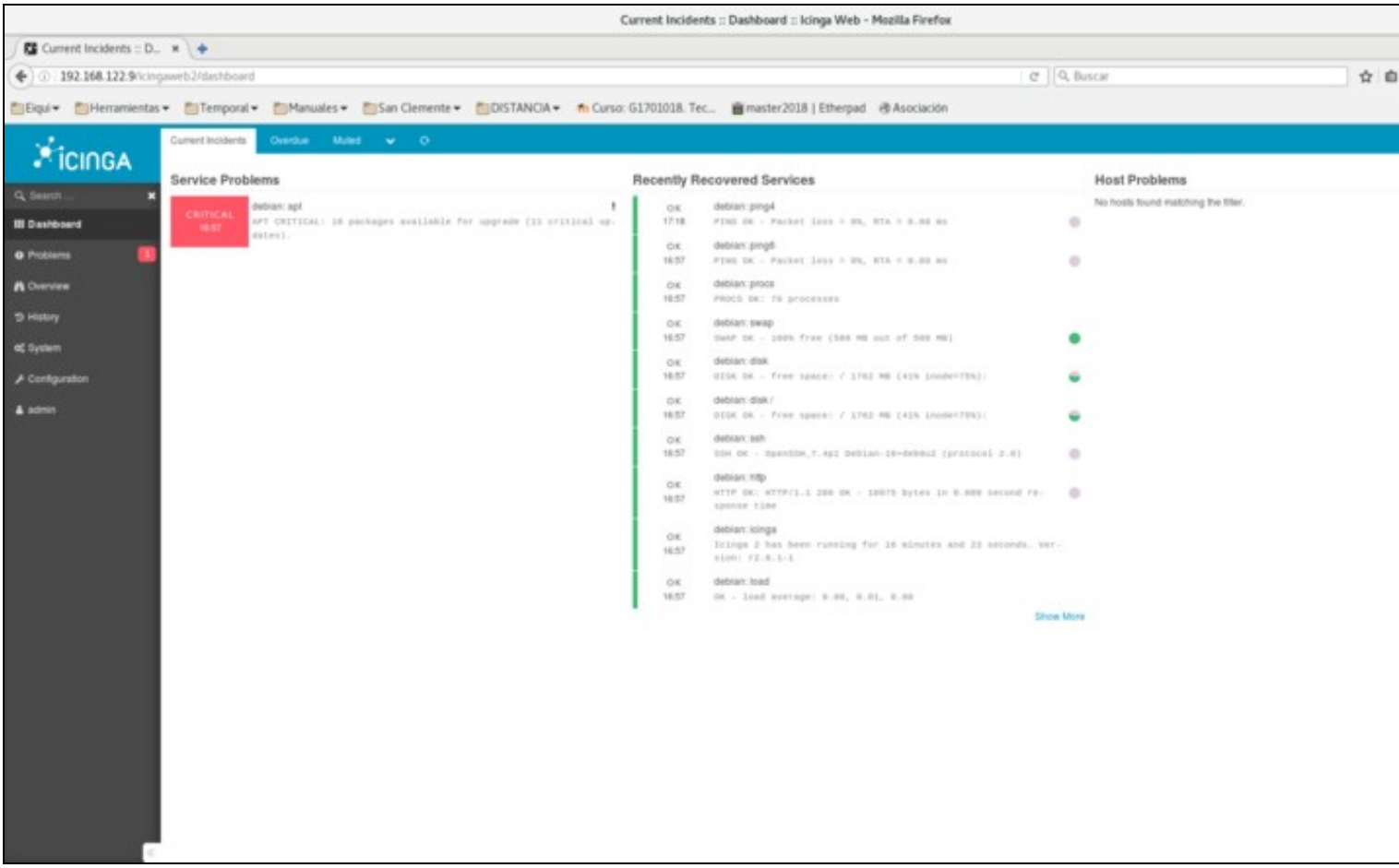

Icinga Web - Mozilla Firefox
Icinga Web
192.168.122.9/icingaweb2/setup
Buscar
Eiqui
Herramientas
Temporal
Manuales
San Clemente
DISTANCIA
Curso: G1701018. Te
Welcome
Modules
Requirements
Configuration
Monitoring IDO Resource
Please fill out the connection details below to access the IDO database of your monitoring environment.
The configuration has been successfully validated.
Validation Log
Connection to icingaido as icinga on localhost: successful
have_ssl: DISABLED
protocol_version: 10
version: 10.1.26-MariaDB-0+deb9u1
version_compile_os: debian-linux-gnu
Resource Name *
icinga_ido
Database Type *
MySQL
Host *
localhost
Port
Database Name *
icingaido
Username *
icinga
Password
Character Set
utf8
Persistent
Use SSL
Back
Next
Validate Configuration
* Required field

En la página **Command Transport** seleccionamos Transport Type ?**Local Command File**?

Tras lo cual pulsamos **Finish**. Si todo va bien deberíamos de ver la siguiente página


Icinga Web - Mozilla Firefox
Icinga Web
192.168.122.9/icingaweb2/setup
Buscar
Eiqui
Herramientas
Temporal
Manuales
San Clemente
DISTANCIA
Curso: G1701018. Te
Welcome
Modules
Requirements
Configuration
Congratulations! Icinga Web 2 has been successfully set up.
Successfully connected to existing database "icingaweb2"...
Creating database schema...
Login "icinga" already exists...
Required privileges were already granted to login "icinga".
The database has been fully set up!
General configuration has been successfully written to: /etc/icingaweb2/config.ini
Authentication configuration has been successfully written to: /etc/icingaweb2/authentication.ini
Account "admin" has been successfully created.
Account "admin" has been successfully defined as initial administrator.
User Group Backend configuration has been successfully written to: /etc/icingaweb2/groups.ini
User Group "Administrators" has been successfully created.
Login to Icinga Web 2

Una vez hacemos login en Icinga e introducimos el usuario y contraseña creados durante el proceso de setup (usuario: admin, password: abc123.), veremos el Dashboard de monitorización de IcingaWeb



Principios de monitorización de Icinga

El objetivo principal a la hora de diseñar Icinga fue la modularidad. El concepto de monitorización es muy genérico y dependiente del contexto, por tanto los sistemas que ofrecen soluciones al problema deben de ser muy flexibles, adaptables y abiertos en cuanto a requisitos y funcionalidad.

La ventaja del diseño modular de Icinga es que satisface los requisitos de flexibilidad y carácter adaptable y abierto. Por el contrario, nos encontramos con un sistema complejo y en algunos aspectos difícil de comprender. Aún así, para casos de uso no complicados, el tiempo de puesta en marcha y el esfuerzo necesario para el despliegue es bastante bajo.

Vamos a ver algunos de los elementos centrales que constituyen el núcleo del motor de monitorización de Icinga.

Host y Services

Un **host** es un equipo que alberga servicios que pueden ser monitorizados. Entre otros atributos se definen comunmente su nombre y dirección o hostname

Un **service** es el elemento fundamental a monitorizar, y que por lo general corresponde con algún servicio en ejecución.

Veamos algún ejemplo:

```
object Host "my-server1" {
  address = "10.0.0.1"
  check_command = "hostalive"
}

object Service "ping4" {
  host_name = "my-server1"
  check_command = "ping4"
}
```

```
object Service "http" {
  host_name = "my-server1"
  check_command = "http"
}
```

Vemos en los ejemplos anteriores:

- Un elemento **Host,** dentro del cual se definen los atributos:
  - **address**: la dirección IP o hostname del host
  - **check_command**: identifica el comando de chequeo a efectuar en relación al host, en este caso un hostalive corresponde con un ping
- Dos elementos **Service**, en los que se definen
  - **host_name**: el nombre del host asociado al service, es decir, el nombre del host que ?corre? el service, en este caso el definido anteriormente
  - **check_command**: comando de chequeo para monitorizar el servicio

Los comandos de chequeo forman parte de los plugins de Icinga, y constituyen el elemento funcional más importante del sistema de monitorización. En el ejemplo anterior encontramos dos ejemplos:

- **http**: comando del plugin check_http
- **ping4**: comando perteneciente al plugin check_ping que efectúa un ping IPv4

NOTA

Para determinar las ubicaciones de los plugins podemos visualizar el contenido del archivo **/etc/icinga2/constants.conf** donde se definen, entre otras, variables de localización de los directorios de plugins, por defecto estos se ubican en /usr/lib/nagios/plugins

Host Groups y Service Groups

Es posible agrupar conjuntos de hosts mediante la definición de un elemento **HostGroup**. Del mismo modo pueden agruparse Servicios utilizando elementos de tipo **ServiceGroup**. Veamos algún ejemplo:

extracto del archivo /etc/icinga2/conf.d/groups.conf

```
object HostGroup "linux-servers" {

display_name = "Linux Servers"

assign where host.vars.os == "Linux"

}
```

La sintaxis es clara, se define un **HostGroup** y se agruparán los hosts utilizando como criterio lo establecido por la directiva **assign**, que en este caso agrupa todos los Host que tengan establecida la variable **vars.os** al valor ?Linux?.

De modo análogo podríamos establecer un HostGroup para los windows-servers

```
object HostGroup "windows-servers" {

display_name = "Windows Servers"

assign where host.vars.os == "Windows"

}
```

A continuación vemos la definición de un objeto **ServiceGroup**

```
object ServiceGroup "ping" {

display_name = "Ping Checks"

assign where match("ping*", service.name)

}
```

Este ejemplo es similar, con la salvedad de que se está definiendo un elemento de tipo ServiceGroup, cuya finalidad es agrupar Services que tengan algo en común; en este caso según la directiva **assign**, se establece que todos los Services que tengan en el valor del atributo **service.name** el texto ?ping? serán agrupados bajo el ServiceGroup ?Ping Checks?

Otros ejemplos

```
object HostGroup "web-servers" {

display_name = "Web Servers"

assign where host.vars.webserver == true

}
```

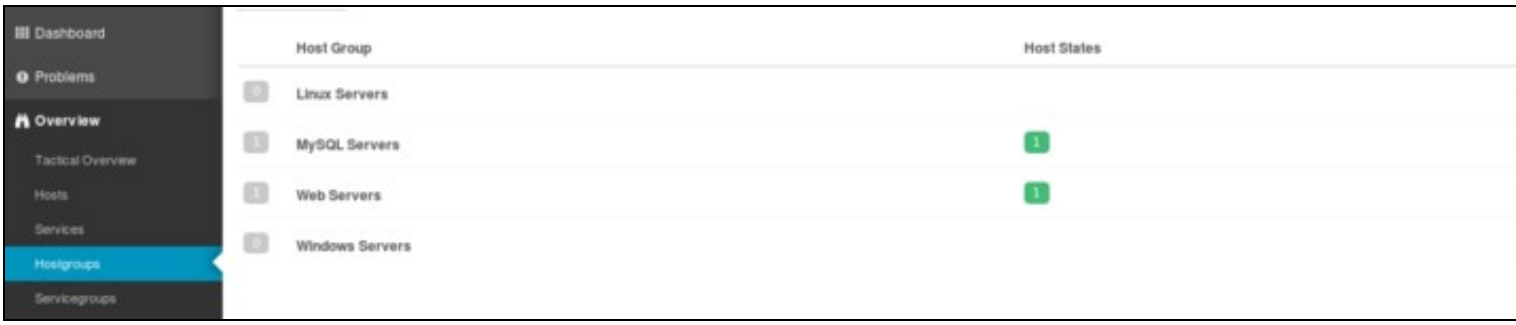
De nuevo resulta claro el tipo de agrupamiento, en este caso todos aquellos Host que tengan establecido a true al atributo **vars.webserver**

Por último definimos un **HostGroup** para agrupar los Host que corran un servicio MySQL, estos erán identificados mediante el atributo de Host **vars.mysql** establecido a true

```
object HostGroup "mysql-servers" {

display_name = "MySQL Servers"

assign where host.vars.mysql == true

}
```

Vemos en el panel IcingaWeb los correspondientes HostGroup y ServiceGroup

HostGroups



ServiceGroups

Commands

Los **command** son los elementos que definen como se realizan los chequeos de monitorización. Existe una directiva definida en los objetos de tipo service, check_command, que define el tipo de chequeo a realizar.

Hay los siguientes tipos

Check Command

Definen como se efectúan los comandos de chequeo asociados a la monitorización. Básicamente los objetos CheckCommand se definen para pasar parámetros de ejecución a los plugins de monitorización. Veamos un ejemplo

```
object CheckCommand "my-mysql" {
  command = [ PluginDir + "/check_mysql" ] //constants.conf -> const PluginDir

  arguments = {
    "-H" = "$mysql_host$"
    "-u" = {
      required = true
      value = "$mysql_user$"
    }
    "-p" = "$mysql_password$"
    "-P" = "$mysql_port$"
    "-s" = "$mysql_socket$"
    "-a" = "$mysql_cert$"
    "-d" = "$mysql_database$"
    "-k" = "$mysql_key$"
    "-C" = "$mysql_ca_cert$"
    "-D" = "$mysql_ca_dir$"
    "-L" = "$mysql_ciphers$"
    "-f" = "$mysql_optfile$"
    "-g" = "$mysql_group$"
    "-S" = {
      set_if = "$mysql_check_slave$"
      description = "Check if the slave thread is running properly."
    }
    "-l" = {
      set_if = "$mysql_ssl$"
      description = "Use ssl encryption"
    }
  }

  vars.mysql_check_slave = false
  vars.mysql_ssl = false
  vars.mysql_host = "$address$"
}
```

En la definición del CheckCommand anterior podemos ver tres tipos de atributos:

- **command**: indica el plugin invocado por el CheckCommand, utiliza la ruta absoluta establecida a partir del parámetro que define en constants.conf la ruta al directorio de plugins
- **arguments**: definen los argumentos pasados al plugin, se construyen a partir de los atributos (variables) definidas en los distintos contextos.
- **custom attributes**: atributos definidos de forma personalizada. Se utiliza con este propósito el attribute **vars**, donde se pueden almacenar de modo arbitrario valores de información

Notification Command

Indican como se gestionan las notificaciones asociadas a eventos de notificación, por ejemplo envío de mails, mensaje IRC, etc.

Templates

Los **templates** son definiciones utilizadas para englobar parámetros comunes que se pueden aplicar a otros elementos de forma conjunta. De este modo evitamos el definir varias veces los mismos parámetros. Veamos algún ejemplo:

extracto del archivo: /etc/icinga2/conf.d/templates.conf:

```
template Service "generic-service" {

max_check_attempts = 5

check_interval = 1m

retry_interval = 30s

}
```

El template anterior define una plantilla para servicios, estableciendo atributos por defecto compartidos por todos los objetos Service que hagan un import del template. Básicamente establece el tiempo entre chequeos, 1 minuto, el máximo número de reintentos en caso de fallo, 5, y el período de reintento ante fallo, 30 segundos.

A continuación vemos la definición de un Service que hace uso del template anterior

extracto del archivo: /etc/icinga2/conf.d/services.conf:

```
apply Service for (disk => config in host.vars.disks) {

import "generic-service"

check_command = "disk"

vars += config

}
```

En la definición del cuerpo del **Service** vemos la cláusula **import** que hace referencia a **generic-service**, mostrado anteriormente, por tanto todos los atributos de generic-service serán de aplicación en el elemento **apply.** Más adelante volveremos sobre este ejemplo.

Macros y Attributes

Los **atributtes** se asocian a la definición de los hosts, services, commands, etc., y permiten definir los objetos mediante la especificación de su información esencial. Existen varios scopes o ámbitos de atributos y también la posibilidad de definir **custom attributes,** característica muy útil para personalizar el comportamiento de los objetos de Icinga. Por tanto los objetos Icinga serán definidos a partir de sus elementos attribute.

Por ejemplo:

```
object Host "localhost" {
  check_command = "ssh"
  vars.ssh_port = 2222
}
```

El extracto anterior muestra la definición de un objeto Host asociado al cual se definen dos attributes el atributo **check_command** que establece el comando de chequeo asociado al objeto, en este caso ?ssh? indica que se verificará la disponibilidad del servicio ssh

- el **custom attribute ssh_port** definido dentro de la variable vars, la cual almacena valores personalizados asociados al objeto, en este caso en el ámbito del objeto Host. El funcionamiento sería el siguiente:

Al ejecutar el check_command asociado al objeto es necesario pasar una serie de parámetros al mismo. En este caso el comando ssh necesitaría información para el chequeo, básicamente:

- la dirección IP o hostname, **address**, del host: como no se especifica se toma por defecto la dirección o name del propio Host, en este caso ?localhost?
- El **puerto**: si no se especifica se tomaría por defecto el 22, sin embargo, en este caso hemos definido un atributo, ssh_port, dentro de vars, lo cual permite establecer un custom attribute. Este valor así definido sobreescribiría el valor por defecto, 22, del atributo ssh_port definido en el propio command, de este modo se efectuará el test contra localhost en el puerto 2222.

Los elementos **macro** permiten recuperar información en tiempo real sobre los elementos a monitorizar. Algunos permiten recuperar simplemente variables con información, otros permiten definir comportamientos dinámicos mediante la definición de funciones. Constituyen por tanto elementos de información que se pueden consultar y que serán buscados en los distintos elementos, recuperando la coincidencia más cercana. Es importante

consultar la documentación de los plugins para conocer los parámetros y valores por defecto de estos en los command.

Por ejemplo:

```
vars.ping_address = "$address$"
```

hace uso de la macro $address$ que en este caso será localizada a partir del atributo address del objeto host. Concretamente la línea anterior asigna un custom attribute, dentro de la variable vars, el atributo ping_address, al que se asigna el valor del atributo address del host.

El orden de evaluación, la primera coincidencia, a la hora de buscar los atributos a los que se refieren las macros son:

- **User** object (solo para notificaciones)
- **Service** object
- **Host** object
- **Command** object
- **Global custom attributes** definidos en **vars**

Apply Rules

Las **apply rules** dotan de una gran potencia a Icinga, al permitir especificar comportamientos de los objetos en función de la verificación de reglas. De este modo se puede realizar una aplicación condicional e iterada sobre elementos del entorno. Simplifican enormemente el trabajo de definición de las políticas de monitorización.

Veamos un ejemplo:

extracto del archivo: /etc/icinga2/conf.d/services.conf:

```
apply Service "ping4" {
  import "generic-service"
  check_command = "ping4"
  assign where host.address
}
```

la cual define una regla que aplica un comando, definido en la directiva **check_command** a los elementos de tipo **Service** que verifiquen la condición establecida en la directiva **assign.** Además se hace uso del template **generic-service**, cuyos atributos se incluyen mediante la directiva **import.** La definición del template generic-service se encuentra en el archivo /etc/icinga2/conf.d templates.conf.

Volvamos a un ejemplo anterior:

```
apply Service for (disk => config in host.vars.disks) {
import "generic-service"
check_command = "disk"
vars += config
}
```

En este caso usamos la directiva **apply** para definir la aplicación de un Service en función de la colección definida en el atributo de **host vars.disks**, el cual podrá ser definido en la propia definición del host. En la definición del cuerpo del **Service** vemos la cláusula import que hace referencia a **generic-service**, mostrado anteriormente, por tanto todos los atributos de generic-service serán de aplicación en el elemento **apply**. El apply funciona como un bucle for que itera por todos los elementos del array vars.disks del host.

El elemento anterior aplica un chequeo para cada elemento iterado, recuperando los atributos de cada elemento disk del array vars.disks, en el elemento **config**, y añadiéndolos a los custom attributes, en vars, de cada Service de chequeo de cada disco.

En resumen, el apply anterior sería algo como ?para cada objeto disco definido el la variable vars.disks del host, un array, se realizará un chequeo basado en el comando disk, con los parámetros de chequeo definidos a partir de los attributes de cada elemento disk?.

Plugins

Los **plugins** son los elementos que definen las herramientas de chequeo, invocadas mediante elementos command, utilizadas a la hora de establecer las políticas de monitorización. El objetivo es que se ejecuten en un entorno en el que sea posible y ágil el definir nuevos elementos de este tipo, así como facilitar una integración sencilla con los elementos de monitorización.

Cada plugin ofrece una funcionalidad accesible a través de los command asociados. Es posible definir objetos CheckCommand que indiquen la

parametrización pasada al command, para ello es importante leer la documentación del plugin, sus comandos asociados, y los parámetros soportados

En la lista de Referencias de más abajo encontramos el listado de plugins estándar de Icinga, en esta lista se incluyen plugins para monitorización de:

- Sistemas Windows
- Sistemas Linux
- Sistemas de Gestión de Bases de Datos
- Sistemas de Servicio Web
- Sistemas de Red
- y un largo etc.

Entorno de Icinga

Al instalar Icinga se crean varios directorios en los que se almacenan los elementos necesarios para el construir el entorno de monitorización:

Directorio /etc/icinga2

En este directorio nos encontramos:

- **icinga2.conf**: archivo de configuración principal de Icinga. Básicamente consta de una serie de directivas ?include? que incorporan al entorno los elementos del ecosistema de monitorización
- **constants.conf**: archivo donde se definen constantes de entorno. Principalmente se definen constantes relacionadas con las rutas de los directorios en los que nos encontramos los plugins de monitorización. Los plugins son el soporte para los commands que definen las acciones de chequeo o monitorización.
- **conf.d**: Directorio en el que se ubican los archivos con las definiciones de los elementos vistos en el punto anterior: hosts, services, templates, etc.
- **scripts**: Directorio en el que encontramos scripts de shell invocados desde objetos Command de tipo Notification

Directorio /usr/lib/nagios/plugins

Directorio por defecto de los plugins de Icinga (basado en Nagios). En el nos encontramos ejecutables, binarios o scripts, que implementan una funcionalidad concreta de monitorización. Por lo general tienen nombres del tipo check_xxx autoexplicativos.

Ejemplo. Monitorización del Host Icinga

El propio Host en el que corre Icinga está sujeto a la monitorización establecida por defecto. Veamos la definición del propio Host en el archivo **/etc/icinga2/hosts.conf:**

```
object Host NodeName {
  /* Import the default host template defined in `templates.conf`. */
  import "generic-host"

  /* Specify the address attributes for checks e.g. `ssh` or `http`. */
  address = "127.0.0.1"
  address6 = "::1"

  /* Set custom attribute `os` for hostgroup assignment in `groups.conf`. */
  vars.os = "Docker"

  /* Define http vhost attributes for service apply rules in `services.conf`. */
  vars.http_vhosts["http"] = {
    http_uri = "/"
  }
  /* Uncomment if you've sucessfully installed Icinga Web 2. */
  //vars.http_vhosts["Icinga Web 2"] = {
  //  http_uri = "/icingaweb2"
  //}

  /* Define disks and attributes for service apply rules in `services.conf`. */
  vars.disks["disk"] = {
    /* No parameters. */
  }
  vars.disks["disk /"] = {
    disk_partitions = "/"
```

```
  }

  /* Define notification mail attributes for notification apply rules in `notifications.conf`. */
  vars.notification["mail"] = {
    /* The UserGroup `icingaadmins` is defined in `users.conf`. */
    groups = [ "icingaadmins" ]
  }
}
```

Se define el elemento Host de nombre "NodeName", el primer elemento en el cuerpo de la definición es la directiva **import**, que establece que se tomen los atributros del template **generic-host**, el cual establece básicamente la periodicidad de los chequeos.

Puede apreciarse a continuación como se definen atributos:

Algunos sobreescriben atributos específicos del objeto Hosts:

- **address**
- **address6**

Otros son atributos específicos (custom attributes) que definen información arbitraria que será utilizada por el sistema de plugins de monitorización. Estos valores se almacenan en vars, en el ejemplo anterior tenemos:

- **vars.os**: identifica el sistema que corre el Host
- **vars.http_vhosts**: incorporaría información relevante para monitorización de servicios http de ser el caso
- **vars.disks**: almacenaría información relativa a los dispositivos de almacenamiento y los puntos de montaje a monitorizar para el Host
- **vars.notification**: este mapa asociativo permitiría definir información relacionada con la notificación de eventos de monitorización de Icinga

A continuación inspeccionamos el archivo /**etc/icinga2/services.conf**, para ver como se hace uso de la información definida en los atributos del ejemplo anterior

```
apply Service "ping4" {
  import "generic-service"

  check_command = "ping4"

  assign where host.address
}

apply Service "ping6" {
  import "generic-service"

  check_command = "ping6"

  assign where host.address6
}

apply Service "ssh" {
  import "generic-service"

  check_command = "ssh"

  assign where (host.address || host.address6) && host.vars.os == "Linux"
}

apply Service for (http_vhost => config in host.vars.http_vhosts) {
  import "generic-service"

  check_command = "http"

  vars += config
}

apply Service for (disk => config in host.vars.disks) {
  import "generic-service"

  check_command = "disk"

  vars += config
}
```

```
apply Service "icinga" {
  import "generic-service"

  check_command = "icinga"

  assign where host.name == NodeName
}

apply Service "load" {
  import "generic-service"

  check_command = "load"

  /* Used by the ScheduledDowntime apply rule in `downtimes.conf`. */
  vars.backup_downtime = "02:00-03:00"

  assign where host.name == NodeName
}

apply Service "procs" {
  import "generic-service"

  check_command = "procs"

  assign where host.name == NodeName
}

apply Service "swap" {
  import "generic-service"

  check_command = "swap"

  assign where host.name == NodeName
}

apply Service "users" {
  import "generic-service"

  check_command = "users"

  assign where host.name == NodeName
}
```

Podemos observar como se definen los Service a partir de los elementos **apply** que aplican condiciones en función de la directiva **assign**. Cada Service definido se aplicaría solo en caso de que la directiva assign arrojase el valor verdadero.

Debemos observar como se hace uso extensivo de los atributos del Host, referenciados mediante la sintaxis host.<atributo>, y que establecen las condiciones mencionadas arriba.

Por ejemplo, el elemento **apply Service ssh** solo será aplicado cuando se haya asignado una dirección, address, IPv4 o IPv6, al Host, y éste tenga definida la variable host.vars.os al valor "Linux". A todo Host definido según esas premisas se le aplicará el Service ssh.

Tenemos otros elementos apply más complejos, como el correspondiente a la monitorización de los dispositivos de almacenamiento:

```
apply Service for (disk => config in host.vars.disks) {
  import "generic-service"

  check_command = "disk"

  vars += config
}
```

El cual en su definición recorre, mediante un for, los elementos definidos en el array **host.vars.disk**, procesando cada uno de los elementos y ejecutando el checkcommand correspondiente, en este caso un comando de chequeo disk, que verifica el estado del dispositivo. Por tanto, para todo elemento del Host definido en ese array se ejecutará el check.

Por último, vemos también el elemento

```
apply Service "swap" {
```

```
  import "generic-service"

  check_command = "swap"

  assign where host.name == NodeName
}
```

Que ejecuta un check del espacio de intercambio, swap, del Host, pero en este caso éste solo se aplica en caso de que el **host.name** sea "NodeName", es decir, el Host de monitorización.

Caso práctico 1. Monitorización simple de servicios

Vamos a ver un sencillo caso de uso en el que monitorizaremos dos servicios:

- Un servicio **MySQL** corriendo en otro host con IP **192.168.0.10**
- Un servicio **Nginx** corriendo en un host con IP **192.168.0.11**

Necesitaremos por tanto:

- Un host corriendo **Icinga** Core y la interfaz Icinga Web
- Un host corriendo un servicio **mysql** en el puerto estándar 3306 con IP **192.168.0.10** accesible en la misma subred que el nodo Icinga
- Un host corriendo un servicio **nginx** en el puerto estándar 80 con IP **192.168.0.11** accesible en la misma subred que el nodo Icinga

Una vez tengamos los 3 sistemas anteriores en funcionamiento procedemos a crear los elementos para la monitorización.

Para ello, vamos a definir:

- Dos objetos de tipo Host que representen los sistemas en los que corren los servicios
- Dos objetos de tipo Service que representan los servicios en sí

La implementación pasará por crear el archivo **mis_servicios.conf** dentro del directorio /**etc/icinga2/conf.d**

Dentro de ese archivo tendremos el siguiente contenido

```
object Host "mysqlmi" {

import "generic-host"

address = "192.168.0.10"

vars.mysql = true

}
```

Este primer elemento define el Host del servicio mysql, los tributos definidos en esta sección incluyen:

- El **nombre** del objeto Host, en este caso mysqlmi
- La directiva **import** hace uso del template generic-host, en el que se definen los atributos de un Host genérico, como ya hemos visto más arriba
- El atributo **address** define la dirección IP, en este caso 192.168.0.10, del Host
- Por último definimos un **atributo personalizado** (custom attribute), los cuales como vimos siempre se definen dentro de vars, que será utilizado a la hora de aplicar el correspondiente Servicio de monitorización a los Host etiquetados como ?servicios mysql?

```
apply Service "mysql-test" {

import "generic-service"

display_name = "MySQL Test"

check_command = "mysql"

vars.mysql_database = "mysql"

vars.mysql_username = "root"
```

```
vars.mysql_password = "abc123."

assign where host.vars.mysql == true

}
```

El elemento apply anterior define la aplicación del Service definido en función del atributo **host.vars.mysql**, es decir, los Hosts etiquetados como tal serán tenidos en cuenta como servicios de ese tipo a los que se aplican las directivas de monitorización indicadas, entre las cuales tenemos:

- La directiva **import** que hace uso del template generic-service
- El **display_name**, que indica el nombre del servicio de monitorización de mysql
- El **check_command**, que indican el comando de monitorización utilizado para supervisar el servicio mysql, en este caso el comando mysql, definido en el plugin check_mysql
- 3 **atributos personalizados**, dentro de **vars**, que serán utilizados como parámetros por el command anterior
- Por último la directiva **assign** establece que se aplicará la monitorización en función del valor del atributo **host.vars.mysql,** el cual como vimos se estableció en la definición del Host, por tanto solo se aplicará a los Hosts etiquetados con tal atributo

De forma análoga a lo anterior definimos el objeto Host y el apply Service para un servicio web, en este caso para un servidor nginx:

```
object Host "nginxmi" {

import "generic-host"

address = "192.168.0.11"

vars.webserver = true

}
```

En este caso el atributo **vars.webserver**, etiqueta el Host para vincularlo con el apply Service que sigue:

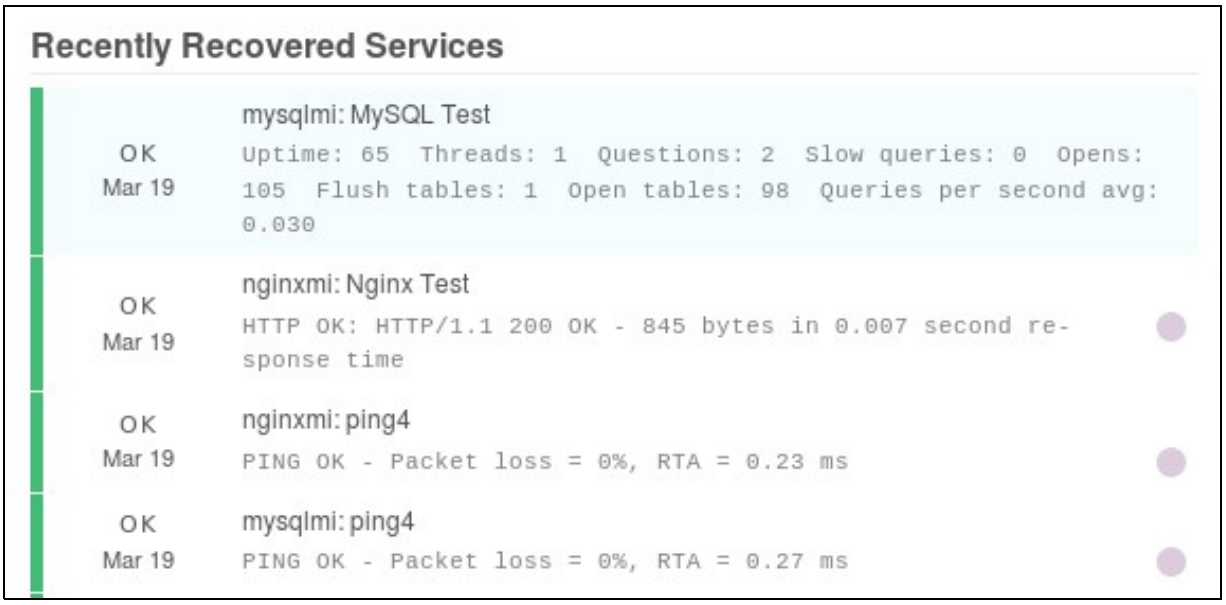
```
apply Service "nginx-test" {

import "generic-service"

display_name = "Nginx Test"

check_command = "http"

vars.http_port = "80"

vars.http_timeout = "10"

assign where host.vars.webserver == true

}
```

La estructura del elemento anterior es idéntica a la del apply Service para mysql, simplemente cambian el nombre, el check_command, en este caso un comando http del plugin check_http, los custom attributes pasados como parámetros al comando, y la directiva assign que aplica la condición en la cual se aplicará la monitorización del servicio.

Tras añadir al archivo /etc/icinga2/conf.d/mis_servicios.conf los contenidos anteriores reiniciamos el servicio

```
service icinga2 restart
```

Entramos al interfaz web de Icinga y vemos los nuevos servicios a monitorizar



Detalles del test Service mysql_test

UP
since Mar 19

mysqlmi

OK
since Mar 19

Service: MySQL Test (mysql-test)

Check now | Comment | Notification | Downtime

Plugin Output

```
Uptime: 122  Threads: 1  Questions: 5  Slow queries: 0  Opens: 106  Flush tables: 1  Open tables: 9
```

Problem handling

Comments — Add comment

Downtimes — Schedule downtime

Performance data

Label	Value
Connections	4.00
Open_files	14.00
Open_tables	100.00
Qcache_free_memory	1,031,832.00
Qcache_hits	0.00
Qcache_inserts	0.00
Qcache_lowmem_prunes	0.00
Qcache_not_cached	0.00
Qcache_queries_in_cache	0.00
Queries	6.00
Questions	3.00
Table_locks_waited	0.00
Threads_connected	1.00
Threads_running	1.00
Uptime	122.00

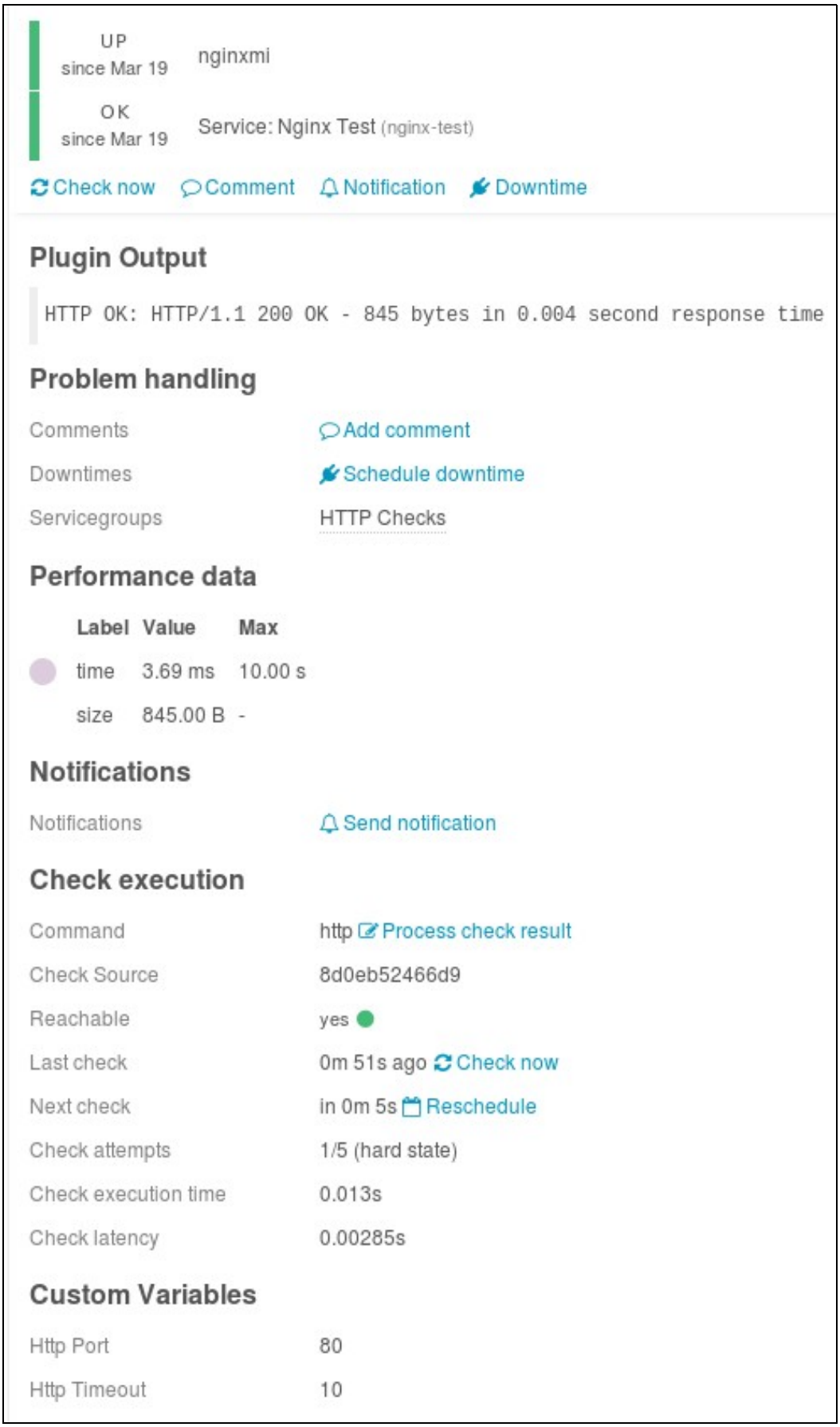
Notifications

Notifications — Send notification

Check execution

Command — mysql Process check result

Detalles del test Service nginx_test



Caso práctico 2. Monitorización de Sistema Windows

Vamos a describir brevemente como podría incorporarse un host Windows a la infraestructura de monitorización de Icinga.

En este Caso práctico veremos otro modo de afrontar la monitorización con Icinga. El nodo **cliente** a monitorizar, un host Windows, integrará las herramientas de monitorización de Icinga y se comunicará con el servidor, nodo **master**, al que le enviará la información de monitorización recopilada. Este intercambio ser realizará de modo seguro, haciendo uso del protocolo **SSL**. Este escenario supone de hecho un caso de **monitorización distribuida**.

En un escenario de este tipo se define una estructura jerárquica, en la que encontramos nodos

- **master**: En la raíz de la jerarquía
- **satellite**: Que dependen de un master o de otros satellite
- **client**: Actúan como agentes conectados a un nodo master o satellite

Infraestructuras de este tipo ayudan a organizar y desplegar sistemas de monitorización arbitrariamente complejos en alta disponibilidad y fragmentando las cargas. El master podría distribuir las cargas de monitorización entre los satellite.

Los nodos pueden ser organizados en elementos **zone**, las cuales se relacionan entre sí a través de relaciones de confianza "padre-hijo".

Por último tenemos los elementos **endpoint**, que serán miembros de un elemento zone. Los miembros de una misma zona actuarán en alta disponiblidad, de modo que nodos endpoint en la misma zona balancearán las cargas de check.

Aún cuando usaremos este ejemplo para ilustrarlo la monitorización distribuida puede aplicarse a hosts Windows y Linux.

Necesitaremos:

- Equipo corriendo **Windows 10** en la misma red que el servidor Icinga
- Instalación de el **paquete Icinga en Windows**
- **Intercambio de claves SSL** de acceso para monitorización distribuída
- **Configuración de las directivas de monitorización** necesarias

En este caso vamos a definir un nodo master y un endpoint a modo de cliente que ejecutará los chequeos y reportará al master.

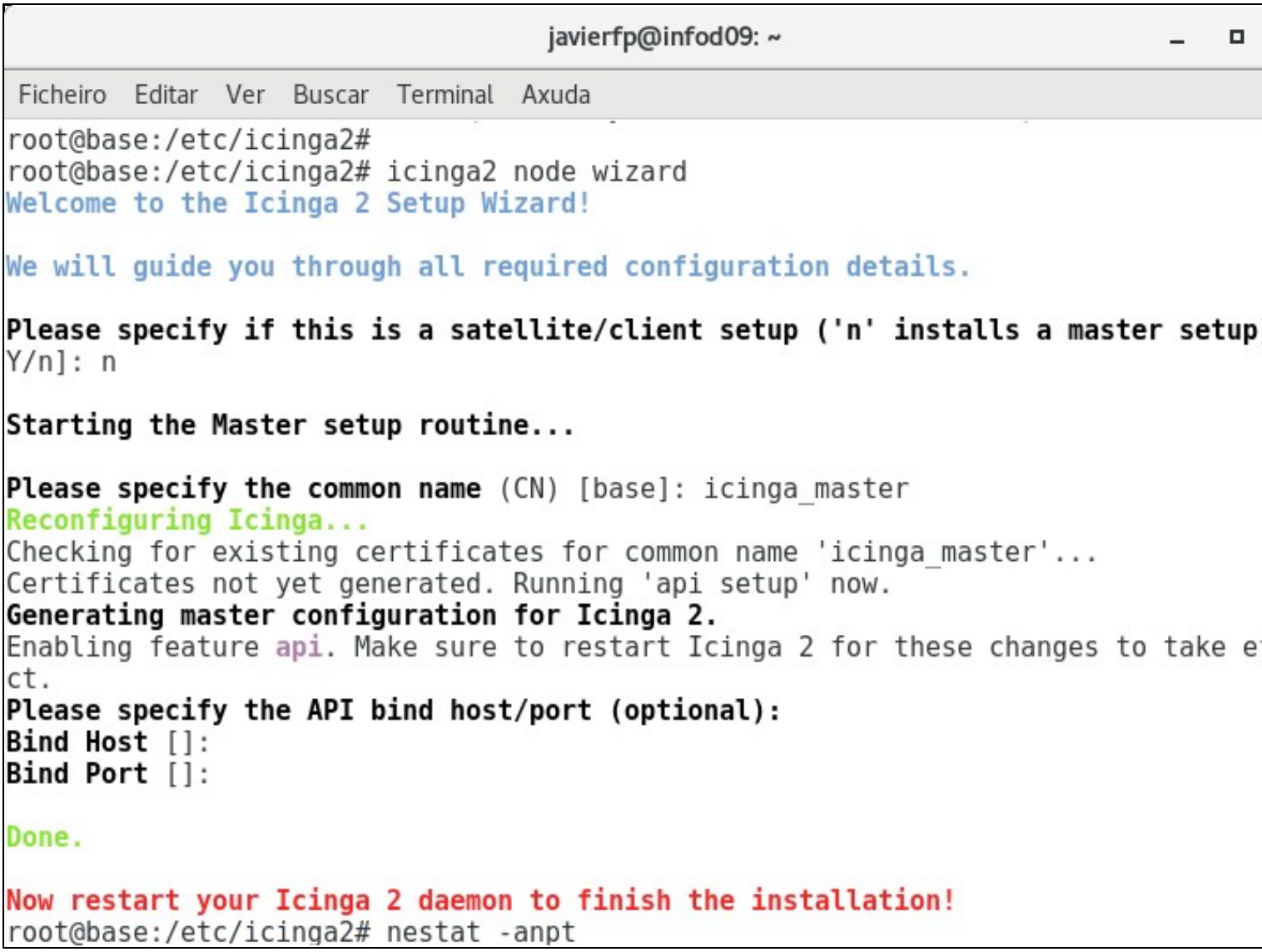
Configuración del Master

En primer lugar necesitaremos configurar Icinga para que pueda comunicarse de modo seguro con otros host. En este modo de operación el host monitorizado, en este caso el Windows 10, correrá las herramientas de monitorización de Icinga en local, y enviará reportes periódicos al servidor sobre su estado.

Para configurar el servidor Icinga como nodo master en la infraestructura de monitorización distribuida, ejecutamos:

```
icinga2 node wizard
```

Este comando lanzará un pequeño asistente al que deberemos responder "n" a la primera cuestión, la cual configurará el nodo como **master**, a continuación facilitaremos un CN (common name) para el Nodo, en este caso **icinga_master**


javierfp@infod09: ~
Ficheiro Editar Ver Buscar Terminal Axuda
root@base:/etc/icinga2#
root@base:/etc/icinga2# icinga2 node wizard
Welcome to the Icinga 2 Setup Wizard!
We will guide you through all required configuration details.
Please specify if this is a satellite/client setup ('n' installs a master setup
Y/n]: n
Starting the Master setup routine...
Please specify the common name (CN) [base]: icinga_master
Reconfiguring Icinga...
Checking for existing certificates for common name 'icinga_master'...
Certificates not yet generated. Running 'api setup' now.
Generating master configuration for Icinga 2.
Enabling feature api. Make sure to restart Icinga 2 for these changes to take e
ct.
Please specify the API bind host/port (optional):
Bind Host []:
Bind Port []:
Done.
Now restart your Icinga 2 daemon to finish the installation!
root@base:/etc/icinga2# nestat -anpt

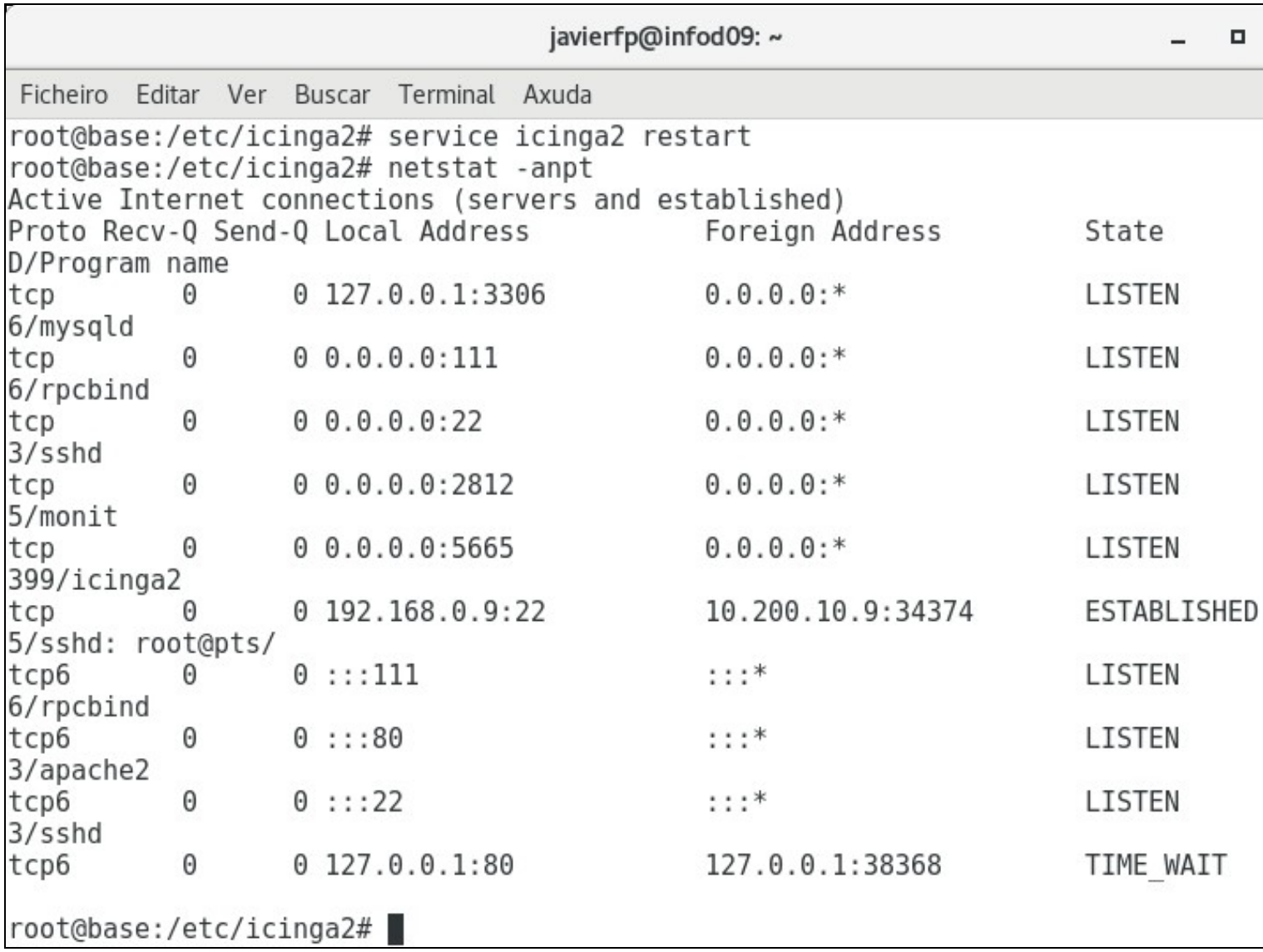
Tras lo cual reiniciamos el servicio

```
systemctl restart icinga2.service
```

A continuación ejecutamos (si no está instalado podemos instalar el paquete **net-tools**)

```
netstat -anpt
```

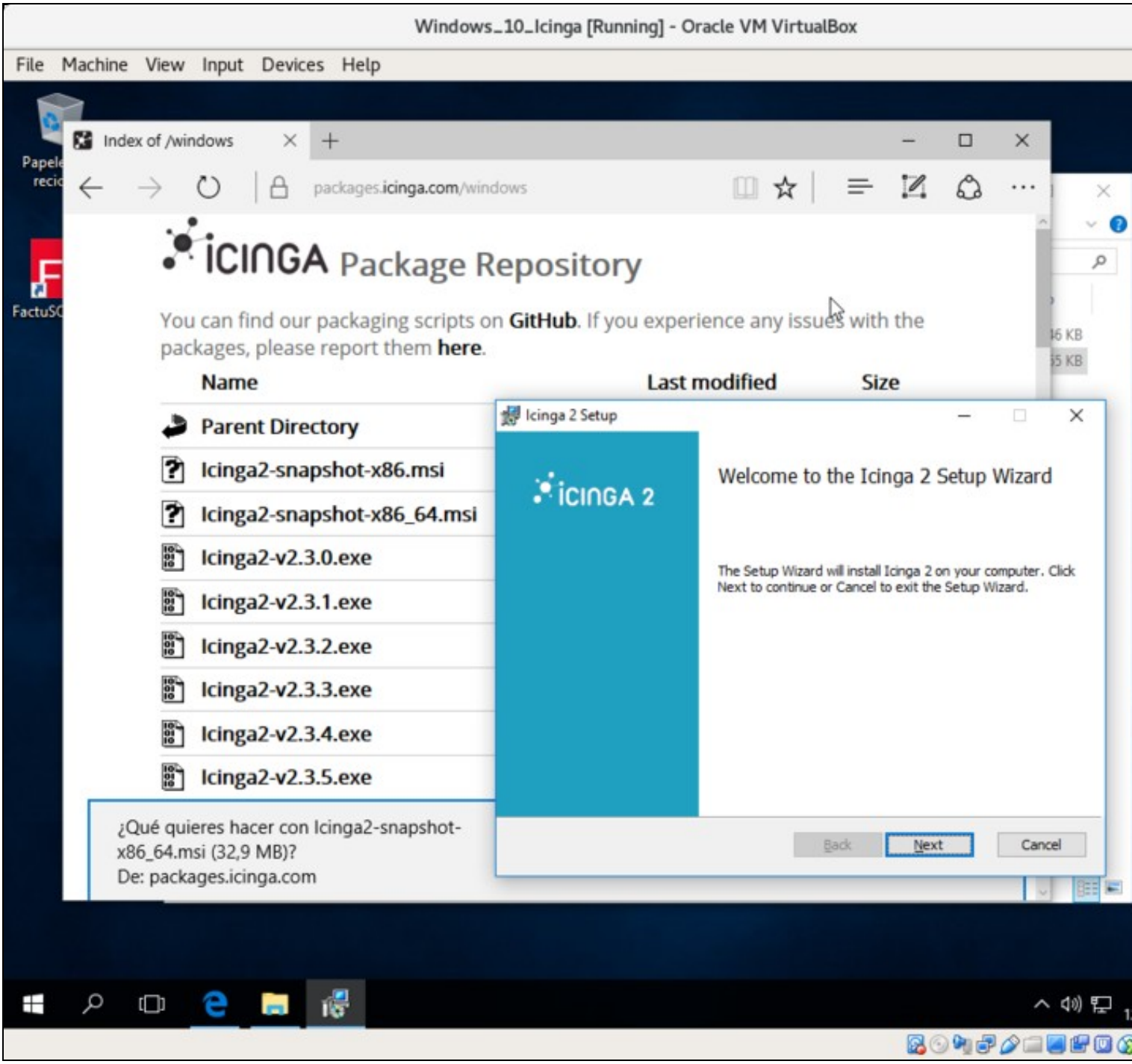
cuyo resultado nos mostrará como hay corriendo un proceso en el puerto 5665, que es el utilizado para intercomunicación entre nodos Icinga.



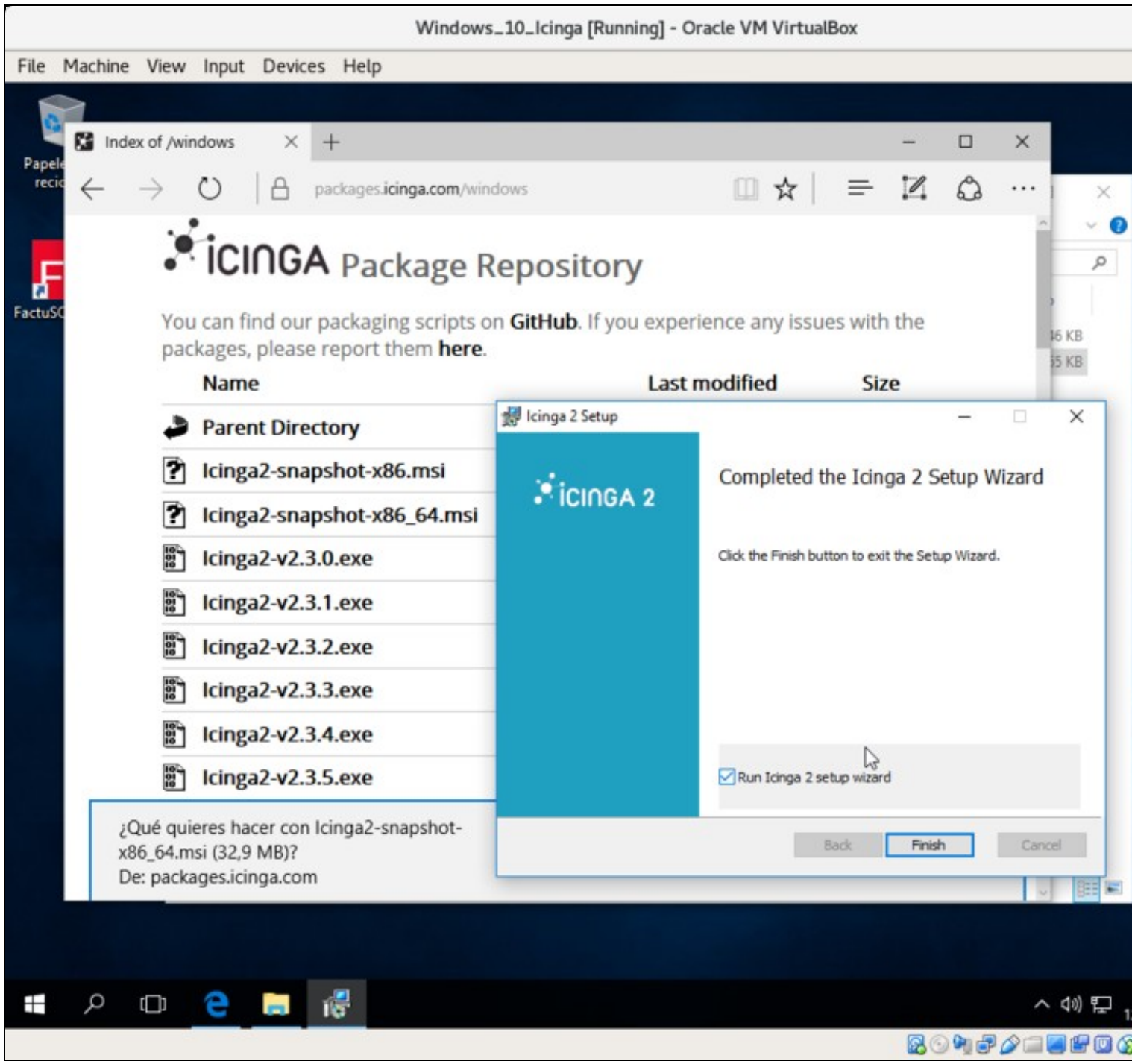
Configuración del host Windows a monitorizar

Desde el equipo Windows 10 descargamos el paquete instable **icinga2-snapshot-x86_64.msi** desde http://packages.icinga.com/windows

Una vez descargado lanzamos el .msi



Seguimos el asistente pulsando Next en todos los pasos



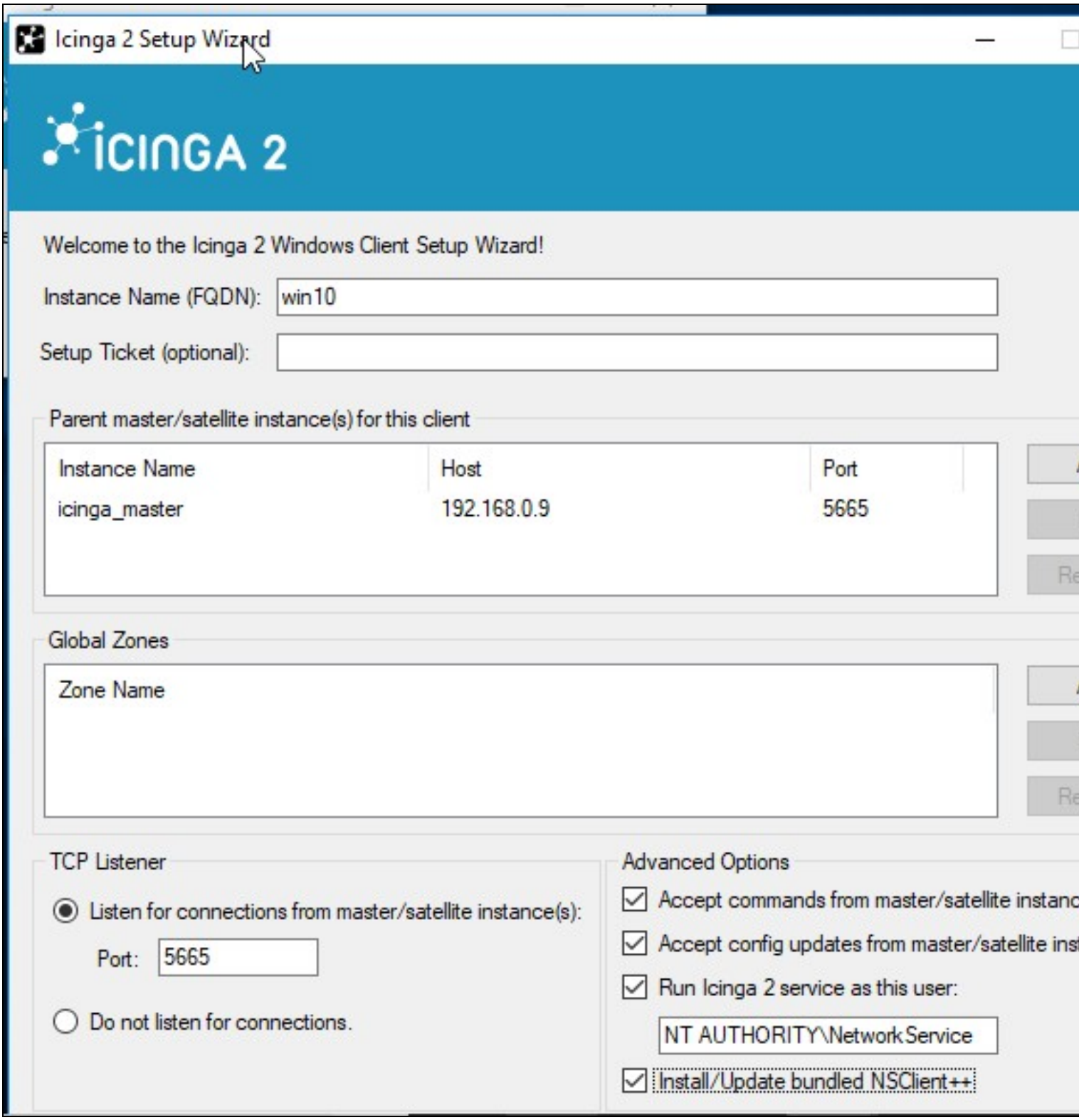
Ahora ejecutaremos la configuración de Icinga en el nodo Windows 10, el cual se lanzará tras pulsar "Finish" tras la instalación del .msi anterior

En la ventana del setup establecemos

- El **Instance Name**: Nombre del Nodo Windows en este caso, le asignaremos el valor **win10**, aunque se recomienda utilizar nombres FQDN

A continuación pulsamos en el botón Add de la zona "Parent/master/satelite...", en la ventaja emergente que aparece configuramos

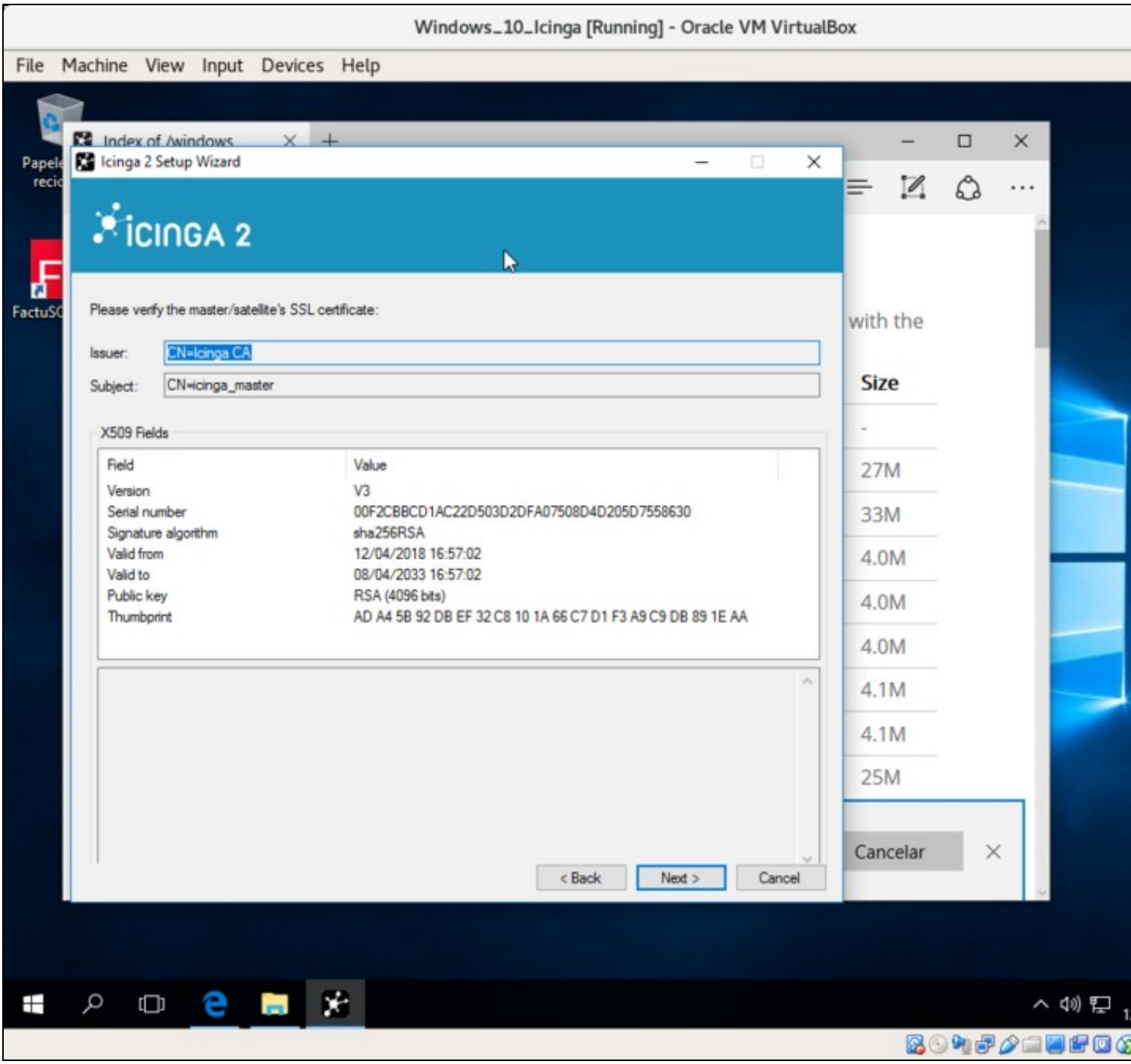
- El **Instance Name**: En este caso correspondiente al Icinga master, que para nosotros era icinga_master, y que configuramos previamente en la máquina Debian donde corre el servidor Icinga
- El **Host**: hostname o dirección IP del master, en este caso la IP del nodo del Servidor de Icinga



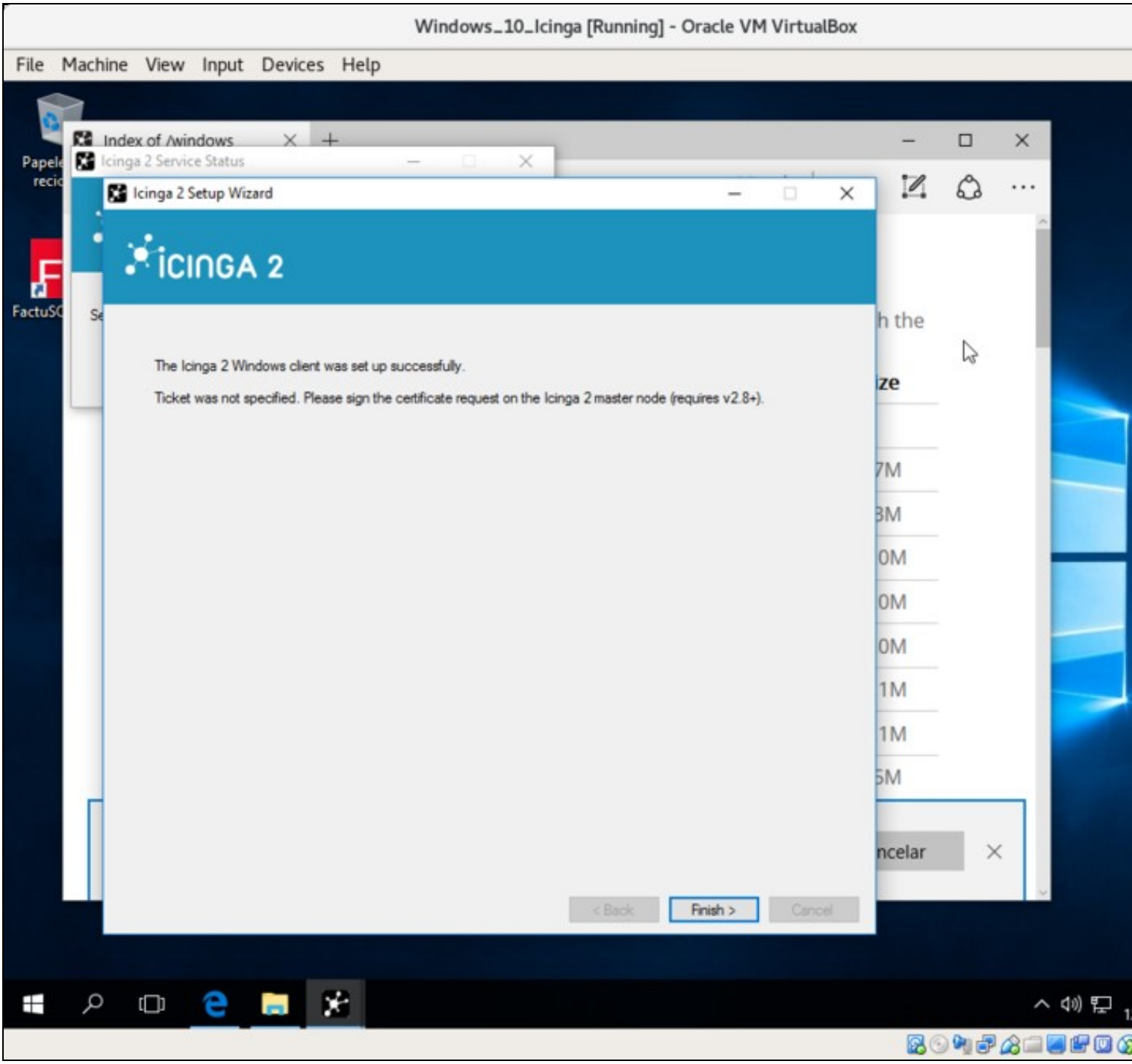
Activar todos los checks de las opciones de la captura anterior. Al final del asistente se instalarán las herramientas de cliente NSClient++

Pulsamos "Ok" y "Next"

En la siguiente captura se ve como se configura el certificado para la autenticación SSL entre master (Servidor Icinga) y el nodo cliente (Windows 10)



Vemos como termina el asistente de modo correcto



NOTA El servicio Icinga escucha en el puerto 5665, por tanto habrá que tener en cuenta este aspecto en el firewall de Windows. El propio asistente de configuración debería activar la regla correspondiente, pero no está de más revisar ante cualquier tipo de error.

Creación de los objetos para monitorización en el Nodo master

En la máquina Debian 9 donde corre el servidor Icinga editamos **/etc/icinga2/zones.conf**

El contenido de ese archivo será:

```
object Endpoint NodeName {
}

object Zone "master" {
        endpoints = [ NodeName ]
}

object Endpoint "win10" {
  host = "192.168.0.8"
}

object Zone "win10" {
  endpoints = [ "win10" ]
  parent = "master"
}

object Zone "global-templates" {
        global = true
}

object Zone "director-global" {
        global = true
}
```

En esta configuración se establece

- Un objeto **Endpoint** para el propio Nodo master de Icinga
- Un objeto **Zone** para el Nodo master de Icinga al que se añade como Endpoint el propio nodo Icinga
- Un objeto **Endpoint** que define el "otro extremo" a monitorizar, en este caso a través de el atributo **host** se establece la IP de la máquina Windows
- Un objeto **Zone** para el Nodo a Windows 10 a monitorizar, podrían añadirse más elementos a la zona en el atributo **endpoints**. Este objeto Zone está subordinado al objeto node creado con anterioridad para el master, este aspecto se define en la directiva **parent**.

Añadimos al final del archivo /etc/icinga2/conf.d/hosts.conf

```
object Host "win10" {
  import "generic-host"
  address = "192.168.0.8"
  vars.http_vhosts["http"] = {
    http_uri = "/"
  }
    vars.disks["disk"] = {
    }
  vars.disks["disk /"] = {
    disk_partitions = "/"
  }
  vars.notification["mail"] = {
    groups = [ "icingaadmins" ]
  }
  vars.client_endpoint = "win10"
  vars.os_type = "windows"
  vars.os_version = "10"
}
```

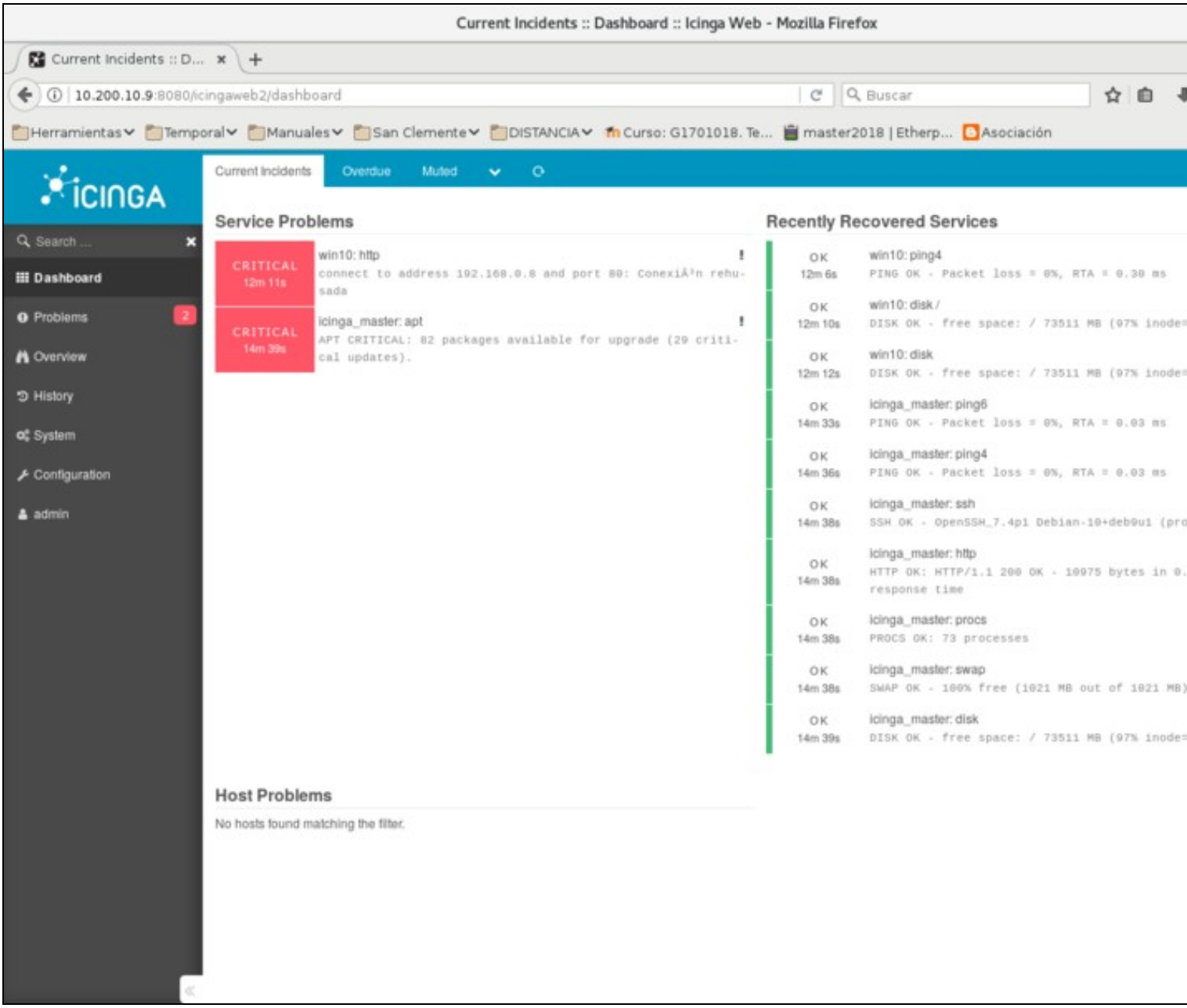
En esta configuración se establece

- Un objeto **Host** que define los atributos del Nodo Windows a efectos de Monitorización como ya vimos en otros ejemplos

Tras la definición de estos elementos reiniciamos Icinga

```
systemctl restart icinga2.service
```

Por último, accedemos al interfaz IcingaWeb para comprobar como efectivamente se realiza la monitorización del Nodo Windows 10



Otros chequeos

Utilizando el agente de monitorización **NSClient**, el cual es compatible con Nagios e Icinga, vamos a efectuar una serie de checks adicionales al nodo Windows.

En el nodo master editamos el archivo **/etc/icinga2/conf.d/services.conf** al que añadimos al final

```
#Monitorizar nodo Windows
/* CPU */
apply Service "nscp-local-cpu" {
  check_command = "nscp-local-cpu"
  command_endpoint = host.vars.client_endpoint

  vars.nscp_cpu_showall = true

  assign where host.vars.client_endpoint && host.vars.os_type == "windows"
}

/* Memory */
apply Service "nscp-local-memory" {
  check_command = "nscp-local-memory"
  command_endpoint = host.vars.client_endpoint

  vars.nscp_memory_showall = true

  assign where host.vars.client_endpoint && host.vars.os_type == "windows"
}
```

Se efectúan los checks:

- **CPU**: mediante el apply efectúa un check de CPU a los host que verifican que tienen establecido el atributo host.vars.client_endpoint y que el valor del atributo host.vars.os_type es "windows". En ese caso el check_command efectúa el check de CPU contra el objeto command_endpoint indicado or el atributo host.vars.client_endpoint, el cual establecimos en la definición del nodo Windows.
- **Memoria**: consideraciones similares se hacen para el check de memoria, en este caso cambian el check_command

Referencias

Inicio rápido: https://www.icinga.com/docs/icinga2/latest/doc/02-getting-started/

Instalación de Icinga2: https://www.icinga.com/docs/icingaweb2/latest/doc/02-Installation/

Principios de monitorización con Icinga: https://www.icinga.com/docs/icinga2/latest/doc/03-monitoring-basics

Plugins de monitorización: https://www.icinga.com/docs/icinga2/latest/doc/05-service-monitoring/#service-monitoring-plugins

Monitorización Windows con NSClient++: https://www.icinga.com/2017/07/05/monitoring-windows-clients-with-icinga-2-and-local-nsclient-checks/

Monitorización distribuida: https://www.icinga.com/docs/icinga2/latest/doc/06-distributed-monitoring/#roles-master-satellites-and-clients

Volver

JavierFP 19:10 09 vie 2018 (CET)