UNC | Universidad Nacional

Cátedra de Sistemas Operativos II

Trabajo Práctico Nº III

Illesca Martín Andres

4 de Junio del 2020

Índice

Introducción

Propósito

El proposito de este trabajo es la implementacion de un servidor web a traves del uso de systemd,nignx y el framework ulfius para manejo de http.

Ámbito del Sistema

el sistema esta diseñado para su uso en sistemas operativos del tipo linux ya que utiliza systemd para poder montar dos servicios y habilitarlos al momento de encender el servidor, en principio esta pensado para una arquitectura de 32 bits pero podria funcionar tambien en una de 64. El servidor por su parte se ha probado en una maquina virtual con linux de 32 bits.

Definiciones, Acrónimos y Abreviaturas

Referencias

1. man de linux
2. https://docs.nginx.com/
3. https://github.com/babelouest/ulfius
4. https://github.com/babelouest/yder
5. https://api.jquery.com/

Descripción General del Documento

Descripción General

Perspectiva del Producto

El producto esta desarrollado en lenguaje c haciendo uso tambien de nginx para reverse proxy y systemd para definicion de servicios.

El mismo cuenta con 2 binarios, 2 servicios de systemd y 1 site de nginx

- user: binario que corre el servicio de usuario esta copiado sobre */usr/*bin/tp3_user.

- servers: binario que corre el servicio de usuario esta copiado sobre *usr/*bin/tp3_servers.
- tp3_users.service,tp3_servers.service: archivos unit de systemd en este se define el usuario dueño de los procesos y el binario que debe ejecutar y se configura el servicio para su ejecucion.
- tp3: archivo de texto para configuracion del sitio de nginx esta configurado para correr en el puerto 80 y redirije las llamadas a los servicios. correspondientes, se establece tambien el archivo de contraseñas para poder acceder al servidor.

Funciones del Producto

- Obtener listado de usuarios a traves del endpoint /*api*/users con un metodo HTTP GET, este devuelve una lista con usuarios los cuales tendran id y nombre de usuario.
- Crear un usuario nuevo del endpoint /*api*/users con un metodo HTTP POST este, en caso de que exista un usuario con el nombre provisto devolvera error y tambien lo hara en caso de que el usuario o contraseña no sean strings.
- Obtener informacion del servidor donde corre la aplicación, esto se hara atraves de una llamada al endpoint /api/servers/hardwareinfo y listara informacion sobre el sistema operativo y el hardware del servidor.

Características de los Usuarios

Se puede acceder al servidor tanto por una herramienta al estilo de curl o postman como a traves de un navegador, en ambos casos es necesario saber el usuario y contraseña de nginx y la ip del servidor, se incluye una pagina sencilla escrita en js para el acceso a los datos, la misma estara en el servidor y se podra acceder mediante la url {ip_server}/pagina.

Restricciones

- los usuarios creados no tendran permisos de administrador.
- No pueden existir nombres de usuarios repetidos.
- no pueden existir nombres de usuarios que incluyan espacios

- solo se accedera al servicio ingresando el usuario y contraseña correcto para su uso.
- Solo se aceptan strings como formato para el username y el password.

Suposiciones y Dependencias

Se asume que los dispositivos donde se corra el producto cuenten con un sistema operativo del tipo Unix. Ademas los mismos deben contar con las siguientes aplicaciones/librerias:

- gcc
- openssl
- nginx
- apache2-utils
- cppcheck
- ulfius

Requisitos Futuros

En un futuro podria considerarse implementar aun mas endpoints para poder extender la funcionalidad del producto.

Requisitos Específicos

Interfaces Externas

- En este caso se haria uso remoto del software, el cliente deberia tener acceso a una pantalla y un teclado al menos para poder hacer los request.
- Es posible el uso tanto a traves de un navegador como a traves de curl o postman para dar un ejemplo por lo tanto el usuario deberia tener acceso a alguna de esas opciones.

Funciones

El usuario cuenta con dos servidores entre los cuales suman tres endpoints accesibles:

- */api/users GET :metodo para obtener listado de usuarios del sistema, devuelve una lista con los usuario siendo cada uno un objeto json de la forma:*

  {user_id:int,

  username:const char}*

- */api/users POST: metodo para crear usuario nuevo espera un content-type:"application/json" el cual debe estar compuesto de la siguiente manera:*

  {

  username:const char,*

  *password:const char**

  }

  en caso de éxito devuelve un json de la forma:

  {

  id:int

  *username:const char**

  created_at:date

  }

  en caso de error devuelve un json con la descripcion de dicho error.

- */api/servers/hardwareinfo: GET devuelve informacion acerca del SO y hardware del servidor.*

Requisitos de Rendimiento

se espera un tiempo de respuesta maximo del servidor de 200 ms.

Restricciones de Diseño

- Los servicios deben estar configurados con systemd para la ejecucion automatica de los mismos.
- Debe usarse nginx para el redireccionamiento de los request a cada servicio correspondiente.
- Debe hacerse el log de los procesos en *var*/log, los procesos deben hacer un log cada vez que atienden un request.

Atributos del Sistema

Otros Requisitos

Diseño de solución

Los servicios se desarrollaron en lenguaje c contando con dos archivos:

- user.c: en este se hacen todas las configuraciones para atender a los endpoint api/users
- servers.c: es este servicio se atiende a los endpoints del tipo api/servers.

el proyecto a ademas cuenta con una estructura de directorio como se detalla a continuacion:

partiendo del directorio raiz del proyexto se tiene las siguientes carpetas/archivos

- ./sources : contenedora de los archivos .c antes mencionado.
- ./resources: carpeta de recursos adicionales como el codigo de la pagina y los archivos de configuracion de nginx y systemd

Luego una vez que se ejecute make se creara una carpeta ./bin con todos los archivos ejecutables y .o.

El makefile proyecto a su vez cuenta con una instrucción install la cual se encarga de copiar los archivos de configuracion en los directorios adecuados (/lib/systemd/system/ para systemd ,/etc/nginx/sites-available creando ademas el link simbolico a sites-enabled), ademas se copian los binarios de la carpeta bin en /usr/bin, se crea el archivo de log en *var*/log/tp3 , se crea el usuario de los servicios en el sistema(tp3user) otorgandole servicios de sudoer a este y se le configura NOPASSWD para que este pueda correr como supuer usuario sin necesidad de ingresar la contraseña, tambien se inician y habilitan para ejecucion desde el inicio a los servicios de systemd.

Implementación y Resultados

el producto fue codificado y configurado mediante el uso del framework sugerido y cumpliendo las restricciones y caracteristicas antes especificadas, se han corrido los tests propuestos en clase y los mismos han sido exitosos, se comprobo el funcionamiento a su vez de la configuracion de systemd comprobando que realmente los procesos estaban corriendo desde el arranque y podian utilizarse sin necesidad de correrlos a mano.

Conclusiones

Se ha comprobado que el uso de frameworks facilita mucho la implementacion de una api REST mas en un lenguaje no tan flexible como es c ademas de la practicidad de poder definir los servicios de systemd para la ejecucion automatica de los servicios y de poder redireccionar desde nginx de modo que no cambia el la direccion del server (ni el puerto) para las peticiones, se pudieron usar herramientas de nginx para la basic auth del servicio y para la conservacion de la ip real desde la cual se llama al endpoint y se pudo utilizar las los comandos de systemd (restart,start,stop) para actualizar el comportamiento habiendose actualizado el archivo binario, ademas se adquirio experiencia en la elaboracion de un make install pudiendo implementarlo correctamente y corrido la instalacion con éxito verificando que se cumplieron todos los procedimientos arriba expresados.

Apéndices