Marcologger

Diego Martín Arroyo

Viernes, 15 de mayo de 2015

Resumen

Marcologger es una herramienta que permite observar la salida de un programa distribuido desde una única interfaz, integrándose además en rsyslog, utilizando Snorky para la gestión de los WebSockets.

Índice

Introducción

Uno de los problemas típicos a la hora de desarrollar aplicaciones distribuidas es analizar el comportamiento de una ejecución, en particular a la hora de realizar procesos de depuración. Uno de los motivos por los que este problema aparece es la ausencia de un periférico de salida (pantallas, indicadores, etcétera), que el programador pueda interpretar. Con el objetivo de facilitar esta tarea se crea **marcologger**.

Requisitos

Los requisitos del sistema son los siguientes:

- Las salidas estándar y de error (**stdout**, **stderr**) deben ser observables y diferenciables.
- La información debe ser observable en el momento que se genera.
- La información sobre cada nodo y ejecución debe ser diferenciable.
- No deben existir cuellos de botella, las conexiones se deben realizar entre los nodos que realizan la ejecución y el cliente.
- Únicamente el usuario que ha realizado la ejecución está autorizado a visualizar la información de salida.
- El sistema debe realizarse en el *framework* **Tornado**.s

Funcionamiento

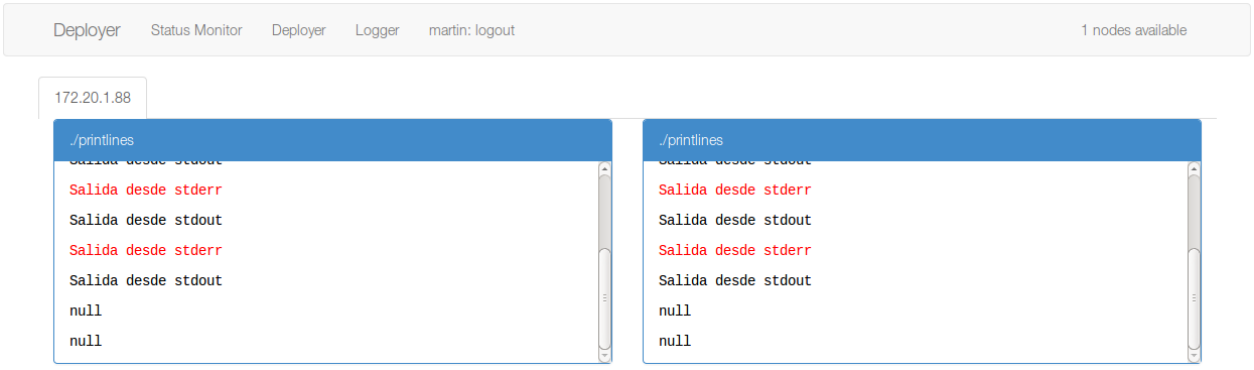

Figura 1: Interfaz principal del sistema

marcologger se diseña con el objetivo de ser integrado en la herramienta **marcodeployer**, a fin de que un usuario pueda evaluar la salida de un comando introducido en la interfaz al realizar un despliegue o bien de forma individual. Una vez que el archivo ha sido distribuido en cada nodo comienza a ejecutarse el comando, redirigiendo las salidas especificadas a una conexión **WebSocket** con el cliente.

Arquitectura

La aplicación está conformada por los siguientes componentes:

- **Tornado** como servidor web.
- **Snorky** para la gestión de los **WebSocket**
- **Jade** como motor de renderizado de plantillas (mediante **pyjade**.
- **jQuery** como biblioteca para la gestión de peticiones asíncronas y gestión de los diferentes elementos con el *plugin* **jQuery Cookie**.
- **Bootstrap** para la gestión de los elementos visuales.
- **MarcoPolo** para la detección de los diferentes nodos.
- **PAM** para la gestión de usuarios mediante la interfaz **python-pam**.

En los nodos sobre los que se realizan las ejecuciones se ejecuta una instancia de **receiver.py**, que se encarga de la recepción de los diferentes ejecutables y comandos a ejecutar.
En el nodo al que se conecta el cliente (en caso de que no sea su propio equipo) se ejecuta una instancia de **deployer.py**, encargado de la generación de la interfaz web, detección de nodos y realización del despliegue.
El cliente web sigue la siguiente secuencia de pasos:

1. Una vez que todos los archivos han sido recibidos, solicita la lista de nodos sobre los que puede realizar despliegues.
2. Establece una conexión mediante un **WebSocket** para determinar si será posible crear las conexiones posteriores. Este paso es necesario debido a la arquitectura X.509 utilizada. (Ver **??**).
3. Al ordenar un despliegue, el sistema crea conexiones **WebSocket** a cada uno de los nodos sobre los que se desee ejecutar el comando[1]
4. Al producirse una salida por pantalla, el nodo envía al cliente un mensaje con la información sobre el ejecutable que ha generado la cadena de texto, y el contenido de la misma.
5. Al recibir el mensaje, el cliente determina en qué elemento de la interfaz debe mostrarse la información según los datos del mismo.
6. Una vez que los *buffers* han sido cerrados en el nodo, el cliente recibe un mensaje para conocer dicho cambio de estado, y actúa en consecuencia, indicándolo por pantalla.

Funcionamiento

La autenticación en el cliente se realiza mediante la interfaz PAM disponible en Python. En caso de ser exitosa, el cliente recibe una cookie segura, firmada mediante HMAC y marcada temporalmente. Dicha cookie es enviada en todas las transmisiones WebSocket como token de autenticación (los nodos son capaces de descifrar el valor de la misma al conocer la clave criptográfica con la que se ha realizado el proceso de firmado).
Antes de realizarse el despliegue se crea el canal **WebSocket**. Todas las salidas se envían por el mismo canal. Los nodos mantienen una tabla clave-valor para identificar los **WebSockets** abiertos, siendo la clave el nombre de usuario en el sistema (mediante PAM dichos nombres son homogéneos en el sistema). Cada ejecución cuenta con un identificador aleatorio que el servidor comparte con el cliente.

[1] No es posible determinar en qué momento comenzará a existir una salida por pantalla, por lo que la conexión sobre la que se redirige la misma se crea antes de que la ejecución comience.

Seguridad

UX y CN

La arquitectura del sistema se compone de una interfaz web creada siguiendo un modelo *Single Page Application*, que ofrece una experiencia de usuario más dinámica, al evitar regargar el navegador por completo.