Informe Laboratorio de Sistemas Operativos y Redes: Buildbot integrado con Git

Integrantes:

- María Belén Amat
- Lautaro Lozano
- German Cigala
- Nicolas Rodriguez

Profesor: Jose Luis DiBiase

Guía de instalación e implementación en Ubuntu 16.04

¿Qué es Buildbot?

Buildbot es un software de integración continua que se encarga de automatizar la compilación de un código y la ejecución de sus respectivos tests para validar los cambios que los autores del proyecto vayan realizando con el correr del tiempo.
Está implementado en Python y su configuración se realiza también en Python, aunque puede utilizarse para ejecutar cualquier tipo de tarea. Tiene una licencia pública general, lo que le garantiza a los usuarios la posibilidad de ver, estudiar, compartir y modificar el código fuente.
Buildbot encola tareas, luego las ejecuta e informa a los usuarios sobre posibles errores. Estas notificaciones pueden realizarse a través de un correo electrónico, un web status, e IRC entre otros.

¿Qué es Git?

Git es un software de control de versiones. Un control de versiones se define como la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Git permite que varios usuarios se unan a un proyecto y reciban todas las actualizaciones de versiones que van subiendo los otros usuarios. También permite crear distintas ramas (branches) para distribuir el trabajo de los distintos desarrolladores.

Requisitos

-*Un servidor corriendo en Ubuntu 16.04*

Lo primero que tenemos que hacer es conectarnos en el servidor mediante el protocolo SSH

-*Python*

Para instalar la versión 2.7 de Python (que es la recomendada por la página oficial de Builbot) abrimos una terminal e ingresamos lo siguiente:

```
sudo wget https://www.python.org/ftp/python/2.7.14/Python-
2.7.14.tgz
```

Extraemos el paquete descargado

```
sudo tar xzf Python-2.7.14.tgz
```

Por último, compilamos

```
cd Python-2.7.14
sudo ./configure
sudo make altinstall
```

-*Pip*

¿Qué es PIP? Pip viene del acrónimo recursivo, “Pip Installs Packages” o “Pip Installs Python”. Esto significa que este programa hace una gestión de paquetes y se utiliza para instalar y desinstalar software que se ha escrito para Python. Para instalarlo desde la terminal:

```
sudo apt-get install python-pip
```

Instalación de Buildbot

Comenzaremos con la instalación del paquete Buildbot que incluye un master, un worker y todas las dependencias necesarias, incluso las que sean requeridas por la interfaz web.
Pip crea archivos de extensión .cache en donde se ejecuta. Vamos a usar el flag –H para poner esos archivos en el lugar correcto

```
sudo -H pip install
```

Si la instalación fue exitosa, deberíamos ver lo siguiente:

```
Successfully installed Automat-0.5.0 Jinja2-2.9.6 MarkupSafe-1.0
PyJWT-1.5.0 Tempita-0.5.2 Twisted-17.1.0 attrs-16.3.0 autobahn-
17.5.1
buildbot-0.9.6 buildbot-console-view-0.9.6 buildbot-waterfall-view-
0.9.6
buildbot-worker-0.9.6 buildbot-www-0.9.6 constantly-15.1.0
decorator-4.0.11 future-0.16.0 incremental-16.10.1 pbr-3.0.0
python-dateutil-2.6.0 six-1.10.0 sqlalchemy-1.1.9
sqlalchemy-migrate-0.11.0 sqlparse-0.2.3 txaio-2.7.1
zope.interface-4.4.0
'buildbot[bundle]'
```

Luego de hacer esto tenemos que volver a actualizar nuestro PIP.

```
sudo -H pip install --upgrade pip
```

Lo último que haremos será verificar que buildbot se instaló correctamente, y para eso verificaremos la versión.

```
buildbot --version
```

```
Buildbot version: 0.9.6
Twisted version: 17.1.0
```

Tenemos que verificar que el firewall UFW solamente permite el tráfico de paquetes que trabajan bajo el protocolo SSH. Un UFW es un firewall de uso sencillo, escrito en Python para Ubuntu.

```
sudo ufw status
```

Primera complicación

Al chequear el estado del Firewall, la salida debería decir lo siguiente:

```
Output
Status: active

To                         Action      From
--                         ------      ----
OpenSSH                    ALLOW       Anywhere
OpenSSH (v6)               ALLOW       Anywhere
(v6)
```

Nos encontramos con el problema de que el estado nos aparecía “Inactive”. Nuestro problema era que solo teníamos instalado el cliente SSH y nos faltaba instalar el servidor.

Solución

Tuvimos que instalar el servidor SSH con el siguiente comando

```
sudo apt install openssh-server
```

Finalmente, reiniciamos el servicio

```
sudo systemctl restart sshd.service
```

Buildbot utiliza el puerto 8010 para el uso de la interfaz web. Pero hubo que activarlo

```
sudo ufw allow 8010
```

Agregamos un grupo y usuario para poder correr los servicios de Buildbot

```
sudo addgroup --system buildbot

sudo adduser buildbot --system --ingroup buildbot --shell /bin/bash
```

Y nos logeamos

```
sudo --login --user buildbot
```

¡Listo! Pasemos a configurar el master.

Segunda complicación

Para poder comenzar con la configuración del master, teníamos que logearnos en buildbot pero no lo conseguíamos. Tuvimos que volver a crear el grupo y el usuario.

Configuración del Master

El master es el servidor en donde corre Buildbot el cual posee toda la configuración. Para crearlo utilizamos el siguiente comando. Pusimos ~/master para que se cree en ese directorio.

```
buildbot$ buildbot create-master ~/master
```

Tuvimos que copiar el archivo master.cfg.sample a master.cfg

```
buildbot$ cp ~/master/master.cfg.sample ~/master/master.cfg
```

A continuación, tuvimos que editar dicho archivo (utilizamos el editor nano). Para poder acceder a una interfaz web con builbot, había que modificar el archivo para declarar cuál era la dirección IP a la que buildbot debería conectarse para poder establecer vínculo con el servidor.

Reemplazamos esta línea del archivo:

```
c['buildbotURL'] = "http://IP or site domain:8010/"
```

Por esta:

```
c['buildbotURL'] = "http://Localhost:8010/"
```

Como el servidor al que debe conectarse buildbot está configurado en la misma computadora que él mismo, completamos la línea con Localhost

Para poder configurar el envío de las estadísticas de Buildbot a nosotros los usuarios, tuvimos que poner la variable buildbotNetUsageData en basic.

Otra vez, utilizamos el editor de textos nano para cambiar esta línea

```
c['buildbotNetUsageData'] = None
```

Por esta

```
c['buildbotNetUsageData'] = 'basic'
```

Chequeamos errores de compilación en la configuración del master con la siguiente línea

```
buildbot$ buildbot checkconfig
```

Iniciando el servicio

Una vez configurados el archivo de configuración del master tuvimos que iniciar el servicio.

```
buildbot$ buildbot start ~/master
```

Para chequear si el servicio se inició, accedimos al sitio http://localhost:8010 desde un navegador.

Configuración de los Workers

Para poder crear un worker en el sistema, tuvimos que usar el comando create-worker pasándole varios parámetros

```
buildbot$ buildbot-worker create-worker ~/worker localhost example-worker pass
```

En donde:

- ~/worker es el nombre del directorio en donde se van a guardar las configuraciones del worker.
- Localhost es la dirección en donde está corriendo el master al que se tiene que conectar el worker.
- Example-worker es el nombre que le queremos poner al worker. Dentro del archivo de configuración ~/master/master.cfg los nombres de los workers no pueden repetirse.
- Pass es la contraseña para el worker que estamos instanciando y dicha contraseña debe coincidir con la que está en el archivo ~/master/master.cfg.

Nosotros lo completamos de la siguiente manera

```
buildbot$ buildbot-worker create-worker ~/labo localhost labo Admin1234
```

Dejamos la parte de localhost porque el servidor corre de manera local.

Tuvimos que configurar el worker desde el archivo de configuración master.cfg. Utilizamos, otra vez, el editor de textos nano para modificar el archivo.
En la parte de los workers, agregamos esta línea.

```
c['workers'] = [worker.Worker("labo", Admin1234) ]
```

Agregamos a los workers la instancia creada.
El nombre de usuario y la contraseña seteada cuando utilizamos el comando create-worker deben ser los mismos que cuando agregamos el worker a la lista en el archivo master.cfg

Para poder obtener más información sobre las fallas de los test que envían los workers. Para que esas fallas lleguen a una dirección de correo es necesario cambiar una línea de código dentro del archivo ~/worker/info/admin. Nosotros utilizamos nano para modificarlo.

```
Your Name Here <admin@youraddress.invalid>
```

```
Belen Amat <belen.amat29@gmail.com>
```

Una vez creado el worker y seteadas las configuraciones, hay que hacer que el worker establezca una conexión con el servidor. Para eso se usa el comando start.

```
buildbot$ buildbot-worker start ~/worker
```

Tercera complicación

Al intentar establecer una conexión entre el worker y el servidor, nos aparecía un error que decía que al Worker le tomo más de 10 segundos establecer la conexión y que no había manera de garantizar que se haya llevado a cabo de manera exitosa. Esto nos sucedió varias veces. Probamos reiniciando el sistema, pero no pudimos solucionarlo de esa manera.

Lo que estaba ocurriendo era que el Worker estaba intentando conectarse al servidor a través de un puerto que no era por el que estaba escuchando el servidor.

Tuvimos que abrir el archivo de configuración de nuestro worker y editarlo para hacer coincidir los dos puertos.

En el tutorial que seguimos, el puerto del servidor estaba configurado como 8010. Nosotros cambiamos el puerto que teníamos configurado por defecto (9989) para que también sea el 8010. El problema surgió cuando el worker quiso conectarse a través del puerto 9989 al servidor.

Configuración avanzada del Master

Change sources

Un sistema de control de versiones mantiene un arbol de código fuente y le avisa al buildmaster cuando se realizan cambios. El primer paso de cada build casi siempre es adquirir una copia de ese repositorio.
La configuración de Change Sources es necesaria para que el buildmaster sepa cómo escuchar cambios.

Builders

Cada Builder le dice a Buildbot como realizar un build, es decir los pasos que debe realizar y sobre qué worker. Dentro del archivo master.cfg pusimos las siguientes líneas para declarar un builder y le agregamos dos “pasos” a realizar.

```
c['builders'] = []  → se comienza con una lista vacía de builders

bf = util.BuildFactory() → se crea una instancia de builder

bf.addStep(steps.Git(repourl='git@github.com:gercgl/labo.git',
mode='full')) → Se agrega un paso al builder que declara cual será
el repositorio de github sobre el que se va a trabajar

bf.addStep(steps.ShellCommand(command=["sh", "build.sh"],
                              workdir="build")) → recibe los strings
de la lista como commandos y los ejecuta sobre el archive build.sh

c['builders'].append(util.BuilderConfig(name="drupalbuilder",
workernames=['drupal'], factory=bf)) → se agrega al builder a la
lista de builders
```

Schedulers

Los schedulers son los responsables de decidir en orden y criterio bajo el que se ejecutan los distintos builds de un builder. Un build es un conjunto de instrucciones a ejecutar por Buildbot. Hay varios tipos de schedulers. Algunos schedulers esperan un determinado tiempo desde que llega el build para ejecutarlo. Otros, generan otros builds sin cambios basados en eventos que ocurren en el master.

Configuración de los schedulers

La configuración de los schedulers también se lleva a cabo dentro del archivo de configuración del master (master.cfg). Creamos un scheduler y seteamos sus configuraciones.

```
c['schedulers'] = []
```

→ iniciamos con una lista de schedulers vacía

```
c['schedulers'].append(schedulers.SingleBranchSche
duler(
    ➔ Instanciamos un scheduler de la clase
      SingleBranchScheduler
            name="masterchange",
    ➔ Declaramos su nombre
            fileIsImportant=isImportant,
    ➔ Funcion que recibe un objeto change como
      parámetro que contiene toda la información de
      los cambios. La función decide si el cambio
      es importante o no
      change_filter=util.ChangeFilter(branch='maste
      r'),
    ➔ Filtra que los cambios recibidos sean del
      branch master
            treeStableTimer=60,
    ➔ Espera 60 segundos para ejecutar el build
            onlyImportant=True,
    ➔ Solo ejecuta el build si los cambios son
      importantes
            builderNames=["drupalbuilder"]))
```

Nosotros elegimos el Single Branch Scheduler, el cuál escucha los cambios que ocurren en una branch específica del repositorio. Permite configurar un tiempo de espera entre cambios y una vez que este expira ejecuta el build (conjunto de instrucciones).

Una vez instanciado el scheduler, guardamos los cambios y cerramos el archivo de configuración.

Change Hooks

Una de las opciones que brinda Buildbot es Change Hooks.
Al habilitar la configuración ‘change_hook_dialects’ habilita /change_hook, que es una url “magica” que brinda Buildbot para aceptar solicitudes HTTP y convertirlas en cambios para el buildmaster. El formato de la url es /change_hook/DIALECT , donde DIALECT es un paquete dentro del directorio de hook. En nuestro caso la url será “http://ip_donde_corre_buildbot/change_hook/github”
Change_hook esta deshabilitado por defecto y cada DIALECT tiene que ser habilitado de forma separada, por razones de seguridad.

GitHub WebHook

Para habilitar Github es necesario agregar las siguientes líneas en el archivo de configuración del master (master.cfg)

```
c['www']['change_hook_dialects'] = {
      'github': {
            'secret': 'your_secret_value',
      }
}
```

Desde Github debemos crear el webhook para indicar, por ejemplo dónde debe notificar mediante una petición HTTP cuando ocurre un evento.

Desde el administrador del repositorio ir a settings / webhooks / add webhook dónde se configura el webhook

Payload URL es la url dónde GitHub va a enviar la petición HTTP y en la cuál Buildbot va a tener que escuchar. En nuestro caso, debemos completarla con el host dónde tenemos el servicio Buildbot corriendo /change_hook/github

Content Type permite elegir el formato en que vamos a recibir los datos, por defecto es application/x-www-form-urlencoded

Secret es una clave que deberá configurarse en ambos servicios para una mínima validación.

Autentificación

Pudimos configurar un usuario y contraseña para proteger la interfaz pero requería configurar los otros componentes para que sigan funcionando como estaban

```
c['www']['authz'] = util.Authz(
        allowRules = [
        util.AnyEndpointMatcher(role="admins")
        ],
        roleMatchers = [
        util.RolesFromUsername(roles=['admins'],
usernames=[admin])
        ]
)
c['www']['auth'] = util.UserPasswordAuth({admin: p4ssw0rd})
```

Conclusion

La parte que más trabajo y complicaciones nos llevó fue la instalación, la configuración del sistema y de los puertos, y lograr la conexión entre los workers y el servidor.

Bibliografía

- https://svn.apache.org/repos/asf/subversion/branches/fs-successor-ids/tools/buildbot/master/master.cfg
- http://docs.buildbot.net/current/
- https://www.digitalocean.com/community/tutorials/how-to-install-buildbot-on-ubuntu-16-04
- https://developer.github.com/webhooks/

Link del repositorio GitHub

http://github.com/gercgl/labo