Memoria P4

David Carrascal Acebrón

Junio del 2019

Índice general

Capítulo 1

Lenguaje P4

1.1. Enlaces útiles

- Tutoriales: `https://github.com/p4lang/tutorials/`
- Protobuf: `https://developers.google.com/protocol-buffers/`
- gRPC: `https://grpc.io/`
- gRPC ejemplo: `https://www.nocountryforgeeks.com/grpc-protocol-buffers/`
- Lista de reproducción sobre conferencias: `https://www.youtube.com/playlist?list=PLRffjASqEa5VMsUml2V34_xDTeVydeM_L`
- Correo explicando BM2: `http://lists.p4.org/pipermail/p4-dev_lists.p4.org/2016-June/002112.html`

1.2. Papers de interés

- Paper sobre la agregación de tecnología p4 para parsear paquetes generados por sensores IoT a una WAN [2].
- Paper sobre como emplear Switches p4 frontera para servicios de IoT [3]
- Paper sobre el gran potencial que tiene p4 para interconectar el mundo SDN con la próxima generación de nodos finales que interactúan con clusters de TI innovadores, 5G fronthaul e internet de las cosas. [1]

1.3. Introducción teórica

En este punto se va a tratar de abordar todos los conceptos teóricos que nos vayan surgiendo según exploramos p4. Se intentará tratar de entender todas las tecnologías que emplea p4, cual es su uso, cuales son sus pros y sus posibles contras.

1.3.1. ¿Qué es P4?

1.4. Instalación

- Hemos instalado una máquina virtual de Ubuntu 16.04, con 20GB de almacenamiento ya que con 10 se nos queda corto. 4 GiB de ram y cuatro núcleos
- Hemos hecho un update y upgrade, e instalado git, vim, htop, openssh-server para conectarse vía ssh en remoto.
- Hemos hecho un git clone del repositorio con el tutorial: `https://github.com/p4lang/tutorials`
- Lanzamos los dos shell-scripts user-bootstrap.sh y root-bootstrap.sh.
- Hemos encontrado varios problemas en el momento de llevar a cabo la instalación. Por lo que se ha decidido modificar los shells-scripts de instalación ofrecidos en el repositorio.

Como se ha introducido antes, hemos tenido que crear nuestro propio script de instalación **install.sh**. Habrá que darle permisos de ejecución **chmod 777 install.sh** y lanzarlo con **sudo**.

La necesidad de crear otro método de instalación es la deficiencia del los métodos alternativos de instalación que tienen aparte de Vagrant. Ellos disponen de dos shell-scripts, donde crean un enviroment muy cercano a personas con poco conocimiento de Linux, facilitándoles user propio p4, generar iconos en el escritorio, configurar IDEs... Tareas propias de cada de desarrollador en función de sus preferencias.

Por lo que se ha decidido limpiar el proceso de instalación, dejando únicamente lo estrictamente necesario para llevar a cabo el tutorial de p4. Dejando únicamente un shell-script, más optimizado y más limpio. Se ha tenido que hacer una doble llamada al shell-script **autogen.sh** debido a este issue:

- `https://github.com/protocolbuffers/protobuf/issues/149`

Como forma de agradecer a la organización de p4 el hecho de que tengan un buen repositorio de tutoriales dando un buen entrypoint para los noobies que acaban de empezar con p4 (Yo) se ha ofrecido el script de instalación para que se incorpore al master.

Al hacer el pull-request, el mantenedor de los tutoriales le pareció una buena idea, pero el hecho de tener un método de instalación nuevo significa más trabajo para mantener el repositorio. Me pidió que tratara de incluir la instalación nativa de Vagrant para que así con un único script podamos tener la instalación sobre Vagrant y el método que habíamos propuesto. Para poder llevar esta tarea a cabo se tuvo que estudiar la herramienta de Vagrant. En los anexos se han puesto unas pinceladas de los aspectos más significativos de esta herramienta.

De momento no está incluido en el master pero se puede hacer de uso del nuevo método de instalación el fork creado de los tutoriales de p4.

- `https://github.com/davidcawork/testP4/tree/master/tutorial`

Se puede consultar el estado del pull-request aquí.

- `https://github.com/p4lang/tutorials/pull/261`

Para llevar a cabo la instalación del modo nativo sobre Vagrant se debe instalar Vagrant, instalar un proveedor de maquinas virtuales y cargar el VagrantFile. Para más información sobre el uso de Vagrant consulte la información referente a ello en los anexos.

1.5. Test P4

En esta sección vamos a seguir el tutorial y los test que proponen la organización de p4 en su repositorio oficial.

- `https://github.com/p4lang/tutorials`

La metodología de los tutoriales consiste en completar el esqueleto del código p4 dado por ellos para hacer funcionar las distintas pruebas. El escenario que se ha empleado para establecer un eviroment de pruebas es Mininet.

1.6. Test 1: Implementando el fordwarding básico

El objetivo de este test es escribir un programa P4 que implemente el fordwarding básico. Con el reenvío de IPv4, el conmutador debe realizar las siguientes acciones para cada paquete:

- Actualizar las direcciones MAC de origen y destino.
- Disminuir el campo de la cabecera IP (TTL).
- Reenviar el empaquete el puerto apropiado.

Nuestro switch tendrá una sola tabla, que el plano de control llenará con reglas estáticas. Cada regla asignará una dirección IP a la dirección MAC y al puerto de salida para el próximo salto. Ya se han definido las reglas del plano de control, por lo que solo necesitamos implementar la lógica del plano de datos de nuestro programa P4 (*basic.p4*).

1.6.1. Comprobar el modelo suministrado

En este punto siguiendo el tutorial, tenemos que comprobar como efectivamente el modelo suministrado no es capaz de establecer comunicaciones entre nodos finales. Esto se debe a que el plano de datos de los switches está incompleto, y será nuestra misión la de completar el plano de datos con nuestro programa en p4 que dotará de la capacidad de fordwarding de paquetes a los switches.

Procedimiento a seguir para llevar a cabo el test:

- Hacemos uso del Makefile que trae el tutorial: **make run**, este target del Makefile automatizará las siguientes tareas:
  - Compilará el archivo *basic.p4* para el behavioral model 2 (bm2) el target simple switch
  - Lanzará Mininet con una topología de tres switch conectados triangularmente y a cada switch se conectará un host. Los host tendrán respectivamente las IPs 10.0.1.1, 10.0.2.2, 10.0.3.3.
- En el mismo directorio se encuentran dos herramientas esritas en Python, servidor y cliente, que nos ayudaran a generar tráfico desde un host a otro. Estas herramientas hacen uso de Scapy para poder visualizar el contenido el paquete generado. Herramientas *receive.py* y *send.py*.
- Levantaremos dos terminales por ejemplo, host 1 y host 2: **xterm h1 h2**
- Ejecutaremos las herramientas para generar tráfico entre ambos host.
- El mensaje no llegará, salimos de Mininet con **exit**, y limpiamos el escenario y Mininet con **make stop** (Este target llamará a **sudo mn -c** por nosotros).

Un programa P4 define una pipe-line de procesamiento de paquetes, pero las reglas dentro de cada tabla se insertan desde el plano de control. Cuando una regla coincide con un paquete (Hay un hit), su acción se invoca con los parámetros proporcionados por el plano de control como parte de la regla.

En este test, ya se ha implementado la lógica del plano de control, está suministrado por el equipo de p4. A la hora de levantar la instancia de Mininet, el comando make run instalará las reglas de procesamiento

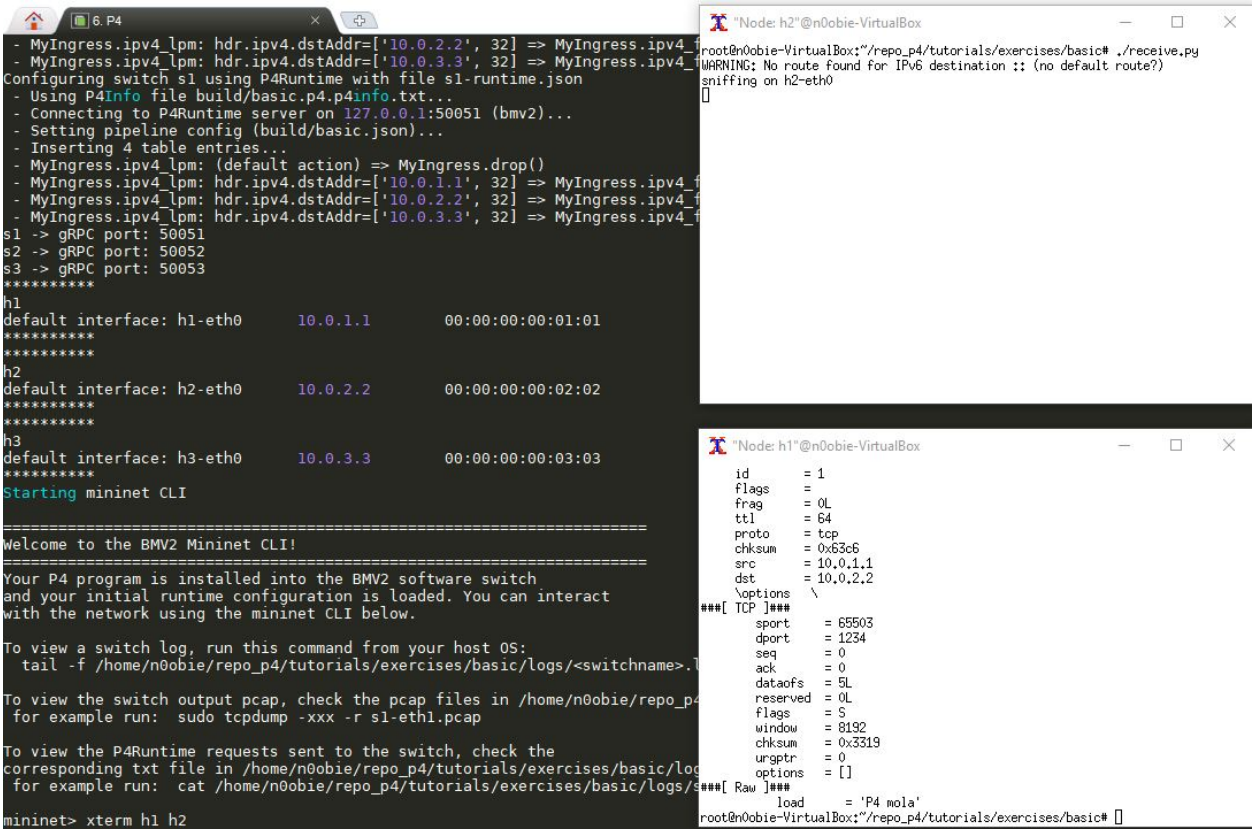

Figura 1.1: Test Mininet: No llega el mensaje.

de paquetes en las tablas de cada switch. Estas se definen en los archivos sX-runtime.json, donde X corresponde al número de switch.

Se hace uso de P4Runtime para instalar las reglas del plano de control. El contenido de los archivos sX-runtime.json se refiere a nombres específicos de tablas, claves y acciones, tal como se define en el archivo P4Info producido por el compilador (busque el archivo build / basic.p4info después de ejecutar make run). Cualquier cambio en el programa P4 que agregue o cambie el nombre de tablas, claves o acciones deberá reflejarse en estos archivos sX-runtime.json.

1.6.2. Desarrollo del fordwarding L3

El archivo basic.p4 contiene un programa P4 esqueleto con piezas lógicas clave donde deberemos completar su cuerpo para el correcto funcionamiento del switch. Las partes que debemos completar son las siguientes:

- Completar el Parser para extraer las cabeceras de ethernet e ipv4.
- Completar un action llamado forward_ipv4 que deberá:
  - Establecer el puerto de salida del paquete.
  - Establecer como dirección MAC origen la MAC destino de la trama recibida.
  - Establecer como dirección MAC destino del paquete la MAC del siguiente salto.
  - Decrementar el campo TTL de la cabecera ipv4.
- Completar el campo apply para decidir si aplicar la tabla.

Para completar el parser de entrada del switch hemos definido tres estados, el primer estado, entry-point, llamado start. El segundo parse_ethernet, para extraer la cabecera de Ethernet, y decidir si si entramos a la ultima fase del parser, parse_ipv4, en función del campo etherType. La ultima fase del parser únicamente extrae la cabecera de Ipv4.

```
/*************************************************************************
*********************** P A R S E R  ***********************************
*************************************************************************/

parser MyParser(packet_in packet,
                out headers hdr,
                inout metadata meta,
                inout standard_metadata_t standard_metadata) {

    state start {
        /* TODO: add parser logic */
        transition parse_ethernet;
    }

    state parse_ethernet{
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType){
                0x800: parse_ipv4;
                default: accept;
        }
    }

    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition accept;
    }
}
```

Figura 1.2: Parser.

Hemos definido un action que hacer forwarding a los paquetes que les llega a los switch, deberá especificar un puerto de salida, modificar las MAC para el siguiente hop, y decrementar en uno el campo TTL de la cabecera Ipv4.

```
action ipv4_forward(macAddr_t dstAddr, egressSpec_t port) {
    /* TODO: fill out code in action body */

    /* Establecer puerto de salida del paquete*/
    standard_metadata.egress_spec = port;

    /* Establecer la dirMAC origen la MAC destino de la trama recibida */
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;

    /* Establecer la dirMAC destino la dir MAC del proximo salto */
    hdr.ethernet.dstAddr = dstAddr;

    /* Decrementar el valro del ttl */
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1 ;

}
```

Figura 1.3: Action forwarding.

Según los requerimientos dados, debemos comprobar con anterioridad si existe y es valida la cabecera Ipv4.

```
table ipv4_lpm {
    key = {
        hdr.ipv4.dstAddr: lpm;
    }
    actions = {
        ipv4_forward;
        drop;
        NoAction;
    }
    size = 1024;
    default_action = NoAction();
}

apply {
    /* TODO: fix ingress control logic
     *  - ipv4_lpm should be applied only when IPv4 header is valid
     */
    if(hdr.ipv4.isValid()){
            ipv4_lpm.apply();
    }
}
```

Figura 1.4: Tabla match-action de nuestro switch.

Por último, únicamente debemos especificar en el deparser como queremos serializar las cabeceras, y en que orden. A continuación, se incluirá la carga útil y se conformará el paquete. Por carga útil entendemos toda información no procesada por el parser de entrada por el switch p4.

```
/*************************************************************************
***********************  D E P A R S E R  ********************************
*************************************************************************/

control MyDeparser(packet_out packet, in headers hdr) {
    apply {
        /* TODO: add deparser logic */
        packet.emit(hdr.ethernet);
        packet.emit(hdr.ipv4);
    }
}
```

Figura 1.5: Deparser de nuestro switch.

A continuación, se puede apreciar en la arquitectura conformada en bloques de nuestro switch. Donde, cada bloque va especificando la funcionalidad para la que ha sido programada. Este diseño es muy útil para ver de manera la operativa funcional de nuestro switch.

```
/*************************************************************************
***********************  S W I T C H  ************************************
*************************************************************************/

V1Switch(
MyParser(),
MyVerifyChecksum(),
MyIngress(),
MyEgress(),
MyComputeChecksum(),
MyDeparser()
) main;
```

Figura 1.6: Topología funcional de nuestro switch.

A continuación, se expone como tras implementar los cambios existe conectividad en la topología entre todos los host. Para implementar los cambios antes hemos realizado un **sudo make stop** y un **sudo make clean** para limpiar los archivos residuales de los switch p4 y de Mininet.

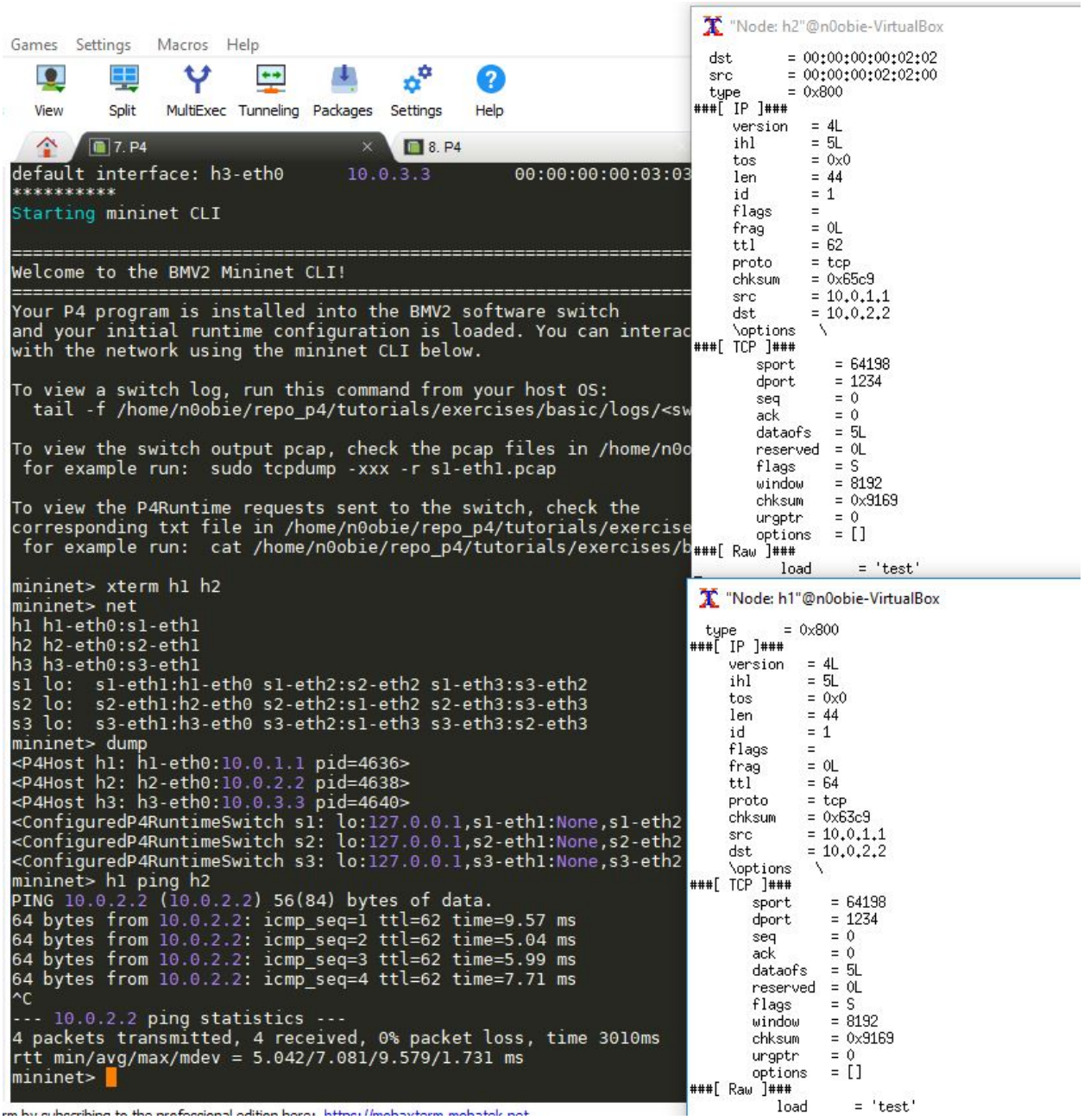

Figura 1.7: Funcionamiento de switch programado.

1.7. Test 2: Implementando el tunelado básico

Bibliografía

[1] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, and P. Castoldi. P4 edge node enabling stateful traffic engineering and cyber security. *IEEE/OSA Journal of Optical Communications and Networking*, 11(1):A84–A95, Jan 2019.

[2] Yi-Bing Linm Shie-Yuan Wang, Chia-Ming Wu and Ching-Chun Huang. High-speed data-plane packet aggregation and disaggregation by P4 switches. *Journal of Network and Computer Applications*, 2019.

[3] M. Uddin, S. Mukherjee, H. Chang, and T. V. Lakshman. Sdn-based multi-protocol edge switching for iot service automation. *IEEE Journal on Selected Areas in Communications*, 36(12):2775–2786, Dec 2018.

Anexos

Apéndice A

Vagrant

Figura A.1: Logo de Vagrant.

A.1. ¿Qué es?

Vagrant es una herramienta que nos ayuda a crear y manejar máquinas virtuales con un mismo entorno de trabajo. Nos permite definir los servicios a instalar así como también sus configuraciones. Está pensado para trabajar en entornos locales y lo podemos utilizar con shell-scripts, Chef, Puppet o Ansible como métodos de provisión de los recursos necesarios en la máquina virtual a desplegar.

Cabe destacar que vagrant no tiene la capacidad para correr una maquina virtual por si mismo, sino que simplemente se encarga de las características con las que debe crearse esa maquina virtual y los complementos a instalar en ella o recursos de aprovisionamiento. Para poder trabajar con las máquinas virtuales es necesario que también tengamos un proveedor de maquinas virtuales, como pueden ser VirtualBox , WMware.

A.2. Instalación

En mi caso hice uso de Vagrant sobre Windows por lo que el método de instalación era descargar un gestor de instalación de ventanas y seguir los pasos establecidos.

- `https://www.vagrantup.com/downloads.html`

Podemos comprobar que la instalación se ha concluido satisfactoriamente haciendo uso de la shell o de la powershell. Escribimos vagrant -v debería mostrarnos la versión de vagrant actual que tenemos montada.

Una vez instalado Vagrant y predispuesto el fichero Vagrantfile únicamente debemos abrir la shell, powershell en Windows. Ir al directorio en cuestión donde tenemos el fichero VagrantFile y hacer **vagrant up**. Este proceso puede tardar más o menos en función de si lo hacemos la primera vez o ya hemos hecho uso de la *box* con la que se va montar la máquina virtual.

A.3. ¿Para que sirve?

Como hemos venido mencionado vagrant sirve para ayudarnos a crear y configurar máquinas virtuales con determinadas características y componentes. La gran ventaja de vagrant es que posee un archivo de configuración **Vagrantfile** donde se centraliza toda la configuración de la VM que creamos además de añadir los métodos de aprovisionamiento de la maquina virtual.

Esto lo puede hacer vía shell-scripts que carga en la maquina virtual a gestionar o con la API que tienen establecida dándonos la posibilidad de describir en el mismo Vagrantfile los scripts deseados. El punto positivo de Vagrant es el hecho de que se puede compartir de una forma relativamente sencilla una maquina virtual con una serie de configuraciones y recursos montados. Esto se puede hacer únicamente compartiendo el archivo de VagrantFile.

A.4. ¿Cómo funciona?

Antes introdujimos el término de *Box*. Vagrant cuando lee el fichero de VagrantFile lee la imagen sobre la que se va a construir la máquina virtual. Esta imagen por así llamarla, es una imagen de un SO con ciertas configuraciones ya predispuestas. Esto ahorrará tiempo en el momento de hacer el boot de la misma. Aquí puede encontrar numerosas Boxes creadas por los usuarios o por propias empresas mantenedoras de ciertos SO's.

- `https://app.vagrantup.com/boxes/search`

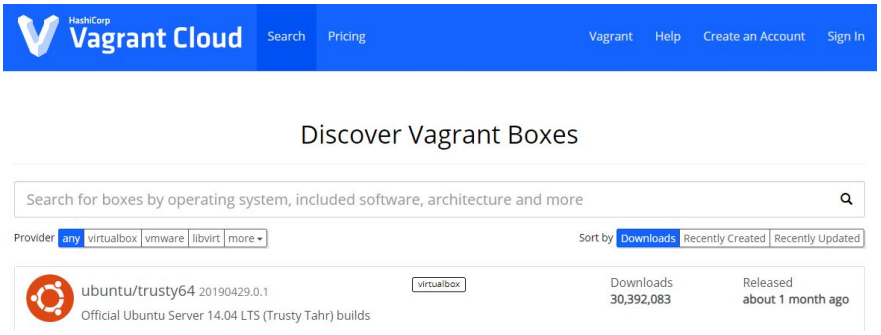

Figura A.2: Repositorio de Boxes.

A.5. Comandos útiles

```
# Arrancar una maquina virtual /  Parar una maquina virtual
vagrant up / vagrant halt
# Conectarse vía ssh
vagrant ssh
# Listar maquinas creadas
vagrant global-status
# Eliminar maquina
vagrant destroy [id]
```