Packet.net Guide

Introduction

Packet.net is a bare metal infrastructure host that's supported by Ansible (>=2.3) via a dynamic inventory script and two cloud modules. The two modules are:

- packet_sshkey: adds a public SSH key from file or value to the Packet infrastructure. Every subsequently-created device will have this public key installed in .ssh/authorized_keys.
- packet_device: manages servers on Packet. You can use this module to create, restart and delete devices.

Note, this guide assumes you are familiar with Ansible and how it works. If you're not, have a look at their :ref:`docs <ansible_documentation>` before getting started.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\scenario_guides\[ansible-devel][docs][docsite][rst][scenario_guides]guide_packet.rst`, line 13); *backlink*
>
> Unknown interpreted text role "ref".

Requirements

The Packet modules and inventory script connect to the Packet API using the packet-python package. You can install it with pip:

```
$ pip install packet-python
```

In order to check the state of devices created by Ansible on Packet, it's a good idea to install one of the Packet CLI clients. Otherwise you can check them via the Packet portal.

To use the modules and inventory script you'll need a Packet API token. You can generate an API token via the Packet portal here. The simplest way to authenticate yourself is to set the Packet API token in an environment variable:

```
$ export PACKET_API_TOKEN=Bfse9F24SFtfs423Gsd3ifGsd43sSdfs
```

If you're not comfortable exporting your API token, you can pass it as a parameter to the modules.

On Packet, devices and reserved IP addresses belong to projects. In order to use the packet_device module, you need to specify the UUID of the project in which you want to create or manage devices. You can find a project's UUID in the Packet portal here (it's just under the project table) or via one of the available CLIs.

If you want to use a new SSH key pair in this tutorial, you can generate it to `./id_rsa` and `./id_rsa.pub` as:

```
$ ssh-keygen -t rsa -f ./id_rsa
```

If you want to use an existing key pair, just copy the private and public key over to the playbook directory.

Device Creation

The following code block is a simple playbook that creates one Type 0 server (the 'plan' parameter). You have to supply 'plan' and 'operating_system'. 'location' defaults to 'ewr1' (Parsippany, NJ). You can find all the possible values for the parameters via a CLI client.

```
# playbook_create.yml

- name: create ubuntu device
  hosts: localhost
  tasks:

  - packet_sshkey:
      key_file: ./id_rsa.pub
      label: tutorial key

  - packet_device:
      project_id: <your_project_id>
      hostnames: myserver
      operating_system: ubuntu_16_04
      plan: baremetal_0
      facility: sjc1
```

After running `ansible-playbook playbook_create.yml`, you should have a server provisioned on Packet. You can verify via a CLI or in the Packet portal.

If you get an error with the message "failed to set machine state present, error: Error 404: Not Found", please verify your project UUID.

Updating Devices

The two parameters used to uniquely identify Packet devices are: "device_ids" and "hostnames". Both parameters accept either a single string (later converted to a one-element list), or a list of strings.

The 'device_ids' and 'hostnames' parameters are mutually exclusive. The following values are all acceptable:

- device_ids: a27b7a83-fc93-435b-a128-47a5b04f2dcf
- hostnames: mydev1
- device_ids: [a27b7a83-fc93-435b-a128-47a5b04f2dcf, 4887130f-0ccd-49a0-99b0-323c1ceb527b]
- hostnames: [mydev1, mydev2]

In addition, hostnames can contain a special '%d' formatter along with a 'count' parameter that lets you easily expand hostnames that follow a simple name and number pattern; in other words, `hostnames: "mydev%d", count: 2` will expand to [mydev1, mydev2].

If your playbook acts on existing Packet devices, you can only pass the 'hostname' and 'device_ids' parameters. The following playbook shows how you can reboot a specific Packet device by setting the 'hostname' parameter:

```
# playbook_reboot.yml

- name: reboot myserver
  hosts: localhost
  tasks:

  - packet device:
      project id: <your project_id>
      hostnames: myserver
      state: rebooted
```

You can also identify specific Packet devices with the 'device_ids' parameter. The device's UUID can be found in the Packet Portal or by using a CLI. The following playbook removes a Packet device using the 'device_ids' field:

```
# playbook_remove.yml

- name: remove a device
  hosts: localhost
  tasks:

  - packet device:
      project id: <your project id>
      device ids: <myserver_device_id>
      state: absent
```

More Complex Playbooks

In this example, we'll create a CoreOS cluster with user data.

The CoreOS cluster will use etcd for discovery of other servers in the cluster. Before provisioning your servers, you'll need to generate a discovery token for your cluster:

```
$ curl -w "\n" 'https://discovery.etcd.io/new?size=3'
```

The following playbook will create an SSH key, 3 Packet servers, and then wait until SSH is ready (or until 5 minutes passed). Make sure to substitute the discovery token URL in 'user_data', and the 'project_id' before running `ansible-playbook`. Also, feel free to change 'plan' and 'facility'.

```
# playbook_coreos.yml

- name: Start 3 CoreOS nodes in Packet and wait until SSH is ready
  hosts: localhost
  tasks:

  - packet_sshkey:
      key_file: ./id_rsa.pub
      label: new

  - packet_device:
      hostnames: [coreos-one, coreos-two, coreos-three]
      operating_system: coreos_beta
      plan: baremetal_0
      facility: ewr1
      project_id: <your_project_id>
      wait_for_public_IPv: 4
      user_data: |
        #cloud-config
        coreos:
          etcd2:
            discovery: https://discovery.etcd.io/<token>
            advertise-client-urls: http://$private_ipv4:2379,http://$private_ipv4:4001
            initial-advertise-peer-urls: http://$private_ipv4:2380
            listen-client-urls: http://0.0.0.0:2379,http://0.0.0.0:4001
            listen-peer-urls: http://$private_ipv4:2380
          fleet:
            public-ip: $private_ipv4
          units:
            - name: etcd2.service
              command: start
            - name: fleet.service
              command: start
    register: newhosts

  - name: wait for ssh
    wait_for:
      delay: 1
      host: "{{ item.public_ipv4 }}"
      port: 22
      state: started
      timeout: 500
    loop: "{{ newhosts.results[0].devices }}"
```

As with most Ansible modules, the default states of the Packet modules are idempotent, meaning the resources in your project will remain the same after re-runs of a playbook. Thus, we can keep the `packet_sshkey` module call in our playbook. If the public key is already in your Packet account, the call will have no effect.

The second module call provisions 3 Packet Type 0 (specified using the 'plan' parameter) servers in the project identified via the 'project_id' parameter. The servers are all provisioned with CoreOS beta (the 'operating_system' parameter) and are customized with cloud-config user data passed to the 'user_data' parameter.

The `packet_device` module has a `wait_for_public_IPv` that is used to specify the version of the IP address to wait for (valid values are `4` or `6` for IPv4 or IPv6). If specified, Ansible will wait until the GET API call for a device contains an Internet-routeable IP address of the specified version. When referring to an IP address of a created device in subsequent module calls, it's wise to use the `wait_for_public_IPv` parameter, or `state: active` in the packet_device module call.

Run the playbook:

```
$ ansible-playbook playbook_coreos.yml
```

Once the playbook quits, your new devices should be reachable via SSH. Try to connect to one and check if etcd has started properly:

```
tomk@work $ ssh -i id_rsa core@$one_of_the_servers_ip
core@coreos-one ~ $ etcdctl cluster-health
```

Once you create a couple of devices, you might appreciate the dynamic inventory script...

Dynamic Inventory Script

The dynamic inventory script queries the Packet API for a list of hosts, and exposes it to Ansible so you can easily identify and act on Packet devices.

You can find it in Ansible Community General Collection's git repo at scripts/inventory/packet_net.py.

The inventory script is configurable through an ini file.

If you want to use the inventory script, you must first export your Packet API token to a PACKET_API_TOKEN environment variable.

You can either copy the inventory and ini config out from the cloned git repo, or you can download it to your working directory like so:

```
$ wget https://raw.githubusercontent.com/ansible-community/contrib-scripts/main/inventory/packet_net.py
$ chmod +x packet net.py
$ wget https://raw.githubusercontent.com/ansible-community/contrib-scripts/main/inventory/packet_net.ini
```

In order to understand what the inventory script gives to Ansible you can run:

```
$ ./packet_net.py --list
```

It should print a JSON document looking similar to following trimmed dictionary:

```
{
  " meta": {
    "hostvars": {
      "147.75.64.169": {
        "packet billing cycle": "hourly",
        "packet created at": "2017-02-09T17:11:26Z",
        "packet facility": "ewr1",
        "packet hostname": "coreos-two",
        "packet href": "/devices/d0ab8972-54a8-4bff-832b-28549d1bec96",
        "packet id": "d0ab8972-54a8-4bff-832b-28549d1bec96",
        "packet locked": false,
        "packet operating system": "coreos_beta",
        "packet plan": "baremetal_0",
        "packet state": "active",
        "packet updated at": "2017-02-09T17:16:35Z",
        "packet user": "core",
        "packet_userdata": "#cloud-config\ncoreos:\n  etcd2:\n    discovery: https://discovery.etcd.io/e0c8a4a9b8fe61acd51ec599e2a4f68e\
      }
    }
```

```
  },
  "baremetal_0": [
    "147.75.202.255",
    "147.75.202.251",
    "147.75.202.249",
    "147.75.64.129",
    "147.75.192.51",
    "147.75.64.169"
  ],
  "coreos_beta": [
    "147.75.202.255",
    "147.75.202.251",
    "147.75.202.249",
    "147.75.64.129",
    "147.75.192.51",
    "147.75.64.169"
  ],
  "ewr1": [
    "147.75.64.129",
    "147.75.192.51",
    "147.75.64.169"
  ],
  "sjc1": [
    "147.75.202.255",
    "147.75.202.251",
    "147.75.202.249"
  ],
  "coreos-two": [
    "147.75.64.169"
  ],
  "d0ab8972-54a8-4bff-832b-28549d1bec96": [
    "147.75.64.169"
  ]
}
```

In the `['_meta']['hostvars']` key, there is a list of devices (uniquely identified by their public IPv4 address) with their parameters. The other keys under `['_meta']` are lists of devices grouped by some parameter. Here, it is type (all devices are of type baremetal_0), operating system, and facility (ewr1 and sjc1).

In addition to the parameter groups, there are also one-item groups with the UUID or hostname of the device.

You can now target groups in playbooks! The following playbook will install a role that supplies resources for an Ansible target into all devices in the "coreos_beta" group:

```
# playbook_bootstrap.yml

- hosts: coreos_beta
  gather_facts: false
  roles:
    - defunctzombie.coreos-boostrap
```

Don't forget to supply the dynamic inventory in the `-i` argument!

```
$ ansible-playbook -u core -i packet_net.py playbook_bootstrap.yml
```

If you have any questions or comments let us know! help@packet.net