» packet_device

Provides a Packet device datasource.

Note: All arguments including the `root_password` and `user_data` will be stored in the raw state as plain-text. Read more about sensitive data in state.

» Example Usage

```
# Fetch a device data by hostname and show it's ID

data "packet_device" "test" {
  project_id       = local.project_id
  hostname         = "mydevice"
}`

output "id" {
  value = data.packet_device.test.id
}
```

```
# Fetch a device data by ID and show its public IPv4

data "packet_device" "test" {

output "ipv4" {
  value = data.packet_device.test.access_public_ipv4
}
```

» Argument Reference

The following arguments are supported:

- `hostname` - The device name
- `project_id` - The id of the project in which the devices exists
- `device_id` - Device ID

User can lookup devices either by `device_id` or `project_id` and `hostname`.

» Attributes Reference

The following attributes are exported:

- `access_private_ipv4` - The ipv4 private IP assigned to the device
- `access_public_ipv4` - The ipv4 management IP assigned to the device
- `access_public_ipv6` - The ipv6 management IP assigned to the device

- `billing_cycle` - The billing cycle of the device (monthly or hourly)
- `facility` - The facility where the device is deployed.
- `description` - Description string for the device
- `hardware_reservation_id` - The id of hardware reservation which this device occupies
- `id` - The ID of the device
- `network` - The device's private and public IP (v4 and v6) network details. When a device is run without any special network configuration, it will have 3 networks:
  - Public IPv4 at `packet_device.name.network.0`
  - IPv6 at `packet_device.name.network.1`
  - Private IPv4 at `packet_device.name.network.2` Elastic addresses then stack by type - an assigned public IPv4 will go after the management public IPv4 (to index 1), and will then shift the indices of the IPv6 and private IPv4. Assigned private IPv4 will go after the management private IPv4 (to the end of the network list). The fields of the network attributes are:
  - `address` - IPv4 or IPv6 address string
  - `cidr` - Bit length of the network mask of the address
  - `gateway` - Address of router
  - `public` - Whether the address is routable from the Internet
  - `family` - IP version - "4" or "6"
- `network_type` - L2 network type of the device, one of "layer3", "layer2-bonded", "layer2-individual", "hybrid"
- `operating_system` - The operating system running on the device
- `plan` - The hardware config of the device
- `ports` - Ports assigned to the device
  - `name` - Name of the port (e.g. `eth0`, or `bond0`)
  - `id` - ID of the port
  - `type` - Type of the port (e.g. `NetworkPort` or `NetworkBondPort`)
  - `mac` - MAC address assigned to the port
  - `bonded` - Whether this port is part of a bond in bonded network setup
- `root_password` - Root password to the server (if still available)
- `ssh_key_ids` - List of IDs of SSH keys deployed in the device, can be both user or project SSH keys
- `state` - The state of the device
- `tags` - Tags attached to the device

» packet_ip_block_ranges

Use this datasource to get CIDR expressions for allocated IP blocks of all the types in a project, optionally filtered by facility.

There are four types of IP blocks in Packet: global IPv4, public IPv4, private IPv4 and IPv6. Both global and public IPv4 are routable from the Internet. Public IPv4 block is allocated in a facility, and addresses from it can only be assigned to devices in that facility. Addresses from Global IPv4 block can be assigned to a device in any facility.

The datasource has 4 list attributes: `global_ipv4`, `public_ipv4`, `private_ipv4` and `ipv6`, each listing CIDR notation (`<network>/<mask>`) of respective blocks from the project.

» Example Usage

```
# List CIDR expressions of all the allocated IP block in you project.

# Declare your project ID
locals {
  project_id = "<UUID_of_your_project>"
}

data "packet_ip_block_ranges" "test" {
  project_id       = local.project_id
}

output "out" {
  value = data.packet_ip_block_ranges.test
}
```

» Argument Reference

- `project_id` - (Required) ID of the project from which to list the blocks.
- `facility` - (Optional) Facility code filtering the IP blocks. Global IPv4 blcoks will be listed anyway. If you omit this, all the block from the project will be listed.

» Attributes Reference

- `global_ipv4` - list of CIDR expressions for Global IPv4 blocks in the project
- `public_ipv4` - list of CIDR expressions for Public IPv4 blocks in the project
- `private_ipv4` - list of CIDR expressions for Private IPv4 blocks in the project
- `ipv6` - list of CIDR expressions for IPv6 blocks in the project

» packet_organization

Provides a Packet organization datasource.

» Example Usage

```
# Fetch a organization data and show projects which belong to it

data "packet_organization" "test" {
  organization_id = local.org_id
}`

output "projects_in_the_org" {
  value = data.packet_organization.test.project_ids
}
```

» Argument Reference

The following arguments are supported:

- `name` - The organization name
- `organization_id` - The UUID of the organization resource

Exactly one of the `name` or `organization_id` must be given.

» Attributes Reference

The following attributes are exported:

- `project_ids` - UUIDs of project resources which belong to this organization
- `description` - Description string
- `website` - Website link
- `twitter` - Twitter handle
- `logo` - Logo URL

» packet_precreated_ip_block

Use this data source to get CIDR expression for precreated IPv6 and IPv4 blocks in Packet. You can then use the cidrsubnet TF builtin function to derive subnets.

» Example Usage

```
# Create device in your project and then assign /64 subnet from precreated block
# to the new device

# Declare your project ID
locals {
  project_id = "<UUID_of_your_project>"
}

resource "packet_device" "web1" {
  hostname         = "web1"
  plan             = "t1.small.x86"
  facilities       = ["ewr1"]
  operating_system = "ubuntu_16_04"
  billing_cycle    = "hourly"
  project_id       = local.project_id

}

data "packet_precreated_ip_block" "test" {
  facility         = "ewr1"
  project_id       = local.project_id
  address_family   = 6
  public           = true
}

# The precreated IPv6 blocks are /56, so to get /64, we specify 8 more bits for network.
# The cirdsubnet interpolation will pick second /64 subnet from the precreated block.

resource "packet_ip_attachment" "from_ipv6_block" {
  device_id = packet_device.web1.id
  cidr_notation = cidrsubnet(data.packet_precreated_ip_block.test.cidr_notation,8,2)
}
```

» Argument Reference

- `project_id` - (Required) ID of the project where the searched block should be.
- `address_family` - (Required) 4 or 6, depending on which block you are looking for.
- `public` - (Required) Whether to look for public or private block.
- `global` - (Optional) Whether to look for global block. Default is false for backward compatibility.

- `facility` - (Optional) Facility of the searched block. (Optional) Only allowed for non-global blocks.

» Attributes Reference

- `cidr_notation` - CIDR notation of the looked up block.

» packet_operating_system

Use this data source to get Packet Operating System image.

» Example Usage

```
data "packet_operating_system" "example" {
  name             = "Container Linux"
  distro           = "coreos"
  version          = "alpha"
  provisionable_on = "c1.small.x86"
}

resource "packet_device" "server" {
  hostname         = "tf.coreos2"
  plan             = "c1.small.x86"
  facilities       = ["ewr1"]
  operating_system = data.packet_operating_system.example.id
  billing_cycle    = "hourly"
  project_id       = local.project_id
}
```

» Argument Reference

- `distro` - (Optional) Name of the OS distribution.
- `name` - (Optional) Name or part of the name of the distribution. Case insensitive.
- `provisionable_on` - (Optional) Plan name.
- `version` - (Optional) Version of the distribution

» Attributes Reference

- `id` - Operating system slug
- `slug` - Operating system slug (same as `id`)

» packet_project

Use this datasource to retrieve attributes of the Project API resource.

» Example Usage

```
# Get Project by name and print UUIDs of its users

data "packet_project" "tf_project_1" {
  name           = "Terraform Fun"
}

output "users_of_Terraform_Fun" {
  value = data.packet_project.tf_project_1.user_ids
}
```

» Argument Reference

The following arguments are supported:

- `name` - The name which is used to look up the project
- `project_id` - The UUID by which to look up the project

» Attributes Reference

The following attributes are exported:

- `payment_method_id` - The UUID of payment method for this project
- `organization_id` - The UUID of this project's parent organization
- `backend_transfer` - Whether Backend Transfer is enabled for this project
- `created` - The timestamp for when the project was created
- `updated` - The timestamp for the last time the project was updated
- `user_ids` - List of UUIDs of user accounts which beling to this project
- `bgp_config` - Optional BGP settings. Refer to Packet guide for BGP.

The `bgp_config` block contains: * `asn` - Autonomous System Numer for local BGP deployment * `md5` - Password for BGP session in plaintext (not a checksum) * `deployment_type` - `private` or `public`, the `private` is likely to be usable immediately, the `public` will need to be review by Packet engineers * `status` - status of BGP configuration in the project * `max_prefix` - The maximum number of route filters allowed per server

» packet_operating_system

Use this data source to get Packet Spot Market Price.

» Example Usage

```
data "packet_spot_market_price" "example" {
  facility = "ewr1"
  plan     = "c1.small.x86"
}
```

» Argument Reference

- `facility` - (Required) Name of the facility.
- `plan` - (Required) Name of the plan.

» Attributes Reference

- `price` - Current spot market price for given plan in given facility.

» packet_spot_market_request

Provides a Packet spot_market_request datasource. The datasource will contain list of device IDs created by referenced Spot Market Request.

» Example Usage

```
# Create a Spot Market Request, and print public IPv4 of the created devices, if any.

resource "packet_spot_market_request" "req" {
  project_id    = local.project_id
  max_bid_price = 0.1
  facilities    = ["ewr1"]
  devices_min   = 2
  devices_max   = 2
  wait_for_devices = true

  instance_parameters {
    hostname         = "testspot"
    billing_cycle    = "hourly"
```

```
    operating_system = "ubuntu_16_04"
    plan             = "t1.small.x86"
  }
}

data "packet_spot_market_request" "dreq" {
  request_id = packet_spot_market_request.req.id
}

output "ids" {
  value = data.packet_spot_market_request.dreq.device_ids
}

data "packet_device" "devs" {
    count = length(data.packet_spot_market_request.dreq.device_ids)
    device_id = data.packet_spot_market_request.dreq.device_ids[count.index]
}

output "ips" {
    value = [for d in data.packet_device.devs: d.access_public_ipv4]
}
```

With the code as `main.tf`, first create the spot market request:

```
$ terraform apply -target packet_spot_market_request.req
```

When the terraform run ends, run a full apply, and the IPv4 addresses will be printed:

```
$ terraform apply

[...]

ips = [
  "947.85.199.231",
  "947.85.194.181",
]
```

» Argument Reference

The following arguments are supported:

- `request_id` - (Required) The id of the Spot Market Request

» Attributes Reference

The following attributes are exported:

- `device_ids` - List of IDs of devices spawned by the referenced Spot Market Request

» packet_volume

Provides a Packet Block Storage Volume datasource to allow you to read existing volumes.

» Example Usage

```
# Read a volume by project ID and name
data "packet_volume" "volume1" {
  name          = "terraform-volume-1"
  project_id    = local.project_id
}

output "volume_size" {
  value = data.packet_volume.volume1.size
}
```

» Argument Reference

The following arguments are supported:

- `volume_id` ID of volume for lookup
- `name` - Name of volume for lookup
- `project_id` - The ID the parent Packet project (for lookup by name)

Either `volume_id` or both `project_id` and `name` must be specified.

» Attributes Reference

The following attributes are exported:

- `id` - The unique ID of the volume
- `name` - The name of the volume
- `project_id` - The project id the volume is in
- `size` - The size in GB of the volume
- `plan` - Performance plan the volume is on

- `billing_cycle` - The billing cycle, defaults to hourly
- `facility` - The facility slug the volume resides in
- `state` - The state of the volume
- `locked` - Whether the volume is locked or not
- `device_ids` - UUIDs of devices to which this volume is attached

» packet_bgp_session

Provides a resource to manage BGP sessions in Packet Host. Refer to Packet BGP documentation for more details.

You need to have BGP config enabled in your project.

BGP session must be linked to a device running BIRD or other BGP routing daemon which will control route advertisements via the session to Packet's upstream routers.

» Example Usage

Following HCL illustrates usage of the BGP features in Packet. It will

- spawn a device in a new BGP-enabled project
- reserve a floating IPv4 address in the project in the same location as the device
- configure the floating IPv4 statically in the device
- install and configure BIRD in the device, and make it announce the floating IPv4 locally

```
locals {
  bgp_password = "955dB0b81Ef"
  project_id   = "<UUID_of_your_project>"
}

# you need to enable BGP config for the project. If you decide to create new
# project, you can use the bgp_config section to enable BGP.
# resource "packet_project" "test" {
#   name = "testpro"
#   bgp_config {
#      deployment_type = "local"
#      md5 = local.bgp_password
#      asn = 65000
#   }
# }

resource "packet_reserved_ip_block" "addr" {
```

```
  project_id = local.project_id
  facility = "ewr1"
  quantity = 1
}

resource "packet_device" "test" {
  hostname         = "terraform-test-bgp-sesh"
  plan             = "t1.small.x86"
  facilities       = ["ewr1"]
  operating_system = "ubuntu_16_04"
  billing_cycle    = "hourly"
  project_id       = local.project_id
}

resource "packet_bgp_session" "test" {
  device_id = packet_device.test.id
  address_family = "ipv4"
}


data "template_file" "interface_lo0" {
  template = <<EOF
auto lo:0
iface lo:0 inet static
   address $${floating_ip}
   netmask $${floating_netmask}
EOF

  vars = {
    floating_ip       = packet_reserved_ip_block.addr.address
    floating_netmask  = packet_reserved_ip_block.addr.netmask
  }
}

data "template_file" "bird_conf_template" {

  template = <<EOF
filter packet_bgp {
    if net = $${floating_ip}/$${floating_cidr} then accept;
}
router id $${private_ipv4};
protocol direct {
    interface "lo";
}
protocol kernel {
    scan time 10;
```

```
    persist;
    import all;
    export all;
}
protocol device {
    scan time 10;
}
protocol bgp {
    export filter packet_bgp;
    local as 65000;
    neighbor $${gateway_ip} as 65530;
    password "$${bgp_password;
}
EOF

  vars = {
    floating_ip    = packet_reserved_ip_block.addr.address
    floating_cidr  = packet_reserved_ip_block.addr.cidr
    private_ipv4   = packet_device.test.network.2.address
    gateway_ip     = packet_device.test.network.2.gateway
    bgp_password   = local.bgp_password
  }
}

resource "null_resource" "configure_bird" {

  connection {
    type = "ssh"
    host = packet_device.test.access_public_ipv4
    private_key = file("/home/tomk/keys/tkarasek_key.pem")
    agent = false
  }

  provisioner "remote-exec" {
    inline = [
      "apt-get install bird",
      "mv /etc/bird/bird.conf /etc/bird/bird.conf.old",
    ]
  }

  triggers = {
    template = data.template_file.bird_conf_template.rendered
    template = data.template_file.interface_lo0.rendered
  }

  provisioner "file" {
```

```
    content     = data.template_file.bird_conf_template.rendered
    destination = "/etc/bird/bird.conf"
  }

  provisioner "file" {
    content     = data.template_file.interface_lo0.rendered
    destination = "/etc/network/interfaces.d/lo0"
  }

  provisioner "remote-exec" {
    inline = [
      "sysctl net.ipv4.ip_forward=1",
      "grep /etc/network/interfaces.d /etc/network/interfaces || echo 'source /etc/network/i
      "ifup lo:0",
      "service bird restart",
    ]
  }
}
```

» Argument Reference

The following arguments are supported:

- `device_id` - (Required) ID of device
- `address_family` - (Required) `ipv4` or `ipv6`
- `default_route` - (Optional) Boolean flag to set the default route policy. False by default.

» Attributes Reference

The following attributes are exported:

- `status`: Status of the session - `up` or `down`

» packet_device

Provides a Packet device resource. This can be used to create, modify, and delete devices.

Note: All arguments including the `root_password` and `user_data` will be stored in the raw state as plain-text. Read more about sensitive data in state.

» Example Usage

Create a device and add it to cool_project

```
resource "packet_device" "web1" {
  hostname         = "tf.coreos2"
  plan             = "t1.small.x86"
  facilities       = ["ewr1"]
  operating_system = "coreos_stable"
  billing_cycle    = "hourly"
  project_id       = local.project_id
}
```

Same as above, but boot via iPXE initially, using the Ignition Provider for provisioning

```
resource "packet_device" "pxe1" {
  hostname         = "tf.coreos2-pxe"
  plan             = "t1.small.x86"
  facilities       = ["ewr1"]
  operating_system = "custom_ipxe"
  billing_cycle    = "hourly"
  project_id       = local.project_id
  ipxe_script_url  = "https://rawgit.com/cloudnativelabs/pxe/master/packet/coreos-stable-pac
  always_pxe       = "false"
  user_data        = data.ignition_config.example.rendered
}
```

Create a device without a public IP address, with only a /30 private IPv4 subnet (4 IP addresses)

```
resource "packet_device" "web1" {
  hostname         = "tf.coreos2"
  plan             = "t1.small.x86"
  facilities       = ["ewr1"]
  operating_system = "coreos_stable"
  billing_cycle    = "hourly"
  project_id       = local.project_id
  ip_address {
      type = "private_ipv4"
      cidr = 30
  }
}
```

Deploy device on next-available reserved hardware and do custom partitioning.

```
resource "packet_device" "web1" {
  hostname         = "tftest"
  plan             = "t1.small.x86"
```

```
  facilities       = ["sjc1"]
  operating_system = "ubuntu_16_04"
  billing_cycle    = "hourly"
  project_id       = local.project_id
  hardware_reservation_id = "next-available"
  storage = <<EOS
{
  "disks": [
    {
      "device": "/dev/sda",
      "wipeTable": true,
      "partitions": [
        {
          "label": "BIOS",
          "number": 1,
          "size": 4096
        },
        {
          "label": "SWAP",
          "number": 2,
          "size": "3993600"
        },
        {
          "label": "ROOT",
          "number": 3,
          "size": 0
        }
      ]
    }
  ],
  "filesystems": [
    {
      "mount": {
        "device": "/dev/sda3",
        "format": "ext4",
        "point": "/",
        "create": {
          "options": [
            "-L",
            "ROOT"
          ]
        }
      }
    },
    {
      "mount": {
```

```
        "device": "/dev/sda2",
        "format": "swap",
        "point": "none",
        "create": {
          "options": [
            "-L",
            "SWAP"
          ]
        }
      }
    }
  ]
}
EOS
}
```

» Argument Reference

The following arguments are supported:

- `hostname` - (Required) The device name
- `project_id` - (Required) The ID of the project in which to create the device
- `operating_system` - (Required) The operating system slug. To find the slug, or visit Operating Systems API docs, set your API auth token in the top of the page and see JSON from the API response.
- `facilities` - List of facility codes with deployment preferences. Packet API will go through the list and will deploy your device to first facility with free capacity. List items must be facility codes or `any` (a wildcard). To find the facility code, visit Facilities API docs, set your API auth token in the top of the page and see JSON from the API response.
- `plan` - (Required) The device plan slug. To find the plan slug, visit Device plans API docs, set your auth token in the top of the page and see JSON from the API response.
- `billing_cycle` - (Required) monthly or hourly
- `user_data` (Optional) - A string of the desired User Data for the device.
- `public_ipv4_subnet_size` (Deprecated) - Size of allocated subnet, more information is in the Custom Subnet Size doc.
- `ipxe_script_url` (Optional) - URL pointing to a hosted iPXE script. More information is in the Custom iPXE doc.
- `always_pxe` (Optional) - If true, a device with OS `custom_ipxe` will continue to boot via iPXE on reboots.
- `hardware_reservation_id` (Optional) - The `full ID` of the hardware reservation where you want this device deployed, or `next-available` if you want to pick your next available reservation automatically. Please be

careful when using hw reservation UUID and `next-available` together for the same pool of resevations. It might happen that the reservation which Packet API will pick as `next-available` is the reservation which you refer with UUID in another packet_device resource. If that happens, and the packet_device with the UUID is created later, resource creation will fail because the reservation is already in use (by the resource created with `next-available`). To workaround this, have the `next-available` resource explicitly depend_on the resource with hw reservation UUID, so that the latter is created first. For more details, see issue #176.

- `storage` (Optional) - JSON for custom partitioning. Only usable on reserved hardware. More information in in the Custom Partitioning and RAID doc.
- `tags` - Tags attached to the device
- `description` - Description string for the device
- `project_ssh_key_ids` - Array of IDs of the project SSH keys which should be added to the device. If you omit this, SSH keys of all the members of the parent project will be added to the device. If you specify this array, only the listed project SSH keys will be added. Project SSH keys can be created with the [packet_project_ssh_key][https://www.terraform.io/docs/providers/packet/r/project_ssh_key.html] resource.
- `network_type` (Optional) - Network type of device, used for Layer 2 networking. Allowed values are `layer3`, `hybrid`, `layer2-individual` and `layer2-bonded`. If you keep it empty, Terraform will not handle the network type of the device.
- `ip_address` (Optional) - A list of IP address types for the device (structure is documented below).
- `wait_for_reservation_deprovision` (Optional) - Only used for devices in reserved hardware. If set, the deletion of this device will block until the hardware reservation is marked provisionable (about 4 minutes in August 2019).
- `force_detach_volumes` (Optional) - Delete device even if it has volumes attached. Only applies for destroy action.

The `ip_address` block has 3 fields:

- `type` - One of [`private_ipv4`, `public_ipv4`, `public_ipv6`]
- `cidr` - CIDR suffix for IP address block to be assigned, i.e. amount of addresses.
- `reservation_ids` - String of UUID of IP block reservations from which the public IPv4 address should be taken.

You can supply one `ip_address` block per IP address type. If you use the `ip_address` you must always pass a block for `private_ipv4`.

To learn more about using the reserved IP addresses for new devices, see the examples in the packet_reserved_ip_block documentation.

» Attributes Reference

The following attributes are exported:

- `access_private_ipv4` - The ipv4 private IP assigned to the device
- `access_public_ipv4` - The ipv4 maintenance IP assigned to the device
- `access_public_ipv6` - The ipv6 maintenance IP assigned to the device
- `billing_cycle` - The billing cycle of the device (monthly or hourly)
- `created` - The timestamp for when the device was created
- `deployed_facility` - The facility where the device is deployed.
- `description` - Description string for the device
- `hardware_reservation_id` - The ID of hardware reservation which this device occupies
- `hostname`- The hostname of the device
- `id` - The ID of the device
- `locked` - Whether the device is locked
- `network` - The device's private and public IP (v4 and v6) network details. When a device is run without any special network configuration, it will have 3 networks:
  - Public IPv4 at `packet_device.name.network.0`
  - IPv6 at `packet_device.name.network.1`
  - Private IPv4 at `packet_device.name.network.2` Elastic addresses then stack by type - an assigned public IPv4 will go after the management public IPv4 (to index 1), and will then shift the indices of the IPv6 and private IPv4. Assigned private IPv4 will go after the management private IPv4 (to the end of the network list). The fields of the network attributes are:
  - `address` - IPv4 or IPv6 address string
  - `cidr` - bit length of the network mask of the address
  - `gateway` - address of router
  - `public` - whether the address is routable from the Internet
  - `family` - IP version - "4" or "6"
- `operating_system` - The operating system running on the device
- `plan` - The hardware config of the device
- `ports` - Ports assigned to the device
  - `name` - Name of the port (e.g. `eth0`, or `bond0`)
  - `id` - ID of the port
  - `type` - Type of the port (e.g. `NetworkPort` or `NetworkBondPort`)
  - `mac` - MAC address assigned to the port
  - `bonded` - Whether this port is part of a bond in bonded network setup
- `project_id`- The ID of the project the device belongs to
- `root_password` - Root password to the server (disabled after 24 hours)
- `ssh_key_ids` - List of IDs of SSH keys deployed in the device, can be both user and project SSH keys
- `state` - The status of the device

- `tags` - Tags attached to the device
- `updated` - The timestamp for the last time the device was updated

» packet_ip_attachment

Provides a resource to attach elastic IP subnets to devices.

To attach an IP subnet from a reserved block to a provisioned device, you must derive a subnet CIDR belonging to one of your reserved blocks in the same project and facility as the target device.

For example, you have reserved IPv4 address block 147.229.10.152/30, you can choose to assign either the whole block as one subnet to a device; or 2 subnets with CIDRs 147.229.10.152/31' and 147.229.10.154/31; or 4 subnets with mask prefix length 32. More about the elastic IP subnets is here.

Device and reserved block must be in the same facility.

» Example Usage

```
# Reserve /30 block of max 2 public IPv4 addresses in Parsippany, NJ (ewr1) for myproject
resource "packet_reserved_ip_block" "myblock" {
  project_id = local.project_id
  facility = "ewr1"
  quantity = 2
}

# Assign /32 subnet (single address) from reserved block to a device
resource "packet_ip_attachment" "first_address_assignment" {
  device_id = packet_device.mydevice.id
  # following expression will result to sth like "147.229.10.152/32"
  cidr_notation = join("/", [cidrhost(packet_reserved_ip_block.myblock.cidr_notation,0), "32
}
```

» Argument Reference

The following arguments are supported:

- `device_id` - (Required) ID of device to which to assign the subnet
- `cidr_notation` - (Required) CIDR notation of subnet from block reserved in the same project and facility as the device

» Attributes Reference

The following attributes are exported:

- `id` - The unique ID of the assignment
- `device_id` - ID of device to which subnet is assigned
- `cidr_notation` - Assigned subnet in CIDR notation, e.g. "147.229.15.30/31"
- `gateway` - IP address of gateway for the subnet
- `network` - Subnet network address
- `netmask` - Subnet mask in decimal notation, e.g. "255.255.255.0"
- `cidr` - length of CIDR prefix of the subnet as integer
- `address_family` - Address family as integer (4 or 6)
- `public` - boolean flag whether subnet is reachable from the Internet

» packet_organization

Provides a resource to manage organization resource in Packet.

» Example Usage

```
# Create a new Project
resource "packet_organization" "tf_organization_1" {
  name = "foobar"
  description = "quux"
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the Organization.
- `description` - Description string.
- `website` - Website link.
- `twitter` - Twitter handle.
- `logo` - Logo URL.

» Attributes Reference

The following attributes are exported:

- `id` - The unique ID of the organization.
- `name` - The name of the Organization.
- `description` - Description string.

- `website` - Website link.
- `twitter` - Twitter handle.
- `logo` - Logo URL.

» packet_port_vlan_attachment

Provides a resource to attach device ports to VLANs.

Device and VLAN must be in the same facility.

If you need this resource to add the port back to bond on removal, set `force_bond = true`.

To learn more about Layer 2 networking in Packet, refer to

- https://www.packet.com/resources/guides/layer-2-configurations/
- https://www.packet.com/developers/docs/network/advanced/layer-2/

» Example Usage

```
# Hybrid network type

resource "packet_vlan" "test" {
  description = "VLAN in New Jersey"
  facility    = "ewr1"
  project_id  = local.project_id
}

resource "packet_device" "test" {
  hostname         = "test"
  plan             = "m1.xlarge.x86"
  facilities       = ["ewr1"]
  operating_system = "ubuntu_16_04"
  billing_cycle    = "hourly"
  project_id       = local.project_id
  network_type     = "hybrid"
}

resource "packet_port_vlan_attachment" "test" {
  device_id = packet_device.test.id
  port_name = "eth1"
  vlan_vnid = packet_vlan.test.vxlan
}
```

```
# Layer 2 network

resource "packet_device" "test" {
  hostname         = "test"
  plan             = "m1.xlarge.x86"
  facilities       = ["ewr1"]
  operating_system = "ubuntu_16_04"
  billing_cycle    = "hourly"
  project_id       = local.project_id
  network_type     = "layer2-individual"
}

resource "packet_vlan" "test1" {
  description = "VLAN in New Jersey"
  facility    = "ewr1"
  project_id  = local.project_id
}

resource "packet_vlan" "test2" {
  description = "VLAN in New Jersey"
  facility    = "ewr1"
  project_id  = local.project_id
}

resource "packet_port_vlan_attachment" "test1" {
  device_id = packet_device.test.id
  vlan_vnid = packet_vlan.test1.vxlan
  port_name = "eth1"
}

resource "packet_port_vlan_attachment" "test2" {
  device_id  = packet_device.test.id
  vlan_vnid  = packet_vlan.test2.vxlan
  port_name  = "eth1"
  native     = true
  depends_on = ["packet_port_vlan_attachment.test1"]
}
```

» Argument Reference

The following arguments are supported:

- `device_id` - (Required) ID of device to be assigned to the VLAN
- `port_name` - (Required) Name of network port to be assigned to the VLAN
- `force_bond` - Add port back to the bond when this resource is removed.

Default is false.

- `vlan_vnid` - VXLAN Network Identifier, integer
- `native` - (Optional) Mark this VLAN a native VLAN on the port. This can be used only if this assignment assigns second or further VLAN to the port. To ensure that this attachment is not first on a port, you can use `depends_on` pointing to another packet_port_vlan_attachment, just like in the layer2-individual example above.

» Attribute Referece

- `id` - UUID of device port used in the assignment
- `vlan_id` - UUID of VLAN API resource
- `port_id` - UUID of device port

» packet_project

Provides a Packet project resource to allow you manage devices in your projects.

» Example Usage

```
# Create a new project
resource "packet_project" "tf_project_1" {
  name           = "Terraform Fun"
}
```

Example with BGP config

```
# Create a new Project
resource "packet_project" "tf_project_1" {
  name           = "tftest"
  bgp_config {
    deployment_type = "local"
    md5 = "C179c28c41a85b"
    asn = 65000
  }
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the project

- `organization_id` - The UUID of organization under which you want to create the project. If you leave it out, the project will be create under your the default organization of your account.
- `payment_method_id` - The UUID of payment method for this project. The payment method and the project need to belong to the same organization (passed with `organization_id`, or default).
- `backend_transfer` - Enable or disable Backend Transfer, default is false
- `bgp_config` - Optional BGP settings. Refer to Packet guide for BGP.

Once you set the BGP config in a project, it can't be removed (due to a limitation in the Packet API). It can be updated.

The `bgp_config` block supports:

- `asn` - Autonomous System Numer for local BGP deployment
- `md5` - (Optional) Password for BGP session in plaintext (not a checksum)
- `deployment_type` - `private` or `public`, the `private` is likely to be usable immediately, the `public` will need to be review by Packet engineers

» Attributes Reference

The following attributes are exported:

- `id` - The unique ID of the project
- `payment_method_id` - The UUID of payment method for this project.
- `organization_id` - The UUID of this project's parent organization.
- `backend_transfer` - Whether Backend Transfer is enabled for this project.
- `created` - The timestamp for when the project was created
- `updated` - The timestamp for the last time the project was updated

The `bgp_config` block additionally exports:

- `status` - status of BGP configuration in the project
- `max_prefix` - The maximum number of route filters allowed per server

» packet_project_ssh_key

Provides a Packet project SSH key resource to manage project-specific SSH keys. Project SSH keys will only be populated onto servers that belong to that project, in contrast to User SSH Keys.

» Example Usage

```
locals {
```

```
  project_id = "<UUID_of_your_project>"
}

resource "packet_project_ssh_key" "test" {
  name       = "test"
  public_key = "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQDM/unxJeFqxsTJcu6mhqsMHSaVlpu+Jj/P+44z
  project_id = local.project_id
}

resource "packet_device" "test" {
  hostname            = "test"
  plan                = "baremetal_0"
  facilities          = ["ewr1"]
  operating_system    = "ubuntu_16_04"
  billing_cycle       = "hourly"
  project_ssh_key_ids = [packet_project_ssh_key.test.id]
  project_id          = local.project_id
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the SSH key for identification
- `public_key` - (Required) The public key. If this is a file, it can be read using the file interpolation function
- `project_id` - (Required) The ID of parent project

» Attributes Reference

The following attributes are exported:

- `id` - The unique ID of the key
- `name` - The name of the SSH key
- `public_key` - The text of the public key
- `project_id` - The ID of parent project
- `owner_id` - The ID of parent project (same as project_id)
- `fingerprint` - The fingerprint of the SSH key
- `created` - The timestamp for when the SSH key was created
- `updated` - The timestamp for the last time the SSH key was updated

» packet_reserved_ip_block

Provides a resource to create and manage blocks of reserved IP addresses in a project.

When a user provisions first device in a facility, Packet API automatically allocates IPv6/56 and private IPv4/25 blocks. The new device then gets IPv6 and private IPv4 addresses from those block. It also gets a public IPv4/31 address. Every new device in the project and facility will automatically get IPv6 and private IPv4 addresses from these pre-allocated blocks. The IPv6 and private IPv4 blocks can't be created, only imported. With this resource, it's possible to create either public IPv4 blocks or global IPv4 blocks.

Public blocks are allocated in a facility. Addresses from public blocks can only be assigned to devices in the facility. Public blocks can have mask from /24 (256 addresses) to /32 (1 address). If you create public block with this resource, you must fill the facility argmument.

Addresses from global blocks can be assigned in any facility. Global blocks can have mask from /30 (4 addresses), to /32 (1 address). If you create global block with this resource, you must specify type = "global_ipv4" and you must omit the facility argument.

Once IP block is allocated or imported, an address from it can be assigned to device with the `packet_ip_attachment` resource.

» Example Usage

Allocate reserved IP blocks:

```
# Allocate /31 block of max 2 public IPv4 addresses in Parsippany, NJ (ewr1) for myproject

resource "packet_reserved_ip_block" "two_elastic_addresses" {
  project_id = local.project_id
  facility = "ewr1"
  quantity = 2
}

# Allocate 1 global floating IP, which can be assigned to device in any facility

resource "packet_reserved_ip_block" "test" {
  project_id = local.project_id
  type     = "global_ipv4"
  quantity = 1
}`
```

Allocate a block and run a device with public IPv4 from the block

```
# Allocate /31 block of max 2 public IPv4 addresses in Parsippany, NJ (ewr1)

resource "packet_reserved_ip_block" "example" {
  project_id = local.project_id
  facility = "ewr1"
  quantity = 2
}

# Run a device with both public IPv4 from the block assigned

resource "packet_device" "nodes" {
  project_id       = local.project_id
  facilities       = ["ewr1"]
  plan             = "t1.small.x86"
  operating_system = "ubuntu_16_04"
  hostname         = "test"
  billing_cycle    = "hourly"
  ip_address {
     type = "public_ipv4"
     cidr = 31
     reservation_ids = [packet_reserved_ip_block.example.id]
  }
  ip_address {
     type = "private_ipv4"
  }
}
```

» Argument Reference

The following arguments are supported:

- `project_id` - (Required) The packet project ID where to allocate the address block
- `quantity` - (Required) The number of allocated /32 addresses, a power of 2
- `type` - (Optional) Either "global_ipv4" or "public_ipv4", defaults to "public_ipv4" for backward compatibility
- `facility` - (Optional) Facility where to allocate the public IP address block, makes sense only for type==public_ipv4, must be empty for type==global_ipv4
- `description` - (Optional) Arbitrary description

» Attributes Reference

The following attributes are exported:

- `facility` - The facility where the block was allocated, empty for global blocks
- `project_id` - To which project the addresses beling
- `quantity` - Number of /32 addresses in the block
- `id` - The unique ID of the block
- `cidr_notation` - Address and mask in CIDR notation, e.g. "147.229.15.30/31"
- `network` - Network IP address portion of the block specification
- `netmask` - Mask in decimal notation, e.g. "255.255.255.0"
- `cidr` - length of CIDR prefix of the block as integer
- `address_family` - Address family as integer (4 or 6)
- `public` - boolean flag whether addresses from a block are public
- `global` - boolean flag whether addresses from a block are global (i.e. can be assigned in any facility)

Idempotent reference to a first /32 address from a reserved block might look like `join("/", [cidrhost(packet_reserved_ip_block.myblock.cidr_notation,0), "32"])`

» packet_ssh_key

Provides a resource to manage User SSH keys on your Packet user account. If you create a new device in a project, all the keys of the project's collaborators will be injected to the device.

The link between User SSH key and device is implicit. If you want to make sure that a key will be copied to a device, you must ensure that the device resource `depends_on` the key resource.

» Example Usage

```
# Create a new SSH key
resource "packet_ssh_key" "key1" {
  name       = "terraform-1"
  public_key = file("/home/terraform/.ssh/id_rsa.pub")
}

# Create new device with "key1" included. The device resource "depends_on" the
# key, in order to make sure the key is created before the device.
resource "packet_device" "test" {
  hostname         = "test-device"
  plan             = "t1.small.x86"
```

```
  facilities       = ["sjc1"]
  operating_system = "ubuntu_16_04"
  billing_cycle    = "hourly"
  project_id       = local.project_id
  depends_on       = ["packet_ssh_key.key1"]
}
```

» Argument Reference

The following arguments are supported:

- `name` - (Required) The name of the SSH key for identification
- `public_key` - (Required) The public key. If this is a file, it can be read using the file interpolation function

» Attributes Reference

The following attributes are exported:

- `id` - The unique ID of the key
- `name` - The name of the SSH key
- `public_key` - The text of the public key
- `fingerprint` - The fingerprint of the SSH key
- `owner_id` - The UUID of the Packet API User who owns this key
- `created` - The timestamp for when the SSH key was created
- `updated` - The timestamp for the last time the SSH key was updated

» packet_spot_market_request

Provides a Packet Spot Market Request resource to allow you to manage spot market requests on your account. For more detail on Spot Market, see this article in Packing documentaion.

» Example Usage

```
# Create a spot market request
resource "packet_spot_market_request" "req" {
  project_id    = local.project_id
  max_bid_price = 0.03
  facilities    = ["ewr1"]
  devices_min   = 1
  devices_max   = 1
```

```
  instance_parameters {
    hostname         = "testspot"
    billing_cycle    = "hourly"
    operating_system = "coreos_stable"
    plan             = "t1.small.x86"
  }
}
```

» Argument Reference

The following arguments are supported:

- `devices_max` - (Required) Maximum number devices to be created
- `devices_min` - (Required) Miniumum number devices to be created
- `max_bid_price` - (Required) Maximum price user is willing to pay per hour per device
- `facilities` - (Required) Facility IDs where devices should be created
- `instance_parameters` - (Required) Device parameters. See device resource for details
- `project_id` - (Required) Project ID
- `wait_for_devices` - (Optional) On resource creation - wait until all desired devices are active, on resource destruction - wait until devices are removed

» Timeouts

The `timeouts` block allows you to specify timeouts for certain actions:

- `create` - (Defaults to 60 mins) Used when creating the Spot Market Request and `wait_for_devices == true`)
- `delete` - (Defaults to 60 mins) Used when destroying the Spot Market Request and `wait for devices == true`

» Attributes Reference

The following attributes are exported:

- `id` - The ID of the Spot Market Request

» packet_vlan

Provides a resource to allow users to manage Virtual Networks in their projects.

To learn more about Layer 2 networking in Packet, refer to * https://www.packet.com/resources/guides/layer-2-configurations/ * https://www.packet.com/developers/docs/network/advanced/layer-2/

» Example Usage

```
# Create a new VLAN in datacenter "ewr1"

resource "packet_vlan" "vlan1" {
  description = "VLAN in New Jersey"
  facility    = "ewr1"
  project_id  = local.project_id
}
```

» Argument Reference

The following arguments are supported:

- `project_id` - (Required) ID of parent project
- `facility` - (Required) Facility where to create the VLAN
- `description` - Description string

» Attributes Reference

The following attributes are exported:

- `vxlan` - VXLAN segment ID

» packet_volume

Provides a Packet Block Storage Volume resource to allow you to manage block volumes on your account. Once created by Terraform, they must then be attached and mounted using the api and `packet_block_attach` and `packet_block_detach` scripts.

» Example Usage

```
# Create a new block volume
resource "packet_volume" "volume1" {
  description   = "terraform-volume-1"
  facility      = "ewr1"
  project_id    = local.project_id
```

```
  plan          = "storage_1"
  size          = 100
  billing_cycle = "hourly"

  snapshot_policies  {
    snapshot_frequency = "1day"
    snapshot_count = 7
  }

  snapshot_policies  {
    snapshot_frequency = "1month"
    snapshot_count = 6
  }
}
```

» Argument Reference

The following arguments are supported:

- `plan` - (Required) The service plan slug of the volume
- `facility` - (Required) The facility to create the volume in
- `project_id` - (Required) The packet project ID to deploy the volume in
- `size` - (Required) The size in GB to make the volume
- `billing_cycle` - The billing cycle, defaults to "hourly"
- `description` - Optional description for the volume
- `snapshot_policies` - Optional list of snapshot policies
- `locked` - Lock or unlock the volume

» Attributes Reference

The following attributes are exported:

- `id` - The unique ID of the volume
- `name` - The name of the volume
- `description` - The description of the volume
- `size` - The size in GB of the volume
- `plan` - Performance plan the volume is on
- `billing_cycle` - The billing cycle, defaults to hourly
- `facility` - The facility slug the volume resides in
- `state` - The state of the volume
- `locked` - Whether the volume is locked or not
- `project_id` - The project id the volume is in
- `attachments` - A list of attachments, each with it's own `href` attribute
- `created` - The timestamp for when the volume was created
- `updated` - The timestamp for the last time the volume was updated

» packet_volume_attachment

Provides attachment of Packet Block Storage Volume to Devices.

Device and volume must be in the same location (facility).

Once attached by Terraform, they must then be mounted using the `packet-block-storage-attach` and `packet-block-storage-detach` scripts, which are presinstalled on most OS images. They can also be found in https://github.com/packethost/packet-block-storage.

» Example Usage

Follwing example will create a device, a volume, and then it will attach the volume to the device over the API.

```
resource "packet_device" "test_device_va" {
  hostname         = "terraform-test-device-va"
  plan             = "t1.small.x86"
  facilities       = ["ewr1"]
  operating_system = "ubuntu_16_04"
  billing_cycle    = "hourly"
  project_id       = local.project_id
}

resource "packet_volume" "test_volume_va" {
  plan = "storage_1"
  billing_cycle = "hourly"
  size = 100
  project_id = local.project_id
  facility = "ewr1"
  snapshot_policies = { snapshot_frequency = "1day", snapshot_count = 7 }
}

resource "packet_volume_attachment" "test_volume_attachment" {
  device_id = packet_device.test_device_va.id
  volume_id = packet_volume.test_volume_va.id
}
```

After applying above hcl, in order to use the volume in the OS of the device, you need to run the attach script. You can run `packet-block-storage-attach` manually over SSH, or you can extend the hcl with following snippet to attach it over remote-exec with Terraform.

```
resource "null_resource" "run_attach_scripts" {
  // re-run the attachment script if any of these resources change
  triggers = {
```

```
    device_id = packet_device.test_device_va.id
    volume_id = packet_volume.test_volume_va.id
  }
  connection {
    type     = "ssh"
    user     = "root"
    private_key = file("/home/user/.ssh/id.dsa")
    host     = packet_device.test_device_va.access_public_ipv4
  }
  provisioner "remote-exec" {
    // run the attach script twice for larger chance of success
    inline = [
            "packet-block-storage-attach",
            "packet-block-storage-attach",
    ]
  }
  depends_on = [packet_volume_attachment.test_volume_attachment]
}
```

» Argument Reference

The following arguments are supported:

- `volume_id` - (Required) The ID of the volume to attach
- `device_id` - (Required) The ID of the device to which the volume should be attached

» Attributes Reference

The following attributes are exported:

- `id` - The unique ID of the volume attachment