» 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"
  facility         = "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"
  facility         = "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_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_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"
  facility         = "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"
  facility         = "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}"
  }

# Deploy device on next-available reserved hardware and do custom partitioning.
resource "packet_device" "web1" {
  hostname         = "tftest"
  plan             = "t1.small.x86"
  facility         = "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.
- `facility` - (Deprecated) The facility in which to create the device.
- `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` (Optional) - 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 id of hardware reservation where you want this device deployed, or `next-available` if you want to pick your next available reservation automatically.
- `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][packet_project_ssh_key.html] resource.
- `network_type` - Network type of device, used for Layer 2 networking. Allowed values are `layer3`, `hybrid`, `layer2-individual` and `layer2-bonded`. Default is `layer3`.

» Attributes Reference

The following attributes are exported:

- `id` - The ID of the device
- `hostname`- The hostname of the device
- `project_id`- The ID of the project the device belongs to
- `facility` - The facility where the device is deployed.
- `plan` - The hardware config 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"
- `access_public_ipv6` - The ipv6 maintenance IP assigned to the device
- `access_public_ipv4` - The ipv4 maintenance IP assigned to the device
- `access_private_ipv4` - The ipv4 private IP assigned to the device
- `locked` - Whether the device is locked
- `billing_cycle` - The billing cycle of the device (monthly or hourly)
- `operating_system` - The operating system running on the device
- `state` - The status of the device
- `created` - The timestamp for when the device was created
- `updated` - The timestamp for the last time the device was updated
- `tags` - Tags attached to the device
- `description` - Description string for the device

- `hardware_reservation_id` - The id of hardware reservation which this device occupies
- `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

» 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_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_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"
  facility         = "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
- `created` - The timestamp for when the SSH key was created
- `updated` - The timestamp for the last time the SSH key was updated

» packet_project_ssh_key

Provides a Packet project SSH key resource to manage project-specific SSH keys. On contrary to user SSH keys, project SSH keys are used to exclusively populate `authorized_keys` in new devices.

If you supply a list of project SSH keys when creating a new device, only the listed keys are used; user SSH keys are ignored.

» 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"
  facility            = "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
- `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_spot_market_request

Provides a Packet Spot Market Request resource to allow you to manage spot market requests on your account. https://support.packet.com/kb/articles/spot-market

» 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_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_attach` and `packet_block_detach` scripts.

» Example Usage

```
resource "packet_device" "test_device_va" {
  hostname         = "terraform-test-device-va"
  plan             = "t1.small.x86"
  facility         = "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}"
}
```

» 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