CLC Provider

The clc provider is used to interact with the many resources supported by CenturyLinkCloud. The provider needs to be configured with account credentials before it can be used.

Use the navigation to the left to read about the available resources.

For additional documentation, see the CLC Developer Center (https://www.ctl.io/developers/)

Example Usage

```
# Configure the CLC Provider
provider "clc" {
  username = "${var.clc_username}"
  password = "${var.clc_password}"
  account  = "${var.clc_account}"  # optional
}

# Create a server
resource "clc_server" "node" {
  # ...
}
```

Account Bootstrap

Trial accounts are available by signing up on the control portal https://control.ctl.io (https://control.ctl.io).

For new accounts, you should initially run these steps manually:

- Create a network. (https://control.ctl.io/Network/network)
- Provision a server. (https://control.ctl.io/create)

Argument Reference

The following arguments are supported:

- `clc_username` - (Required) This is the CLC account username. It must be provided, but it can also be sourced from the `CLC_USERNAME` environment variable.
- `clc_password` - (Required) This is the CLC account password. It must be provided, but it can also be sourced from the `CLC_PASSWORD` environment variable.
- `clc_account` - (Optional) Override CLC account alias. Also taken from the `CLC_ACCOUNT` environment variable if provided.

clc_group

Manages a CLC server group. Either provisions or resolves to an existing group.

See also Complete API documentation (https://www.ctl.io/api-docs/v2/#groups).

Example Usage

```
# Provision/Resolve a server group
resource "clc_group" "frontends" {
  location_id = "WA1"
  name        = "frontends"
  parent      = "Default Group"
}

output "group_id" {
  value = "clc_group.frontends.id"
}
```

Argument Reference

The following arguments are supported:

- `name` - (Required, string) The name (or GUID) of this server group. Will resolve to existing if present.
- `parent` - (Required, string) The name or ID of the parent group. Will error if absent or unable to resolve.
- `location_id` - (Required, string) The datacenter location of both parent group and this group. Examples: "WA1", "VA1"
- `description` - (Optional, string) Description for server group (visible in control portal only)
- `custom_fields` - (Optional) See CustomFields below for details.

CustomFields

`custom_fields` is a block within the configuration that may be repeated to bind custom fields for a server. CustomFields need be set up in advance. Each `custom_fields` block supports the following:

- `id` - (Required, string) The ID of the custom field to set.
- `value` - (Required, string) The value for the specified field.

clc_load_balancer

Manages a CLC load balancer. Manage connected backends with clc_load_balancer_pool (/docs/providers/clc/r/load_balancer_pool.html)

See also Complete API documentation (https://www.ctl.io/api-docs/v2/#shared-load-balancer).

Example Usage

```
# Provision a load balancer
resource "clc_load_balancer" "api" {
  data_center = "${clc_group.frontends.location_id}"
  name        = "api"
  description = "api load balancer"
  status      = "enabled"
}

output "api_ip" {
  value = "clc_load_balancer.api.ip_address"
}
```

Argument Reference

The following arguments are supported:

- `name` - (Required, string) The name of the load balancer.
- `data_center` - (Required, string) The datacenter location of both parent group and this group.
- `status` - (Required, string) Either "enabled" or "disabled"
- `description` - (Optional, string) Description for server group (visible in control portal only)

clc_load_balancer_pool

Manages a CLC load balancer pool. Manage related frontend with clc_load_balancer (/docs/providers/clc/r/load_balancer.html)

See also Complete API documentation (https://www.ctl.io/api-docs/v2/#shared-load-balancer).

Example Usage

```
# Provision a load balancer pool
resource "clc_load_balancer_pool" "pool" {
  data_center   = "${clc_group.frontends.location_id}"
  load_balancer = "${clc_load_balancer.api.id}"
  method        = "roundRobin"
  persistence   = "standard"
  port          = 80

  nodes {
    status      = "enabled"
    ipAddress   = "${clc_server.node.0.private_ip_address}"
    privatePort = 3000
  }

  nodes {
    status      = "enabled"
    ipAddress   = "${clc_server.node.1.private_ip_address}"
    privatePort = 3000
  }
}

output "pool" {
  value = "${join(" ", clc_load_balancer.pool.nodes)}"
}
```

Argument Reference

The following arguments are supported:

- `load_balancer` - (Required, string) The id of the load balancer.
- `data_center` - (Required, string) The datacenter location for this pool.
- `port` - (Required, int) Either 80 or 443
- `method` - (Optional, string) The configured balancing method. Either "roundRobin" (default) or "leastConnection".
- `persistence` - (Optional, string) The configured persistence method. Either "standard" (default) or "sticky".
- nodes - (Optional) See Nodes below for details.

Nodes

`nodes` is a block within the configuration that may be repeated to specify connected nodes on this pool. Each `nodes` block supports the following:

- `ipAddress` (Required, string) The destination internal ip of pool node.
- `privatePort` (Required, int) The destination port on the pool node.
- `status` (Optional, string) Either "enabled" or "disabled".

clc_public_ip

Manages a CLC public ip (for an existing server).

See also Complete API documentation (https://www.ctl.io/api-docs/v2/#public-ip).

Example Usage

```
# Provision a public ip
resource "clc_public_ip" "backdoor" {
  server_id           = "${clc_server.node.0.id}"
  internal_ip_address = "${clc_server.node.0.private_ip_address}"

  ports {
    protocol = "ICMP"
    port     = -1
  }

  ports {
    protocol = "TCP"
    port     = 22
  }

  ports {
    protocol = "TCP"
    port     = 2000
    port_to  = 9000
  }

  source_restrictions {
    cidr = "85.39.22.15/30"
  }
}

output "ip" {
  value = "clc_public_ip.backdoor.id"
}
```

Argument Reference

The following arguments are supported:

- `server_id` - (Required, string) The name or ID of the server to bind IP to.
- `internal_ip_address` - (Required, string) The internal IP of the NIC to attach to. If not provided, a new internal NIC will be provisioned and used.
- `ports` - (Optional) See Ports below for details.
- `source_restrictions` - (Optional) See SourceRestrictions below for details.

Ports

`ports` is a block within the configuration that may be repeated to specify open ports on the target IP. Each `ports` block supports the following:

- `protocol` (Required, string) One of "tcp", "udp", "icmp".
- `port` (Required, int) The port to open. If defining a range, demarks starting port
- `portTo` (Optional, int) Given a port range, demarks the ending port.

SourceRestrictions

`source_restrictions` is a block within the configuration that may be repeated to restrict ingress traffic on specified CIDR blocks. Each `source_restrictions` block supports the following:

- `cidr` (Required, string) The IP or range of IPs in CIDR notation.

clc_server

Manages a CLC server.

Resources and Documentation:

- Datacenter / Capability Map (https://www.ctl.io/data-centers/)
- Hyperscale (https://www.ctl.io/hyperscale/) and Bare Metal (https://www.ctl.io/bare-metal/) Servers
- REST API (https://www.ctl.io/api-docs/v2/#servers-create-server)

Example Usage

```
# Provision a server
resource "clc_server" "node" {
  name_template    = "trusty"
  source_server_id = "UBUNTU-14-64-TEMPLATE"
  group_id         = "${clc_group.frontends.id}"
  cpu              = 2
  memory_mb        = 2048
  password         = "Green123$"

  additional_disks {
    path    = "/var"
    size_gb = 100
    type    = "partitioned"
  }

  additional_disks {
    size_gb = 10
    type    = "raw"
  }
}

output "server_id" {
  value = "clc_server.node.id"
}
```

Argument Reference

The following arguments are supported:

- `name_template` - (Required, string) The basename of the server. A unique name will be generated by the platform.
- `source_server_id` - (Required, string) The name or ID of the base OS image. Examples: "ubuntu-14-64-template", "rhel-7-64-template", "win2012r2dtc-64"
- `group_id` - (Required, string) The name or ID of the server group to spawn server into.
- `cpu` - (Required, int) The number of virtual cores
- `memory_mb` - (Required, int) Provisioned RAM
- `type` - (Required, string) The virtualization type One of "standard", "hyperscale", "bareMetal"

- `password` - (Optional, string) The root/administrator password. Will be generated by platform if not provided.
- `description` - (Optional, string) Description for server (visible in control portal only)
- `power_state` - (Optional, string) See PowerStates below for details. If absent, defaults to `started`.
- `private_ip_address` - (Optional, string) Set internal IP address. If absent, allocated and assigned from pool.
- `network_id` - (Optional, string) GUID of network to use. (Must be set up in advance from control portal.) When absent, the default network will be used.
- `storage_type` - (Optional, string) Backup and replication strategy for disks. One of "standard", "premium"
- `aa_policy_id` - (Optional, string | hyperscale) Anti-Affinity policy ID
- `configuration_id` - (Optional, string | bareMetal) Hardware configuration ID
- `os_type` - (Optional, string | bareMetal) Operating system to install.
- `additional_disks` - (Optional) See Disks below for details.
- `custom_fields` - (Optional) See CustomFields below for details.
- `metadata` - (Optional) Misc state storage for non-CLC metadata.

Server Types

standard

Cloud servers `standard` offer basic, commodity level performance with mixed spindle/SSD storage profiles. Additional features storage backups, snapshot/clone/archive, and widespread availability.

hyperscale

Hyperscale `hyperscale` servers offer significantly higher IOPS than standard servers for CPU and IO intensive servers. See the FAQ (https://www.ctl.io/knowledge-base/servers/hyperscale-server-faq/) for more details.

Physical host redundancy can be managed via Anti-Affinity policies (https://www.ctl.io/knowledge-base/servers/centurylink-cloud-anti-affinity-policies/).

bareMetal

Bare metal `bareMetal` offers optimal compute performance and is available in select datacenters in CLC for approved customers. For more info see the FAQ (https://www.ctl.io/knowledge-base/servers/bare-metal-faq/).

For `bareMetal`, the required fields `source_server_id`, `cpu`, and `memory_mb` are ignored and instead the following fields are required:

- configuration_id
- os_type

Values for `configuration_id` and `os_type` are specific to each datacenter and are available via the API endpoints here (https://www.ctl.io/api-docs/v2/#data-centers-get-data-center-bare-metal-capabilities).

PowerStates

`power_state` may be used to set initial power state or modify existing instances.

- `on` | `started` - machine powered on
- `off` | `stopped` - machine powered off forcefully
- `paused` - freeze machine: memory, processes, billing, monitoring.
- `shutdown` - shutdown gracefully
- `reboot` - restart gracefully
- `reset` - restart forcefully

Disks

`additional_disks` is a block within the configuration that may be repeated to specify the attached disks on a server. Each `additional_disks` block supports the following:

- `type` - (Required, string) Either "raw" or "partitioned".
- `size_gb` - (Required, int) Size of allocated disk.
- `path` - (Required, string, type:`partitioned`) The mountpoint for the disk.

CustomFields

`custom_fields` is a block within the configuration that may be repeated to bind custom fields for a server. CustomFields need be set up in advance. Each `custom_fields` block supports the following:

- `id` - (Required, string) The ID of the custom field to set.
- `value` - (Required, string) The value for the specified field.

Packages

`packages` is a block within the configuration that may be repeated to specify packages and their associated parameters to be run at instantiation. Packages facilitate various tasks like ssh key installation, kernel upgrades, etc. Package ID as well as parameters are configured via this block.

Example:

```
# Configure the CLC Provider
provider "clc_server" "ubuntu" {
  # ...
  packages {
    id     = "77abb844-579d-478d-3955-c69ab4a7ba1a"
    SshKey = "ssh-rsa AAAAB3NzaC1yc2EAAAABIwAA..."
  }
}
```