» datadog_downtime

Provides a Datadog downtime resource. This can be used to create and manage Datadog downtimes.

» Example Usage

```
# Create a new daily 1700-0900 Datadog downtime
resource "datadog_downtime" "foo" {
  scope = ["*"]
  start = 1483308000
  end   = 1483365600

  recurrence {
    type   = "days"
    period = 1
  }
}
```

» Argument Reference

The following arguments are supported:

- `scope` - (Required) A list of items to apply the downtime to, e.g. host:X
- `active` - (Optional) A flag indicating if the downtime is active now.
- `disabled` - (Optional) A flag indicating if the downtime was disabled.
- `start` - (Optional) POSIX timestamp to start the downtime.
- `end` - (Optional) POSIX timestamp to end the downtime.
- `recurrence` - (Optional) A dictionary to configure the downtime to be recurring.
  - `type` - days, weeks, months, or years
  - `period` - How often to repeat as an integer. For example to repeat every 3 days, select a type of days and a period of 3.
  - `week_days` - (Optional) A list of week days to repeat on. Choose from: Mon, Tue, Wed, Thu, Fri, Sat or Sun. Only applicable when type is weeks. First letter must be capitalized.
  - `until_occurrences` - (Optional) How many times the downtime will be rescheduled. `until_occurrences` and `until_date` are mutually exclusive.
  - `until_date` - (Optional) The date at which the recurrence should end as a POSIX timestamp. `until_occurrences` and `until_date` are mutually exclusive.
- `message` - (Optional) A message to include with notifications for this downtime.

- `monitor_id` - (Optional) Reference to which monitor this downtime is applied. When scheduling downtime for a given monitor, datadog changes `silenced` property of the monitor to match the `end` POSIX timestamp.

» Attributes Reference

The following attributes are exported:

- `id` - ID of the Datadog downtime

» Import

Downtimes can be imported using their numeric ID, e.g.

```
$ terraform import datadog_downtime.bytes_received_localhost 2081
```

» datadog_monitor

Provides a Datadog monitor resource. This can be used to create and manage Datadog monitors.

» Example Usage

```
# Create a new Datadog monitor
resource "datadog_monitor" "foo" {
  name               = "Name for monitor foo"
  type               = "metric alert"
  message            = "Monitor triggered. Notify: @hipchat-channel"
  escalation_message = "Escalation message @pagerduty"

  query = "avg(last_1h):avg:aws.ec2.cpu{environment:foo,host:foo} by {host} > 4"

  thresholds {
    ok                = 0
    warning           = 2
    warning_recovery  = 1
    critical          = 4
    critical_recovery = 3
  }

  notify_no_data    = false
  renotify_interval = 60
  notify_audit = false
  timeout_h    = 60
  include_tags = true

  silenced {
    "*" = 0
  }

  tags = ["foo:bar", "baz"]
}
```

» Argument Reference

The following arguments are supported:

- `type` - (Required) The type of the monitor, chosen from:
  - `metric alert`
  - `service check`
  - `event alert`
  - `query alert`
  - `composite`
- `name` - (Required) Name of Datadog monitor
- `query` - (Required) The monitor query to notify on. Note this is not the same query you see in the UI and the syntax is different depending on the monitor `type`, please see the API Reference for details. **Warning:** `terraform plan` won't perform any validation of the query contents.
- `message` - (Required) A message to include with notifications for this monitor. Email notifications can be sent to specific users by using the same '@username' notation as events.
- `escalation_message` - (Optional) A message to include with a re-notification. Supports the '@username' notification allowed elsewhere.
- `thresholds` - (Optional)
  - Metric alerts: A dictionary of thresholds by threshold type. Currently we have four threshold types for metric alerts: critical, critical recovery, warning, and warning recovery. Critical is defined in the query, but can also be specified in this option. Warning and recovery thresholds can only be specified using the thresholds option. Example usage: `thresholds {     critical          = 90     critical_recovery = 85     warning           = 80     warning_recovery  = 75 }` **Warning:** the `critical` threshold value must match the one contained in the `query` argument. The

`threshold` from the previous example is valid along with a query like `avg(last_1h):avg:system.disk.in_use{role:sqlserver} by {host} > 90` but along with something like `avg(last_1h):avg:system.disk.in_use{role:sqlser` `by {host} > 95` would make the Datadog API return a HTTP error 400, complaining "The value provided for parameter 'query' is invalid".

  – Service checks: A dictionary of thresholds by status. Because service checks can have multiple thresholds, we don't define them directly in the query. Default values: `thresholds { ok = 1 critical = 1 warning = 1 unknown = 1 }`

- `notify_no_data` (Optional) A boolean indicating whether this monitor will notify when data stops reporting. Defaults to false.
- `new_host_delay` (Optional) Time (in seconds) to allow a host to boot and applications to fully start before starting the evaluation of monitor results. Should be a non negative integer. Defaults to 300.
- `evaluation_delay` (Optional) Time (in seconds) to delay evaluation, as a non-negative integer. For example, if the value is set to 300 (5min), the timeframe is set to last_5m and the time is 7:00, the monitor will evaluate data from 6:50 to 6:55. This is useful for AWS CloudWatch and other backfilled metrics to ensure the monitor will always have data during evaluation.
- `no_data_timeframe` (Optional) The number of minutes before a monitor will notify when data stops reporting. Must be at least 2x the monitor timeframe for metric alerts or 2 minutes for service checks. Default: 2x timeframe for metric alerts, 2 minutes for service checks.
- `renotify_interval` (Optional) The number of minutes after the last notification before a monitor will re-notify on the current status. It will only re-notify if it's not resolved.
- `notify_audit` (Optional) A boolean indicating whether tagged users will be notified on changes to this monitor. Defaults to false.
- `timeout_h` (Optional) The number of hours of the monitor not reporting data before it will automatically resolve from a triggered state. Defaults to false.
- `include_tags` (Optional) A boolean indicating whether notifications from this monitor will automatically insert its triggering tags into the title. Defaults to true.
- `require_full_window` (Optional) A boolean indicating whether this monitor needs a full window of data before it's evaluated. We highly recommend you set this to False for sparse metrics, otherwise some evaluations will be skipped. Default: True for "on average", "at all times" and "in total" aggregation. False otherwise.

- `locked` (Optional) A boolean indicating whether changes to to this monitor should be restricted to the creator or admins. Defaults to False.
- `tags` (Optional) A list of tags to associate with your monitor. This can help you categorize and filter monitors in the manage monitors page of the UI. Note: it's not currently possible to filter by these tags when querying via the API
- `silenced` (Optional) Each scope will be muted until the given POSIX timestamp or forever if the value is 0. To mute the alert completely:

  ```
  silenced {
    "*" =  0
  }
  ```

  To mute role:db for a short time:

  ```
  silenced {
    "role:db" = 1412798116
  }
  ```

» Attributes Reference

The following attributes are exported:

- `id` - ID of the Datadog monitor

» Import

Monitors can be imported using their numeric ID, e.g.

```
$ terraform import datadog_monitor.bytes_received_localhost 2081
```

» datadog_timeboard

Provides a Datadog timeboard resource. This can be used to create and manage Datadog timeboards.

» Example Usage

```
# Create a new Datadog timeboard
resource "datadog_timeboard" "redis" {
  title       = "Redis Timeboard (created via Terraform)"
  description = "created using the Datadog provider in Terraform"
  read_only   = true
```

```
  graph {
    title = "Redis latency (ms)"
    viz   = "timeseries"

    request {
      q    = "avg:redis.info.latency_ms{$host}"
      type = "bars"
    }
  }

  graph {
    title = "Redis memory usage"
    viz   = "timeseries"

    request {
      q       = "avg:redis.mem.used{$host} - avg:redis.mem.lua{$host}, avg:redis.mem.lua{$ho
      stacked = true
    }

    request {
      q = "avg:redis.mem.rss{$host}"

      style {
        palette = "warm"
      }
    }
  }

  graph {
    title = "Top System CPU by Docker container"
    viz   = "toplist"

    request {
      q = "top(avg:docker.cpu.system{*} by {container_name}, 10, 'mean', 'desc')"
    }
  }

  template_variable {
    name   = "host"
    prefix = "host"
  }
}
```

» Argument Reference

The following arguments are supported:

- `title` - (Required) The name of the dashboard.
- `description` - (Required) A description of the dashboard's content.
- `read_only` - (Optional) The read-only status of the timeboard. Default is false.
- `graph` - (Required) Nested block describing a graph definition. The structure of this block is described below. Multiple graph blocks are allowed within a datadog_timeboard resource.
- `template_variable` - (Optional) Nested block describing a template variable. The structure of this block is described below. Multiple template_variable blocks are allowed within a datadog_timeboard resource.

» Nested `graph` blocks

Nested `graph` blocks have the following structure:

- `title` - (Required) The name of the graph.
- `viz` - (Required) The type of visualization to use for the graph. Valid choices are "change", "distribution", "heatmap", "hostmap", "query_value", timeseries", and "toplist".
- `request` - Nested block describing a graph definition request (a metric query to plot on the graph). The structure of this block is described below. Multiple request blocks are allowed within a graph block.
- `events` - (Optional) A list of event filter strings. Note that, while supported by the Datadog API, the Datadog UI does not (currently) support multiple event filters very well, so use at your own risk.
- `autoscale` - (Optional) Boolean that determines whether to autoscale graphs.
- `precision` - (Optional) Number of digits displayed, use `*` for full precision.
- `custom_unit` - (Optional) Display a custom unit on the graph (such as 'hertz')
- `text_align` - (Optional) How to align text in the graph, can be one of 'left', 'center', or 'right'.
- `style` - (Optional) Nested block describing hostmaps. The structure of this block is described below.
- `group` - (Optional) List of groups for hostmaps (shown as 'group by' in the UI).
- `include_no_metric_hosts` - (Optional) If set to true, will display hosts on hostmap that have no reported metrics.
- `include_ungrouped_hosts` - (Optional) If set to true, will display hosts without groups on hostmaps.
- `scope` - (Optional) List of scopes for hostmaps (shown as 'filter by' in the UI).

- `yaxis` - (Optional) Nested block describing modifications to the yaxis rendering. The structure of this block is described below.
- `marker` - (Optional) Nested block describing lines / ranges added to graph for formatting. The structure of this block is described below. Multiple marker blocks are allowed within a graph block.

» Nested `graph marker` blocks

Nested `graph marker` blocks have the following structure:

- `type` - (Required) How the marker lines will look. Possible values are {"error", "warning", "info", "ok"} {"dashed", "solid", "bold"}. Example: "error dashed".
- `value` - (Required) Mathematical expression describing the marker. Examples: "y > 1", "-5 < y < 0", "y = 19".
- `label` - (Optional) A label for the line or range. **Warning:** when a label is enabled but left empty through the UI, the Datadog API returns a boolean value, not a string. This makes `terraform plan` fail with a JSON decoding error.

» Nested `graph yaxis` block

- `min` - (Optional) Minimum bound for the graph's yaxis, a string.
- `max` - (Optional) Maximum bound for the graph's yaxis, a string.
- `scale` - (Optional) How to scale the yaxis. Possible values are: "linear", "log", "sqrt", "pow##" (eg. pow2, pow0.5, 2 is used if only "pow" was provided). Default: "linear".

» Nested `graph request` blocks

Nested `graph request` blocks have the following structure:

- `q` - (Required) The query of the request. Pro tip: Use the JSON tab inside the Datadog UI to help build you query strings.
- `aggregator` - (Optional) The aggregation method used when the number of data points outnumbers the max that can be shown.
- `stacked` - (Optional) Boolean value to determine if this is this a stacked area graph. Default: false (line chart).
- `type` - (Optional) Choose how to draw the graph. For example: "line", "bar" or "area". Default: "line".
- `style` - (Optional) Nested block to customize the graph style.
- `conditional_format` - (Optional) Nested block to customize the graph style if certain conditions are met. Currently only applies to `Query Value` and `Top List` type graphs.

» Nested `graph` `style` block

The nested `style` block is used specifically for styling `hostmap` graphs, and has the following structure:

- `palette` - (Optional) Spectrum of colors to use when styling a hostmap. For example: "green_to_orange", "yellow_to_green", "YlOrRd", or "hostmap_blues". Default: "green_to_orange".
- `palette_flip` - (Optional) Flip how the hostmap is rendered. For example, with the default palette, low values are represented as green, with high values as orange. If palette_flip is "true", then low values will be orange, and high values will be green.

» Nested `graph` `request` `style` block

The nested `style` blocks has the following structure:

- `palette` - (Optional) Color of the line drawn. For example: "classic", "cool", "warm", "purple", "orange" or "gray". Default: "classic".
- `width` - (Optional) Line width. Possible values: "thin", "normal", "thick". Default: "normal".
- `type` - (Optional) Type of line drawn. Possible values: "dashed", "solid", "dotted". Default: "solid".

» Nested `graph` `request` `conditional_format` block

The nested `conditional_format` blocks has the following structure:

- `palette` - (Optional) Color scheme to be used if the condition is met. For example: "red_on_white", "white_on_red", "yellow_on_white", "white_on_yellow", "green_on_white", "white_on_green", "gray_on_white", "white_on_gray", "custom_text", "custom_bg", "custom_image".
- `comparator` - (Required) Comparison operator. Example: ">", "<".
- `value` - (Optional) Value that is the threshold for the conditional format.
- `custom_fg_color` - (Optional) Used when `palette` is set to `custom_text`. Set the color of the text to a custom web color, such as "#205081".
- `custom_bg_color` - (Optional) Used when `palette` is set to `custom_bg`. Set the color of the background to a custom web color, such as "#205081".

» Nested `template_variable` blocks

Nested `template_variable` blocks have the following structure:

- `name` - (Required) The variable name. Can be referenced as $name in `graph` `request` `q` query strings.

- `prefix` - (Optional) The tag group. Default: no tag group.
- `default` - (Optional) The default tag. Default: ”*” (match all).

» Attributes Reference

The following attributes are exported:

- `id` - The unique ID of this timeboard in your Datadog account. The web interface URL to this timeboard can be generated by appending this ID to `https://app.datadoghq.com/dash/`

» Import

Timeboards can be imported using their numeric ID, e.g.

```
$ terraform import datadog_timeboard.my_service_timeboard 2081
```

» datadog_user

Provides a Datadog user resource. This can be used to create and manage Datadog users.

» Example Usage

```
# Create a new Datadog user
resource "datadog_user" "foo" {
  email  = "new@example.com"
  handle = "new@example.com"
  name   = "New User"
}
```

» Argument Reference

The following arguments are supported:

- `disabled` - (Optional) Whether the user is disabled
- `email` - (Required) Email address for user
- `handle` - (Required) The user handle, must be a valid email.
- `is_admin` - (Optional) Whether the user is an administrator
- `name` - (Required) Name for user

- `role` - (Deprecated) Role description for user. **Warning**: the corresponding query parameter is ignored by the Datadog API, thus the argument would always trigger an execution plan.

» Attributes Reference

The following attributes are exported:

- `disabled` - Returns true if Datadog user is disabled (NOTE: Datadog does not actually delete users so this will be true for those as well)
- `id` - ID of the Datadog user
- `verified` - Returns true if Datadog user is verified

» Import

users can be imported using their handle, e.g.

```
$ terraform import datadog_user.example_user existing@example.com
```