

Terraform Enterprise API Guide

Terraform Enterprise

The Public Terraform Enterprise as well as Private Terraform Enterprise (PTFE) is a tool for building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing and popular service providers (e.g. AWS, Azure, Google) as well as custom in-house solutions. Configuration files describe to Terraform the components needed to run a single application or your entire datacenter. These datacenters can be “Brick and Mortar” or “Cloud Native” installations.

The Terraform Enterprise (TFE) Version is GUI based and provides many collaboration features for teams as well as Enterprise features for public companies. This guide will focus on The Terraform Enterprise API, that can be used with both the Terraform Public SAAS platform, as well as the Private Enterprise Version that can be installed on Air-gapped or Hybrid datacenters.

API calls are general purpose, these can be used for Infrastructure as Code (IaC) configuration as well as other DevOps tasks that include CI/CD pipelines that make use of curl-based REST calls. Platforms like Jenkins, Bamboo, or Spinnaker allow for 3rd party REST integration that will be discussed in this guide.

Design Differences

Our major job as Solution Engineers is to eliminate objections of clients. The API's of the TFE:

1. Allows existing tooling to manage the VCS
2. Allows for integration of unsupported VCS

With an API based workflow, workspaces *are not* directly associated with a VCS repo, and runs *are not* driven by webhooks on your VCS provider. This allows a client's other tooling to drive and queue when a run should occur. Usually this is a CI system, or tooling capable of monitoring changes to the Terraform code and performing actions in response.

Once a 3rd party system has decided a Terraform run should occur, it must make a series of calls to TFE's variables, runs and configuration-versions APIs to upload configuration files and perform a run with them.

The most significant difference in this workflow is that TFE does not fetch configuration files from version control. Instead, your own tooling must upload the configurations as a compressed archive (.tar.gz file). This allows clients to work with configurations from

unsupported version control systems, automatically generate Terraform configurations from some other source of data or build a variety of other integrations.

Assumptions

This document assumes there is at least one TFE Organization set-up. This document also assumes the usage of Postman. This tool is not at all required to use the TFE API but is used for confidence in learning the API calls before integrating into your environment. Your effort will not be wasted using Postman, it has several Code Generation features that can be used in your Clients respective CI/CD implementation.

Terraform API Background

The API design contains ideas from the “JSON Hypertext Application Language (HAL) specification as well as design patterns from “Hypertext as the Engine of Application State” (HATEOS). For most Terraform Enterprise (TFE) resources, CRUD operations are implemented on each resource supported by the API.

CRUD Operation	REST Verb
Create	Post
Read	Get
Update	Patch
Delete	Delete

This allows for full lifecycle operations from creation to destruction of any TFE resource. The TFE API site can be found at: <https://www.terraform.io/docs/enterprise/api/index.html>.

Authentication

The API requires a bearer token be used in Authorization headers on every TFE API invocation. This is what you need to do in the API to bootstrap usage by the API. All TFE REST interfaces make use of this security mechanism. For the purposes of this Guide, a User token will be used. In order to generate a token in TFE:

Navigate to Settings | Tokens

1: click on Generate Token

The image displays two screenshots of the TFE Tokens management interface. The top screenshot shows the 'Generate new token' form. The 'DESCRIPTION' field contains 'Token for TFE Guide'. A blue circle with the number '1' is positioned next to the 'Generate token' button. The bottom screenshot shows the same page after the token has been generated. A green success message is displayed: 'Success: Your authentication token is displayed below. Treat this token like a password, as it can be used to access your account without a username, password, or two-factor authentication.' Below the message, the token is displayed as 'QUuoEjw3ddy' followed by 'pzkc'. A blue circle with the number '2' is next to the 'Copy' button, and a blue circle with the number '3' is next to the 'Download' button.

2: Copy the token, for use in Postman

3: Download a copy to retain in 1Password or other form of personal identity Vault

Most applications will be curl based. The token, as one suggestion, can be configured as an Environment Variable. Just remember, use double quotes, opposed to single quotes – by far the biggest mistake with BASH scripting.

Settings / Tokens

USER SETTINGS

- Profile
- Password
- Two Factor Authentication
- Tokens**

Tokens



Generate new token

DESCRIPTION

Generate token

Use this description to help identify the use of the token in the future.

Tokens (2)

Token for TFE Guide	Created 4 minutes ago	 4
John Dohoney's TFE Access Token	Created 15 days ago	

If steps 2 and 3 were not done properly, the system only allows for the destruction (4) of the token, and a new one must be generated.

Postman

Postman ¹ is a general-purpose REST Client that is both a stand-alone application and Chrome Plugin. Variables can be defined for variable portions of your REST invocation. Here is an example of the GET on a specific Organization:

Organizations (Get)

1 2

GET `https://(API-BASE)/api/v2/organizations/(ORG)` Params Send Save

Authorization Headers (2) Body Pre-request Script Tests Code

Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/> Authorization	Bearer (OAUTH-TOKEN)	Not optional			
<input checked="" type="checkbox"/> Content-Type	application/vnd.api+json				
New key	Value	Description			

Body Cookies Headers (16) Test Results Status: 200 OK Time: 1554 ms

3

¹ Postman Tutorial Series on YouTube:

- <https://www.youtube.com/watch?v=juldrxDrSH0>
- <https://www.youtube.com/watch?v=hHV00Za4zrQ>
- https://www.youtube.com/watch?v=cR_FqveTewo

- 1: API-BASE is the stem of the proxy path that can be your Private TFE or the Hashicorp SAAS TFE
- 2: ORG is the organization I want to find out details on
- 3: Is the token we generated in the previous step as a Bearer Token in an Authorization header.

These are maintained in the “manage environments” page (Click on the Gear in the (near) top right, choose “manage environments” and select the environment in the list to manage. After that you will see the following dialog for variable maintenance (using the Chrome Browser Plugin)

MANAGE ENVIRONMENTS

Manage Environments

Environment Templates

Edit Environment

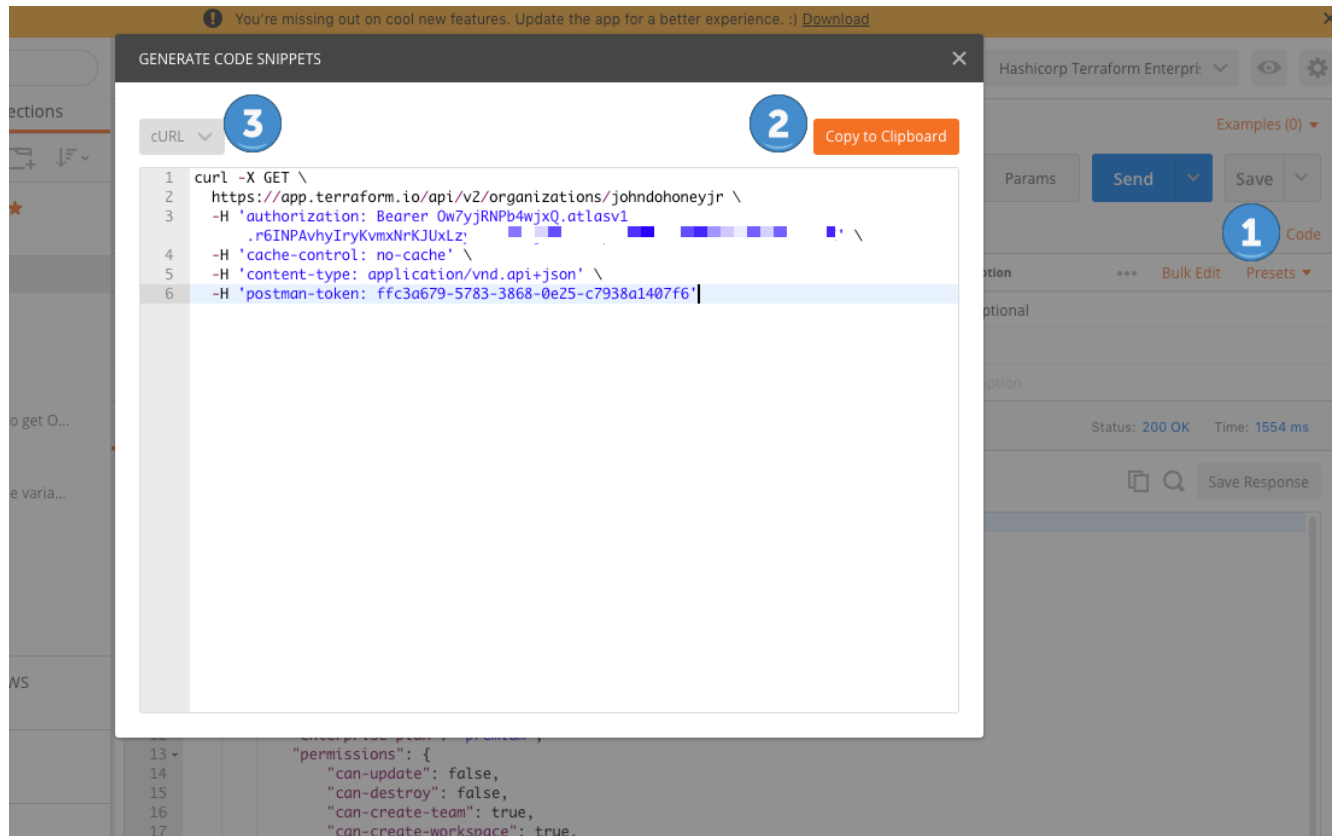
Hashicorp Terraform Enterprise

	Key	Value	Bulk Edit
<input checked="" type="checkbox"/>	API-BASE 1	app.terraform.io	
<input checked="" type="checkbox"/>	ORG 2	johndohoneyjr	
<input checked="" type="checkbox"/>	WORKSPACE	aws-tfe-demo	
<input checked="" type="checkbox"/>	OAuth-TOKEN 3	Ow7yjRNPb4wjxQ.atlasv1.r6INPAvhyIryKvmxNrKJUX...	
<input checked="" type="checkbox"/>	REPLACEWITHORGTODELETED	Replace-Me	
<input checked="" type="checkbox"/>	WORKSPACE-ID	ws-wBnph7GhrYBcrLS	
<input checked="" type="checkbox"/>	RUN-ID	run-11GvUBmF9tYEXqNM	
	New key	Value	

Cancel

Update

Notice the variables that match up to the variable templates in the call parameters. Another handy feature of Postman is the Code generation capability. Click the code link as shown below:



- 1: Click the “code” link (this is grayed out, as this screenshot was taken after code was clicked)
- 2: Copy the code to the Clipboard
- 3: Other Code generation options are available besides Curl

An example of using Go (1) is also shown:

GENERATE CODE SNIPPETS ✕

Go 1 Copy to Clipboard

```
2
3 import (
4     "fmt"
5     "net/http"
6     "io/ioutil"
7 )
8
9 func main() {
10
11     url := "https://app.terraform.io/api/v2/organizations/johndohoneyjr"
12
13     req, _ := http.NewRequest("GET", url, nil)
14
15     req.Header.Add("authorization", "Bearer Ow7yjRNPb4wix0.atlasv1
16     .r6INPAvhyIryKvmxNrKJUxLzyl...")
17     req.Header.Add("content-type", "application/vnd.api+json")
18     req.Header.Add("cache-control", "no-cache")
19     req.Header.Add("postman-token", "216a3e20-1392-7dfd-5649-bd695f7510fc")
20
21     res, _ := http.DefaultClient.Do(req)
22
23     defer res.Body.Close()
24     body, _ := ioutil.ReadAll(res.Body)
25
26     fmt.Println(res)
27     fmt.Println(string(body))
28 }
```

Depending on your tool of choice to implement the integration with TFE, many code generation options exist within Postman.

API Usage

The following is a scenario for this guide that involves:

- Setting up an Organization for Dynamic Continuous Integration (CI)
- Setting up a Workspace
- Population of variables into the Workspace to constrain the Infrastructure
- Upload a configuration to run (recall the API does not use a VCS)
- Queuing of a Run to execute the Terraform in the Configuration to create the infrastructure

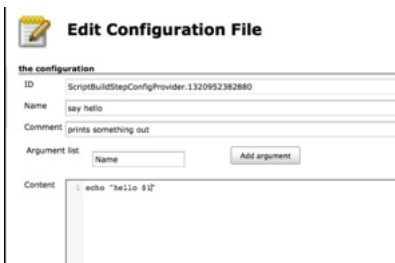
As an example, a dynamic CI Workflow can be orchestrated from Jenkins using the Managed Script plug-in – as one example. Jenkins has many plug-ins that allow for integration with 3rd Party applications. Scripts are one way of invoking commands within CI tools.



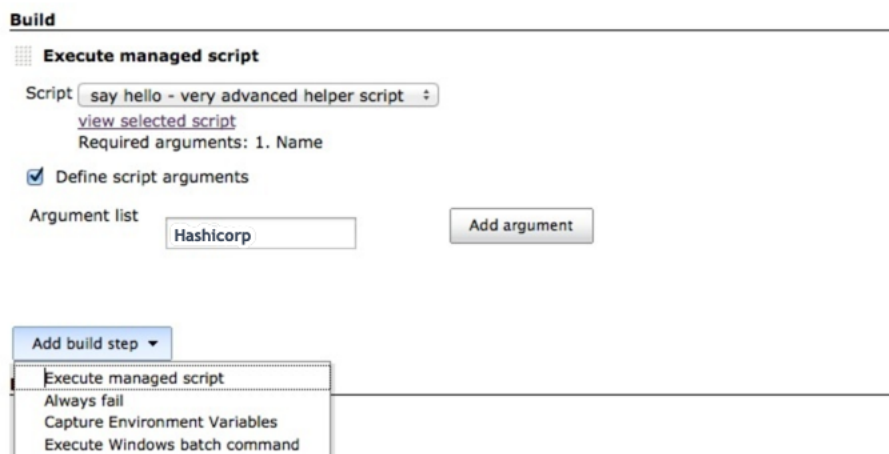
The screenshot shows the Jenkins 'Managed files' page. On the left, there are links for 'Manage Jenkins', 'Config Files', and 'Add a new Config'. Below these is a 'Build Executor Status' table with two rows, both showing 'Idle' status. A 'Logger Console' button is also present. The main area is titled 'Type' and asks the user to 'Select the file type you want to create'. The options are: 'Managed script file - create a managed/reusable script' (selected), 'Maven settings.xml - a settings.xml which can be referenced within Apache Maven jobs', 'Custom file - a custom file (e.g. text or any other not yet available format)', 'Global Maven settings.xml - a global maven settings.xml which can be referenced within Apache Maven jobs', 'Simple XML file - a general xml file', and 'Groovy file - a reusable groovy script'. A 'Submit' button is at the bottom right.

#	Status
1	Idle
2	Idle

This is a simple example that shows building a script. Also, in Jenkins, environment values can be used as well as set. This will need to be done for Terraforms varying ID's that need to be referenced in POST bodies of TFE API calls.



The screenshot shows the 'Edit Configuration File' form. It contains fields for 'ID' (ScriptBuildStepConfigProvider.1320952382880), 'Name' (say hello), 'Comment' (prints something out), and 'Argument list' (Name). There is an 'Add argument' button. The 'Content' field contains the text: `! echo "hello $?"`.



The screenshot shows the Jenkins 'Build' page. It features a section titled 'Execute managed script'. The 'Script' field contains 'say hello - very advanced helper script'. Below it is a link 'view selected script' and the text 'Required arguments: 1. Name'. The 'Define script arguments' checkbox is checked. The 'Argument list' field contains 'Hashicorp', and there is an 'Add argument' button. At the bottom, there is a dropdown menu 'Add build step' with a list of options: 'Execute managed script', 'Always fail', 'Capture Environment Variables', and 'Execute Windows batch command'.

API Usage – Bearer Token

Prior to using the API, we created an Access Token in the TFE GUI. See the Authentication section above→the API requires the ID of the OAUTH token we have previously created. The API Headers makes use of the actual value of the OAUTH token in the Authentication Bearer header; However, the POST body does reference the Token ID. So, the API call to retrieve the ID is necessary

`https://{{API-BASE}}/api/v2/organizations/{{ORG}}/oauth-tokens`

The data[n].id field is the value of interest.



Depending on your method of Automation, this can be saved off as an Environment Variable, and parsed out of the response body with a tool like sed or jq to populate the environment variable.

Here is an example of saving off the OAUTH-ID in an environment variable for future usage using jq². Note the use of double quotes vs single Quotes.

```
#!/bin/bash

export OAUTHID=$(curl --silent -X GET \
  https://app.terraform.io/api/v2/organizations/johndohoneyjr/oauth-tokens \
  -H "authorization: Bearer $BEARER" \
  -H 'content-type: application/vnd.api+json' | \
jq .data[0].id)
echo $OAUTHID
```

² Assuming you are a MAC OSX user, the easiest way to install jq is: **brew install jq**

API Usage – Organizations

Be sure when creating your Organization, that you use a User Token, and that it has admin rights. Otherwise, this operation will fail. Here is a sample curl call that creates an Organization. Check the Postman Collection for “Organizations (POST)”

```
curl -X POST \
  https://app.terraform.io/api/v2/organizations \
  -H 'authorization: Bearer
0aj0JCUYwoAFyw.atlasv1.mgOHjtxvqt71ttmzsDqXg3OvBRLsBnKI7ej0PJfEWbDliVqQWIELc5azF8UxZqm71Vw' \
  -H 'content-type: application/vnd.api+json' \
  -d '{
    "data": {
      "attributes": {
        "enterprise-plan": null,
        "trial-expires-at": null,
        "email": "jdohoney@hashicorp.com",
        "name": "postman-test",
        "fair-run-queuing-enabled": false,
        "saml-enabled": false,
        "owners-team-saml-role-id": null,
        "slack-enabled": false,
        "session-remember": null,
        "session-timeout": null,
        "collaborator-auth-policy": null
      },
      "type": "organizations"
    }
  }'
```

Depending on your PTFE/TFE deployment, an existing Organization can be used as well, so this step can be omitted. If you do create a new Organization or reuse an existing one, be sure to update your Postman Environment with the name, in our case, “postman-test”

For the purposes of this guide, I will use an existing Organization within my TFE. I do believe this will be the typical case for most clients, as there will be some corporate governance behind the creation on Organizations.

MANAGE ENVIRONMENTS

×

Manage Environments

Environment Templates

Edit Environment

Hashicorp Terraform Enterprise

	Key	Value	Bulk Edit
<input checked="" type="checkbox"/>	API-BASE	app.terraform.io	
<input checked="" type="checkbox"/>	ORG	postman-test	×
<input checked="" type="checkbox"/>	WORKSPACE	aws-tfe-demo	
<input checked="" type="checkbox"/>	REPLACEWITHORGTODELETED	Replace-Me	
<input checked="" type="checkbox"/>	WORKSPACE-ID	ws-wBnph7GhrYBcrLS	
<input checked="" type="checkbox"/>	RUN-ID	run-11GvUBmF9tYEXqNM	
<input checked="" type="checkbox"/>	OAuth-TOKEN	0aj0JCUYwoAFyw.atlasv1.mgOHjtxvqt71ttmzsDqXg3...	
	New key	Value	

Cancel

Update

API Usage – Workspaces

The TFE API is no different than the GUI, as the Workspace will be the context for queuing a run in Terraform. The following TFE API call creates the Workspace and associates a Git Repository for creation of our infrastructure.

```
curl -X POST \
  https://app.terraform.io/api/v2/organizations/johndohoneyjr/workspaces \
  -H 'authorization: Bearer 0aj0JCUYwoAFyw.atlasv1.mgOHjtxvqt71ttmzsDqXg3OvBRLsBnKI7ej0PJfEWbDliVqQWIELc5azF8UxZqm71Vw' \
  -H 'content-type: application/vnd.api+json' \
  -d '{
    "data": {
      "attributes": {
        "name": "postman-example-ws",
        "environment": "default",
        "auto-apply": false,
        "working-directory": "",
        "terraform-version": null,
        "vcs-repo": {
          "identifier": "johndohoneyjr/aws-ha-simple-vpc",
          "oauth-token-id": "ot-JbGUhJCEmEvWNYb5",
          "branch": "",
          "ingress-submodules": false,
          "webhook-url": ""
        },
        "slack-enabled": false,
        "slack-channel": "",
        "migration-environment": null
      },
      "relationships": {
        "organization": {
          "data": {
            "type": "organizations",
            "id": "johndohoneyjr"
          }
        }
      },
      "type": "workspaces"
    }
  }'
```

Be sure to save off the Workspace ID, this will be needed in subsequent Variable POST calls to add variables to the Workspace. The technique to save this off is exactly like the OAUTH ID by using environment variables --- either in your BASH shell or in a CI/CD platform that also makes use of environment variables.

```

1  {
2  "data": {
3    "id": "ws-1YHkeodNq42sRyrh",
4    "type": "workspaces",
5    "attributes": {
6      "name": "postman-example-ws",
7      "environment": "default",
8      "auto-apply": false,
9      "locked": false,
10     "created-at": "2018-11-20T22:55:21.473Z",
11     "working-directory": "",
12     "terraform-version": "0.11.10",
13     "latest-change-at": "2018-11-20T22:55:21.473Z",
14     "vcs-repo": {
15       "branch": "",
16       "ingress-submodules": false,
17       "identifier": "johndohoneyjr/aws-ha-simple-vpc",

```

API Usage – Variables

There are many ways to write Terraform code, but I prefer a style that defers variable assignment to the terraform.tfvars file. This is an example of the terraform.tfvars file we will use for this example:

```

aws_region = "us-west-2"
project_name = "la-terraform"

accessip = "0.0.0.0/0"
vpc_cidr = "10.123.0.0/16"
public_cidrs = [
  "10.123.1.0/24",
  "10.123.2.0/24"
]

key_name = "dohoney-se-demos-west"
server_instance_type = "t2.micro"
instance_count = 2

```

From this list of variables, the only one you have to change is “key_name.” Especially if you want to access your instances and poke around. Otherwise, the worse case is you can’t SSH to your server.

In addition, we will need to add our AWS Environment. Assuming you are using AWS, these values can be found at ~/.aws/credentials. These are Environment Variable in TFE—because the file system is not present in the cloud. The file on your system should look something like following:

```
[default]
aws_access_key_id = the-value-of-access-key
aws_secret_access_key = the-value-of-the-secret-key
```

Terraform OSS uses these by default if you are using the Terraform AWS Provider. My TFE Environment for this Github repo contains:

Environment Variables

These variables are set in Terraform's shell environment using `export` .

AWS_ACCESS_KEY_ID	sensitive - write only
AWS_SECRET_ACCESS_KEY	sensitive - write only
AWS_DEFAULT_REGION	us-west-2
CONFIRM_DESTROY	1

The UI will only allow for the deletion of a workspace if “CONFIRM_DESTROY” is set to 1. This prevents erroneous deletions. This variable can be omitted if you like.

Since I have a nice mixture of variable types, the following section will define all the variables. Some are a little tricky to get right, with the required escaping for legal JSON, so these can serve as examples for your own work.

Building a general-purpose pipeline is a time-consuming process. Once you get it right, you are off to the races. So, far, the hard part has been getting variable escaping right. As it turns out, very little data has to be retained for subsequent calls. So, the time investment is highest in your first API integration. From that, second and third will be faster to create.

API – Terraform Variables

Many of the variables are similar, and only vary by:

```
data.attributes.key
data.attributes.value
```

So, consult the `terraform.tfvars` for these key-value pairs. Here is an example of a non-HCL value.

```

curl -X POST \
  https://app.terraform.io/api/v2/vars \
  -H 'authorization: Bearer 0aj0JCUYwoAFyw.atlasv1.mgOHjtxvqt71ttmzsDqXg3OvBRLsBnKI7ej0PJfEWbDliVqQWIELc5azF8UxZqm71Vw' \
  -H 'content-type: application/vnd.api+json' \
  -d '{
    "data": {
      "attributes": {
        "key": "aws_region",
        "value": "us-west-2",
        "category": "terraform",
        "hcl": false,
        "sensitive": false,
        "read-only": false,
        "created-at": null
      },
      "relationships": {
        "workspace": {
          "data": {
            "type": "workspaces",
            "id": "ws-1YHkeodNq42sRyrh"
          }
        }
      },
      "type": "vars"
    }
  }'

```

The following is a list. To keep my sanity with respect to proper escaping, I like to use: <https://jsonlint.com>. As shown below, this will indicate you have formulated your JSON properly.

The screenshot shows the JSONLint website interface. At the top, there's a navigation bar with links to Hashicorp, Apps, Accounts, Training, SDN, Health, and a search bar. Below the navigation bar, there's a main header area with a Toptal advertisement. The central part of the page features a large text area where a JSON object is pasted and validated. The JSON object is a Terraform variable definition. Below the text area, there are buttons for 'Validate JSON' and 'Clear'. To the right of these buttons is a link to 'Support JSONLint for \$2/Month'. At the bottom, a green box displays the result 'Valid JSON'.

```

1 {
2   "data": {
3     "attributes": {
4       "key": "public_cidrs",
5       "value": "[\n  \"10.123.1.0/24\", \n  \"10.123.2.0/24\"\n]",
6       "category": "terraform",
7       "hcl": true,
8       "sensitive": false,
9       "read-only": false,
10      "created-at": null
11    },
12    "relationships": {
13      "workspace": {
14        "data": {
15          "type": "workspaces",

```

Validate JSON Clear Support JSONLint for \$2/Month

Results

Valid JSON

Notice that the HCL attribute is true for passing the list into the TFE API.

```
curl -X POST \
  https://app.terraform.io/api/v2/vars \
  -H 'authorization: Bearer
0aj0JCUYwoAFyw.atlasv1.mgOHjtxvqt71ttmzsDqXg3OvBRLsBnKI7ej0PJfEWbDliVqQWIELc5azF8UxZqm71Vw' \
  -H 'content-type: application/vnd.api+json' \
  -d '{
    "data": {
      "attributes": {
        "key": "public_cidrs",
        "value": "[\n \"10.123.1.0/24\", \n \"10.123.2.0/24\" \n]",
        "category": "terraform",
        "hcl": true,
        "sensitive": false,
        "read-only": false,
        "created-at": null
      },
      "relationships": {
        "workspace": {
          "data": {
            "type": "workspaces",
            "id": "ws-1YHkeodNq42sRyrh"
          }
        }
      },
      "type": "vars"
    }
  }'
```


API – Environment Variables

There are similarities to most aspects of the environment POST body as the variable POST body with the following exceptions:

1. The category changes to “env”

With secure values, remember, these calls are transmitted over https, so the transport is secure. In the case of my AWS Key-Secret, the “sensitive” attribute is true

```
curl -X POST \
  https://app.terraform.io/api/v2/vars \
  -H 'authorization: Bearer
0aj0JCUYwoAFyw.atlasv1.mgOHjtxvqt71ttmzsDqXg3OvBRLsBnKI7ej0PJfEWbDliVqQWIELc5azF8UxZqm71Vw' \
  -H 'content-type: application/vnd.api+json' \
  -d '{
    "data": {
      "attributes": {
        "key": "AWS_ACCESS_KEY_ID",
        "value": "MY-SECRET-KEY-VALUE-PLAINTEXT",
        "category": "env",
        "hcl": false,
        "sensitive": true,
        "read-only": false,
        "created-at": null
      },
      "relationships": {
        "workspace": {
          "data": {
            "type": "workspaces",
            "id": "ws-1YHkeodNq42sRyrh"
          }
        }
      },
      "type": "vars"
    }
  }'
```

The following example is a simple environment variable, in this case so the Workspace can be deleted.

```
curl -X POST \
https://app.terraform.io/api/v2/vars \
-H 'authorization: Bearer
0aj0JCUYwoAFyw.atlasv1.mgOHjtxvqt71ttmzsDqXg3OvBRLsBnKI7ej0PJfEWbDliVqQWIELc5azF8UxZqm71Vw' \
-H 'content-type: application/vnd.api+json' \
-d '{
  "data": {
    "attributes": {
      "key": "CONFIRM_DESTROY",
      "value": "1",
      "category": "env",
      "hcl": false,
      "sensitive": false,
      "read-only": false,
      "created-at": null
    },
    "relationships": {
      "workspace": {
        "data": {
          "type": "workspaces",
          "id": "ws-1YHkeodNq42sRyrh"
        }
      }
    },
    "type": "vars"
  }
}
```

API – Configurations

```
curl -X POST \
https://app.terraform.io/api/v2/workspaces/ws-1YHkeodNq42sRyrh/configuration-versions \
-H 'authorization: Bearer
0aj0JCUYwoAFyw.atlasv1.mgOHjtxvqt71ttmzsDqXg3OvBRLsBnKI7ej0PJfEWbDliVqQWIELc5azF8UxZqm71Vw' \
-H 'content-type: application/vnd.api+json' \
-d '{
  "data": {
    "type": "configuration-versions",
    "attributes": {
      "auto-queue-runs": true
    }
  }
}
```

What is important is the returned URL to upload the compressed archive, this is found below in the attribute value: **data.attributes.upload_url**.

```
{
  "data": {
    "id": "cv-iuocETLjUzKf7iVE",
    "type": "configuration-versions",
    "attributes": {
      "auto-queue-runs": true,
      "error": null,
      "error-message": null,
      "source": "tfe-api",
      "status": "pending",
      "status-timestamps": {},
      "upload-url":
        "https://archivist.terraform.io/v1/object/dmF1bHQ6djE6dTVTa2tPQUUwNmRka2tIMEVVQnFnbElUa0VVamhTeFpSSTdQZDZBe
        nExMnAwWnZpc0tQbWxZMDg3N0dBY1lPdWU1dWJnWDISZlBmcUdLRVBaY1J3QjR0QVJEcGl4cmFFbFA0SXZ6dXFLbmYyWHNHY
        kkyb3B6WUxENHAzaU9EcG5jOTAwQ2VFbkysycmEwcWwwQ0FZd3RMZStGNUQ5YlJyUzhJQ3RJV3AxN1FJMHJYRG5oZkcwMWtUQ
        1UvRmlmWHJZN2dBOFFUL0I3ZzU1ZmpUMkRXNGE5eHMwT1p4ZkVYcWgyV1M4MEJPeHdEcnl5dVNMVE5FSVcVWmNqc0hINXIQ
        cEVzR3JTRXExM1ZnYUdRQWg4anp4UkxlbVJXTmswN0E9"
    },
    "relationships": {
      "ingress-attributes": {
        "data": null,
        "links": {
          "related": "/api/v2/configuration-versions/cv-iuocETLjUzKf7iVE/ingress-attributes"
        }
      }
    },
    "links": {
      "self": "/api/v2/configuration-versions/cv-iuocETLjUzKf7iVE"
    }
  }
}
```

Next