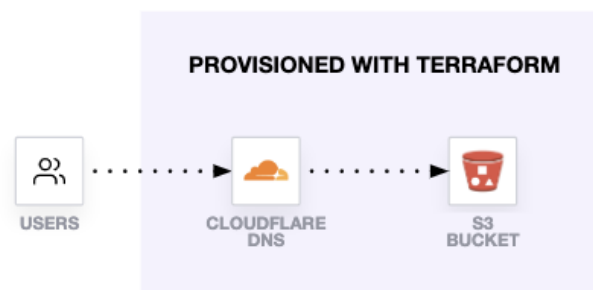


Host a Static Website with S3 and Cloudflare

This tutorial also appears in: [Use Cases](#), [Associate Tutorials](#) and [AWS Services](#).

Cloudflare is a popular service that offers a Content Delivery Network (CDN), Domain Name System (DNS), and protection against Distributed Denial of Service (DDoS) attacks. The Terraform Cloudflare provider allows you to deploy and manage your content distribution services with the same workflow you use to manage infrastructure. Using Terraform, you can provision DNS records and distribution rules for your web applications hosted in AWS and other cloud services, as well as the underlying infrastructure hosting your services.

In this tutorial, you will deploy a static website using the AWS and Cloudflare providers. The site will use AWS to provision an S3 bucket for object storage and Cloudflare for DNS, [SSL](#) and [CDN](#). Then, you will add Cloudflare [page rules](#) to always redirect HTTP requests to HTTPS and to temporarily redirect users when they visit a specific page.



»Prerequisites

This tutorial assumes that you are familiar with the standard Terraform workflow. If you are new to Terraform, complete the [Get Started tutorials](#) first.

For this tutorial, you will need:

- the [Terraform CLI](#) 0.14+ installed locally
- an [AWS account](#) with credentials [configured for Terraform](#)
- a [Cloudflare account](#)

- a domain name with nameservers pointing to Cloudflare. Cloudflare needs to manage the domain's DNS records. You can either [register a new domain](#) or [change an existing domain's nameserver to Cloudflare](#).

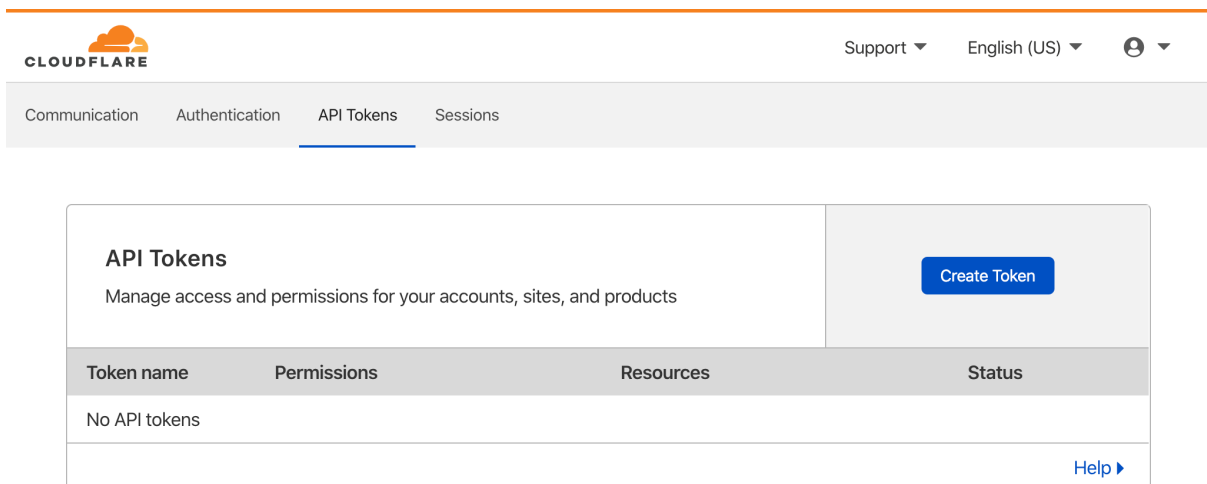
Note: Some of the infrastructure in this tutorial may not qualify for the AWS [free tier](#). Destroy the infrastructure at the end of the guide to avoid unnecessary charges. We are not responsible for any charges that you incur.

»Create a scoped Cloudflare API token

There are several ways to authenticate the Terraform Cloudflare provider. In this tutorial, you will use a Cloudflare API token. This method grants granular control of token permissions, keeps the token out of version control, and allows you to revoke the token when necessary.

This tutorial requires a Cloudflare API token with "Edit" permissions for your zone's DNS and Page Rules. If you would like to use an existing Cloudflare API token that already has these permissions, you can go to the [Clone the sample repository](#) section.

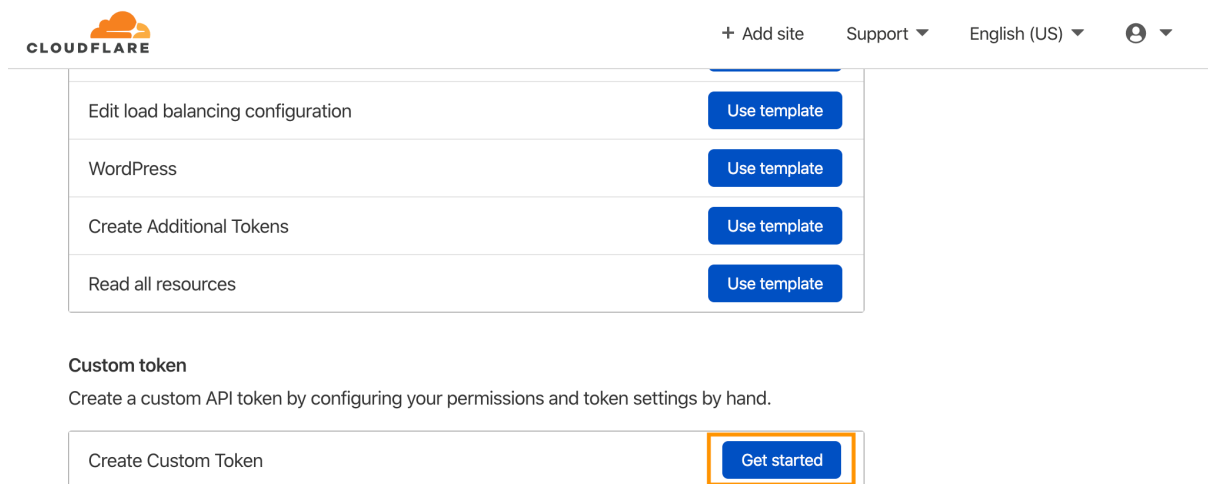
To create this scoped Cloudflare API token, go to the ["API Tokens"](#) page for your Cloudflare account. You can access this page by clicking on the user icon on the top right corner > "My Profile" > "API Tokens".



The screenshot shows the Cloudflare dashboard's "API Tokens" section. At the top, there's a navigation bar with the Cloudflare logo, "Support", "English (US)", and a user icon. Below this is a sub-navigation bar with "Communication", "Authentication", "API Tokens" (which is underlined), and "Sessions". The main content area has a header "API Tokens" with a subtitle "Manage access and permissions for your accounts, sites, and products" and a "Create Token" button. Below this is a table with columns "Token name", "Permissions", "Resources", and "Status". The table currently shows "No API tokens". A "Help" link is at the bottom right.

Token name	Permissions	Resources	Status
No API tokens			


Click on "Create Token", then the "Get Started" button near "Create Custom Token".




On the "Create Custom Token" page, modify the following fields:

1. In "Token name", enter "Terraform Token".
2. In the "Permissions" section, grant the API token permission to edit your zone's DNS and Page Rules.
3. Set the first row to "Zone", "DNS", and "Edit".
4. Set the second row to "Zone", "Page Rules", and "Edit".
5. In the "Zone Resources" section, select "Include", "Specific zone", and the domain you want to manage with Cloudflare.

This page should look like the following, with your domain name instead of hashicorp.fun in the "Zone Resources" section.



Support ▾English (US) ▾

CommunicationAuthenticationAPI TokensSessions

[← Back to view all tokens](#)

Create Custom Token

Token name

Give your API token a descriptive name.

Terraform Token

Permissions

Select edit or read permissions to apply to your accounts or websites for this token.

Zone ▾DNS ▾Edit ▾✕

Zone ▾Page Rules ▾Edit ▾✕

[+ Add more](#)


Zone Resources


Select zones to include or exclude.

Include ▾Specific zone ▾hashicorp.fun ▾

[+ Add more](#)

Click "Continue to summary", then "Create Token" to create your scoped Cloudflare API token.



Support ▾English (US) ▾

CommunicationAuthenticationAPI TokensSessions

[← Edit token](#)

Terraform Token API token summary

This API token will affect the below accounts and zones, along with their respective permissions

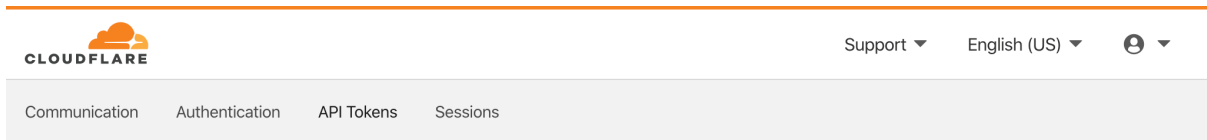
└─ hello@hashicorp.fun's Account

└─ hashicorp.fun **DNS:Edit, Page Rules:Edit**

Cancel

Create Token

Cloudflare only displays your API token once. Record it somewhere safe. You will use this token to authenticate the Cloudflare provider.



To avoid committing your API token to version control, set it as an environment variable.

Create an environment variable named `CLOUDFLARE_API_TOKEN` and set it to your Cloudflare API token. The Cloudflare provider will retrieve the API token from this environment variable.

```
$ export CLOUDFLARE_API_TOKEN=Oo-bF...
```

Terraform will reference this environment variable to authenticate the Cloudflare Provider.

»Clone the sample repository

Clone the [sample repository](#) for this tutorial, which contains Terraform configuration for an S3 bucket and Cloudflare DNS records. The next section will walk you through each resource's configuration.

```
$ git clone https://github.com/hashicorp/learn-terraform-cloudflare-static-website.git
```

Copy

Change into the repository directory.

```
$ cd learn-terraform-cloudflare-static-website
```

Copy

»Review configuration

In this section, you will review the files and Terraform resource definitions in the sample repository. If you want to jump ahead to provisioning the resources, you can go to the [Modify variables](#) section and use this section as a reference later.

This tutorial offers two options for CDN:

- Cloudflare - The simplest option is to use Cloudflare's native SSL and CDN, included in its [free tier](#). If you're using Cloudflare for DNS, Cloudflare's included SSL and CDN services allow you to set up a secure, cached static website with just an S3 bucket.
Note: To use Cloudflare for SSL and CDN, you must use an S3 bucket matching your domain name. Bucket names must be globally unique.
- AWS's ACM and CloudFront - If you cannot create an S3 bucket matching your domain name, you can use ACM for SSL certificate management and CloudFront for CDN. Using both ACM and Cloudfront allows you to secure and cache traffic to your S3 bucket.

Select the tab for your preferred CDN option to review the configuration. If you are not sure which to pick, choose Cloudflare for ease of use.

Cloudflare
ACM and CloudFront

You will find the following files in the sample repository:

- The main.tf file contains configuration to provision the S3 bucket and update your Cloudflare DNS records.
- The outputs.tf file defines outputs that display your S3 bucket name, bucket endpoint, and domain name.
- The providers.tf file contains the required_providers block that specifies which provider versions to use.
- The variables.tf file contains the input variable declarations, including the AWS region and website domain name.
- The terraform.tfvars.example file is an example variable definition file. Later in this tutorial, you will copy this file and modify it to include your AWS region and domain name.
- The /website directory contains [Terramino](#), a demo website containing a HashiCorp-skinned Tetris game. This directory is located in this repository to simplify this tutorial. Ordinarily you should not store your static website in the same directory as your Terraform configuration.
- The terraform.lock.hcl file ensures that [Terraform uses the same provider versions for each run](#).

Open the main.tf file in your editor to review the configuration.

- The AWS provider block [authenticates](#) to AWS, scoped to the region specified by the aws_region input variable. The Cloudflare provider authenticates using the scoped API token you created in the [Prerequisites](#), accessed by an environment variable.

```
provider "aws" {  
  
    region = var.aws_region  
  
}
```

```
provider "cloudflare" {}
```

- The `aws_s3_bucket.site`, `aws_s3_bucket_website_configuration.site`, `aws_s3_bucket_acl.site`, and `aws_s3_bucket_policy.site` resources create a new S3 bucket and set it to be publicly readable. This policy allows anyone to view your static website.

```
resource "random_pet" "bucket" {  
  
    length = 1  
  
}
```

```
locals {  
  
    bucket_name = "${random_pet.bucket.id}-${var.site_domain}"  
  
}
```

```
resource "aws_s3_bucket" "site" {  
  
    bucket = local.bucket_name  
  
}
```

```
resource "aws_s3_bucket_website_configuration" "site" {  
  
    bucket = aws_s3_bucket.site.id
```

```
index_document {  
  
    suffix = "index.html"  
  
}  
  
error_document {  
  
    key = "error.html"  
  
}  
  
}  
  
resource "aws_s3_bucket_acl" "site" {  
  
    bucket = aws_s3_bucket.site.id  
  
    acl = "public-read"  
  
}  
  
resource "aws_s3_bucket_policy" "site" {  
  
    bucket = aws_s3_bucket.site.id  
  
    policy = jsonencode({  
  
        Version = "2012-10-17"  
  
        Statement = [  
  
            {  
  
                Sid      = "PublicReadGetObject"  
  
                Effect    = "Allow"  
  
            }  
  
        ]  
  
    })  
  
}
```



```

Principal = "*"

Action    = "s3:GetObject"

Resource = [

    aws_s3_bucket.site.arn,

    "${aws_s3_bucket.site.arn}/*",

]

},

]

}))

}

```

- Tip: In this tutorial, the S3 bucket policy is set to public-read. When creating an S3 bucket, create the appropriate policy for your bucket.
- The `cloudflare_zones.domain` data source retrieves your Cloudflare zone ID. The Cloudflare resources in this configuration will use this value to apply changes to your zone.

```

data "cloudflare_zones" "domain" {

    filter {

        name = var.site_domain

    }

}

```

- Finally, the `cloudflare_record.site_cname` and `cloudflare_record.www` resources create a Cloudflare CNAME record that points to the CloudFront distribution domain name. Notice that both records have `proxied` set to `true`, which routes your site traffic

through Cloudflare's proxy. This enables you to use CloudFlare page rules and DDoS protection features.

```
resource "cloudflare_record" "site_cname" {

    zone_id = data.cloudflare_zones.domain.zones[0].id

    name    = var.site_domain

    value   = aws_s3_bucket.site.website_endpoint

    type    = "CNAME"

    ttl     = 1

    proxied = true

}

resource "cloudflare_record" "www" {

    zone_id = data.cloudflare_zones.domain.zones[0].id

    name    = "www"

    value   = var.site_domain

    type    = "CNAME"

    ttl     = 1

    proxied = true

}
```

»**Modify variables**

Copy the contents of terraform.tfvars.example into a new file named terraform.tfvars.

```
$ cp terraform.tfvars.example terraform.tfvars
```

Copy
Cloudflare
ACM and CloudFront

Open terraform.tfvars and modify the values to match your site_domain. Your terraform.tfvars file will look similar to the following.

```
aws_region      = "us-east-1"
site_domain     = "hashicorp.fun"
```

Warning: Never commit sensitive values into source control. The .gitignore file found in this repo ignores the terraform.tfvars file. You should include a .gitignore file your Terraform repositories.

»Apply configuration

Initialize the Terraform configuration.

```
$ terraform init
```

Initializing the backend...

Initializing provider plugins...

- Reusing previous version of hashicorp/random from the dependency lock file
- Reusing previous version of hashicorp/aws from the dependency lock file
- Reusing previous version of cloudflare/cloudflare from the dependency lock file
- Installing hashicorp/aws v4.0.0...
- Installed hashicorp/aws v4.0.0 (signed by HashiCorp)
- Installing cloudflare/cloudflare v2.19.2...
- Installed cloudflare/cloudflare v2.19.2 (signed by a HashiCorp partner, key ID DE413CEC881C3283)
- Installing hashicorp/random v3.1.0...
- Installed hashicorp/random v3.1.0 (signed by HashiCorp)

Partner and community providers are signed by their developers.

If you'd like to know more about provider signing, you can read about it here:
<https://www.terraform.io/docs/cli/plugins/signing.html>

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see any changes that are required for your infrastructure. All Terraform commands should now work.

If you ever set or change modules or backend configuration for Terraform, rerun this command to reinitialize your working directory. If you forget, other commands will detect it and remind you to do so if necessary.

Copy

Next, apply the configuration. Respond yes to the prompt to confirm.

Note: If you encounter a duplicate name error while creating the S3 bucket, [destroy the provisioned resources](#), then select the "ACM and CloudFront tab" in the [Review configuration](#) section and follow the steps to re-provision.

```
$ terraform apply
```

```
## ...
```

Plan: 5 to add, 0 to change, 0 to destroy.

Changes to Outputs:

```
+ bucket_endpoint    = (known after apply)
+ domain_name        = "hashicorp.fun"
+ website_bucket_name = (known after apply)
```

Do you want to perform these actions?

Terraform will perform the actions described above.

Only 'yes' will be accepted to approve.

Enter a value:

Copy

Once finished, Terraform returns the output parameters.

Apply complete! Resources: 5 added, 0 changed, 0 destroyed.

Outputs:

```
bucket_endpoint = "hashicorp.fun.s3-website-us-east-1.amazonaws.com"
domain_name     = "hashicorp.fun"
website_bucket_name = "hashicorp.fun"
```

»Add website files to S3 bucket

Now that you have set up the static website, upload the contents in the /website directory to your newly provisioned S3 bucket. Notice that the following command retrieves the bucket name from Terraform output.

```
$ aws s3 cp website/ s3://$(terraform output -raw website_bucket_name)/  
--recursive  
upload: website/index.html to s3://turkey-hashicorp.fun/index.html  
upload: website/background.png to s3://turkey-hashicorp.fun/background.png
```

Copy

Navigate to your website in your web browser. Your Terramino app should start automatically.

»Create Cloudflare page rules

Cloudflare has page rules that trigger actions whenever the page URL matches a specified URL pattern. You can use page rules to forward URLs, configure security/cache levels, and enforce HTTPS. Refer to [Cloudflare's recommended page rules](#) for more use cases.

First, add a page rule to the main.tf file to convert any http:// request to https:// using a 301 redirect.

```
resource "cloudflare_page_rule" "https" {  
  zone_id = data.cloudflare_zones.domain.zones[0].id  
  target = ".*${var.site_domain}/*"  
  actions {  
    always_use_https = true  
  }  
}
```

Copy

Next, add another page rule to the main.tf file to temporarily redirect <your-domain>/learn to the Terraform Learn page, where your-domain is your domain name.

```
resource "cloudflare_page_rule" "redirect-to-learn" {  
  zone_id = data.cloudflare_zones.domain.zones[0].id  
  target = "${var.site_domain}/learn"  
  actions {  
    forwarding_url {
```

```
    status_code = 302
    url          = "https://learn.hashicorp.com/terraform"
  }
}
}
```

Copy

Apply these changes. Respond yes to the prompt to confirm.

```
$ terraform apply
```

```
## ...
```

```
cloudflare_page_rule.redirect-to-learn: Creating...
cloudflare_page_rule.https: Creating...
cloudflare_page_rule.https: Creation complete after 1s
[id=5681df98c982f0b5af15d5183756a487]
cloudflare_page_rule.redirect-to-learn: Creation complete after 1s
[id=c6be51ba9e52cb21ce9c7c8fd584bd22]
```

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

```
bucket_endpoint = "hashicorp.fun.s3-website-us-east-1.amazonaws.com"
domain_name     = "hashicorp.fun"
website_bucket_name = "hashicorp.fun"
```

Copy

Verify these changes by visiting <http://your-domain.com> and <https://your-domain.com/learn>, where your-domain is your domain name. In the first instance, your browser should redirect you to the <https://> version of the website. In the second instance, your browser should redirect you to the [Terraform Learn page](#).

»Clean up resources

Congrats! You successfully used Terraform to set up a SSL secured static website with S3 and Cloudflare. You can choose to keep your website or destroy it.

Keep website

Destroy website

You can repurpose this website to your needs by updating the contents of the S3 bucket

You may want to remove `cloudflare_page_rule.redirect-to-learn`, which temporarily redirects `your-domain.com/learn` to the Terraform Learn page. Remove the resource from `main.tf`.

```
- resource "cloudflare_page_rule" "redirect-to-learn" {  
-   zone_id = data.cloudflare_zones.domain.zones[0].id  
-   target  = "${var.site_domain}/learn"  
-   actions {  
-     forwarding_url {  
-       status_code = 302  
-       url         = "https://learn.hashicorp.com/terraform"  
-     }  
-   }  
- }
```

Apply these changes. Respond yes to the prompt to confirm.

```
$ terraform apply
```

```
## ...
```

Terraform will perform the following actions:

```
# cloudflare_page_rule.https will be updated in-place  
~ resource "cloudflare_page_rule" "https" {  
  id      = "REDACTED"  
  ~ priority = 2 -> 1  
  # (3 unchanged attributes hidden)  
  
  # (1 unchanged block hidden)  
}  
  
# cloudflare_page_rule.redirect-to-learn will be destroyed  
- resource "cloudflare_page_rule" "redirect-to-learn" {  
  - id      = "REDACTED" -> null  
  - priority = 1 -> null  
  - status  = "active" -> null  
  - target  = "hashicorp.fun/learn" -> null  
  - zone_id = "REDACTED" -> null  
  
  - actions {  
    - always_use_https = false -> null  
    - disable_apps     = false -> null  
    - disable_performance = false -> null
```

```

- disable_railgun    = false -> null
- disable_security   = false -> null
- edge_cache_ttl     = 0 -> null

- forwarding_url {
  - status_code = 302 -> null
  - url         = "https://learn.hashicorp.com/terraform" -> null
}
}
}

```

Plan: 0 to add, 1 to change, 1 to destroy.

...

```

cloudflare_page_rule.redirect-to-learn: Destroying...
[id=c6be51ba9e52cb21ce9c7c8fd584bd22]
cloudflare_page_rule.https: Modifying... [id=5681df98c982f0b5af15d5183756a487]
cloudflare_page_rule.redirect-to-learn: Destruction complete after 0s
cloudflare_page_rule.https: Modifications complete after 1s
[id=5681df98c982f0b5af15d5183756a487]

```

Apply complete! Resources: 0 added, 1 changed, 1 destroyed.

...

Next steps

To learn more about managing the resources used in this tutorial with Terraform, visit the following documentation:

1. The [Terraform Cloudflare Provider](#) Registry page contains documentation for Cloudflare resources, including arguments, attributes, and example configuration.
2. The [Create IAM policies](#) tutorial guides you through creating differently scoped IAM policies for your AWS resources.
3. The [Cloudflare-managed Terraform documentation](#) contains tutorials for topics such as using the Cloudflare provider to rate limit and load balance requests, importing existing Cloudflare resources, and customizing the provider.
4. This [AWS Knowledge Center post](#) walks you through updating your CloudFront cache to reflect the latest changes to your S3 bucket.