S3 Static Site — Step-by-step with file structure

This document contains a **complete step-by-step guide** to deploy a simple **"Hello World"** static site to an AWS S3 bucket using **Terraform**, validate Terraform with **Conftest (OPA)**, and add a post-functional test using **Terratest** (Go). A `Makefile` ties the workflow together so you can demonstrate the process to your team lead.

Overview

What you'll get: - Terraform configuration that creates an S3 bucket with static website hosting enabled and uploads `index.html`. - A Conftest (OPA) policy and instructions to run it against a Terraform plan JSON. - A Terratest `Go` test which performs a post-deploy functional check. - `Makefile` targets to run the full flow: init, plan, apply, conftest, test, destroy.

Prerequisites

- Terraform installed (v1.0+ recommended).
- AWS CLI configured (`aws configure`) or environment variables `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_REGION`.
- Go installed (for Terratest) and `$GOPATH` / module support.
- `conftest` installed (https://www.conftest.dev/) for OPA tests.
- `make` available.

Project structure

```
s3-static-site/
├── README.md
├── index.html
├── Makefile
├── provider.tf
├── variables.tf
├── main.tf
├── outputs.tf
├── policy/
│   └── terraform.rego
└── test/
    ├── go.mod
    └── s3_static_test.go
```

File contents

> **Important:** Do not blindly copy/paste into a production account. This example creates a public S3 bucket (intended for static website hosting). For production, consider CloudFront + OAI and stricter controls.

`index.html`

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Hello World</title>
</head>
<body>
  <h1>Hello World from S3!</h1>
</body>
</html>
```

`provider.tf`

```
terraform {
  required_version = ">= 1.0.0"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = ">= 4.0"
    }
  }
}

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

`variables.tf`

```
variable "aws_region" {
  description = "AWS Region"
  type        = string
  default     = "us-east-1"
}

variable "bucket_name" {
  description = "Name of the S3 bucket to create (must be globally unique)"
  type        = string
}
```

main.tf

```
resource "aws_s3_bucket" "static_site" {
  bucket = var.bucket_name
  acl    = "public-read"   # public so website can be served directly

  website {
    index_document = "index.html"
    error_document = "index.html"
  }

  tags = {
    Name        = "HelloWorldStaticSite"
    Environment = "dev"
  }
}

resource "aws_s3_bucket_ownership_controls" "ownership" {
  bucket = aws_s3_bucket.static_site.id
  rule {
    object_ownership = "BucketOwnerPreferred"
  }
}

resource "aws_s3_bucket_public_access_block" "public_access" {
  bucket = aws_s3_bucket.static_site.id

  block_public_acls       = false
  block_public_policy     = false
  ignore_public_acls      = false
  restrict_public_buckets = false
}

resource "aws_s3_bucket_policy" "public_policy" {
  bucket = aws_s3_bucket.static_site.id
  policy = jsonencode({
    Version = "2012-10-17",
    Statement = [
      {
        Sid       = "PublicReadGetObject",
        Effect    = "Allow",
        Principal = "*",
        Action    = ["s3:GetObject"],
        Resource  = "${aws_s3_bucket.static_site.arn}/*"
      }
    ]
  })
}

resource "aws_s3_bucket_object" "index" {
```

```
  bucket       = aws_s3_bucket.static_site.id
  key          = "index.html"
  source       = "index.html"
  content_type = "text/html"
  acl          = "public-read"
}
```

`outputs.tf`

```
output "bucket_name" {
  description = "S3 bucket name"
  value       = aws_s3_bucket.static_site.id
}

output "website_url" {
  description = "S3 website endpoint"
  value       = aws_s3_bucket.static_site.website_endpoint
}
```

`policy/terraform.rego` **(Conftest OPA policy)**

This policy demonstrates a few simple checks: - Bucket name must start with `dev-` (example requirement) - Bucket must set tags containing `Environment`.

```
package terraform.s3

deny[msg] {
  resource := input.planned_values.root_module.resources[_]
  resource.type == "aws_s3_bucket"
  not startswith(resource.values.bucket, "dev-")
  msg = sprintf("S3 bucket name '%s' does not start with 'dev-'",
[resource.values.bucket])
}

deny[msg] {
  resource := input.planned_values.root_module.resources[_]
  resource.type == "aws_s3_bucket"
  not resource.values.tags.Environment
  msg = sprintf("S3 bucket '%s' does not have tags.Environment set",
[resource.values.bucket])
}
```

> Notes: - Conftest reads the Terraform plan JSON (`terraform show -json`). The layout used above (`planned_values.root_module.resources`) matches the Terraform plan JSON structure.

`test/go.mod`

```
module github.com/example/s3-static-site/test

go 1.20

require (
    github.com/gruntwork-io/terratest v0.42.0
    github.com/stretchr/testify v1.8.4
)
```

`test/s3_static_test.go`

```
package test

import (
    "io"
    "net/http"
    "strings"
    "testing"

    "github.com/gruntwork-io/terratest/modules/terraform"
    "github.com/stretchr/testify/assert"
)

func TestS3StaticSite(t *testing.T) {
    t.Parallel()

    opts := &terraform.Options{
        TerraformDir: "..",
    }

    // Destroy at the end of the test run
    defer terraform.Destroy(t, opts)

    // Init and apply
    terraform.InitAndApply(t, opts)

    websiteURL := terraform.Output(t, opts, "website_url")

    // website_url sometimes returns host without scheme, ensure we have
scheme
    if !strings.HasPrefix(websiteURL, "http") {
        websiteURL = "http://" + websiteURL
    }

    resp, err := http.Get(websiteURL)
    assert.NoError(t, err)
    defer resp.Body.Close()
```

```
    body, err := io.ReadAll(resp.Body)
    assert.NoError(t, err)
    assert.Equal(t, 200, resp.StatusCode)
    assert.Contains(t, string(body), "Hello World")
}
```

`Makefile`

```
SHELL := /bin/bash

init:
    terraform init

fmt:
    terraform fmt -recursive

validate:
    terraform validate

plan:
    terraform plan -var="bucket_name=$(BUCKET_NAME)" -out=tfplan.binary

show-plan-json: plan
    terraform show -json tfplan.binary > plan.json

conftest: show-plan-json
    conftest test plan.json -p policy || (echo "Conftest failed" && exit 1)

apply:
    terraform apply -auto-approve -var="bucket_name=$(BUCKET_NAME)"

test: init
    cd test && go test -v

destroy:
    terraform destroy -auto-approve -var="bucket_name=$(BUCKET_NAME)"

clean:
    rm -f tfplan.binary plan.json
```

Step-by-step run (example)

1. Edit `variables.tf` or pass `BUCKET_NAME` on the Make command line. **Bucket name must be globally unique** (e.g. `dev-hello-world-example-12345`).

2. Initialize Terraform:

```
make init
```

1. Format and validate:

```
make fmt
make validate
```

1. Create a plan and export it to `plan.json` and run Conftest:

```
make plan BUCKET_NAME=dev-hello-world-example-12345
make show-plan-json
make conftest
```

If Conftest fails, fix the Terraform code or the values to satisfy your policy.

1. Apply the configuration:

```
make apply BUCKET_NAME=dev-hello-world-example-12345
```

Terraform outputs include `website_url`. Example returned endpoint:

```
my-bucket.s3-website-us-east-1.amazonaws.com
```

1. Verify manually by opening the `website_url` in a browser — you should see **Hello World from S3!**

2. Run the automated Terratest (post-functional) test (this will run `terraform init/apply` again inside the test unless you modify the test options):

```
make test
```

1. Destroy the environment when finished:

```
make destroy BUCKET_NAME=dev-hello-world-example-12345
make clean
```

Demonstration checklist for your team lead

- [ ] Show `Makefile` targets and run `make init` and `make apply` with a unique bucket name.
- [ ] Point the browser to the `website_url` and show the `Hello World` page.

- [ ] Run `make plan` + `make show-plan-json` and then `make conftest` to demonstrate compliance checks.
- [ ] Run `make test` (Terratest) to demonstrate the end-to-end verification.
- [ ] Run `make destroy` to show clean up.

Notes & Suggestions

- For production, avoid making the bucket public. Use CloudFront + Origin Access Identity (or OAC) and restrict bucket policy to CloudFront only.
- Conftest policies should be adjusted to reflect your org gates (naming convention, tags, public access rules, cost controls, etc.).
- Terratest tests may take time to run as they perform infrastructure actions. Set appropriate timeouts and use `t.Parallel()` strategically.

If you want, I can: - Generate a `cloudfront` + `acm` example in Terraform to front the S3 site. - Add an IAM policy and an automation script to request sandbox admin access after Conftest passes. - Customize Conftest policies to match your org's naming/tagging guidelines.

Tell me which of those you'd like next.