Document: Deploying Node.js Application on Google Cloud App Engine with Terraform

In response to the request to create a Terraform configuration for Google Cloud App Engine, the following document outlines the key steps and configuration files involved in this process.

Introduction

This document details the deployment of a Node.js application on Google Cloud App Engine using Terraform. The provided Terraform configuration files, main.tf and variables.tf, enable the creation of necessary resources such as a Google Cloud Storage bucket and the deployment of the Node.js application on App Engine.

main.tf

```
terraform {

  required_providers {

    google = {

      source  = "hashicorp/google"

      version = "5.15.0"

    }

  }

  backend "gcs" {

    bucket = var.bucket_name

    prefix = "ts-appengine-terraform/state"

  }

}

provider "google" {

  project = var.project_id

  region  = var.region

}
```

```
resource "google_storage_bucket" "terraform_state" {

  name         = var.bucket_name

  force_destroy = false

  location      = "US"

  storage_class = "STANDARD"

  versioning {

    enabled = true

  }

}

resource "google_storage_bucket_object" "object" {

  name   = var.bucket_object_name

  bucket = google_storage_bucket.terraform_state.name

  source = var.source_path

}

resource "google_app_engine_standard_app_version" "myapp_v1" {

  version_id = var.app_version

  service    = var.service_name

  runtime    = var.runtime

  entrypoint {

    shell = var.entrypoint_shell

  }

  deployment {

    zip {

      source_url = 
"https://storage.googleapis.com/${google_storage_bucket.terraform_state.name}/${google_storage_bucket_obj
ect.object.name}"

    }

  }
  env_variables = {
    port = var.environment_port
  }
  automatic_scaling {
    max_concurrent_requests = 10
    min_idle_instances     = var.automatic_scaling_min_instances
    max_idle_instances     = 3
    min_pending_latency    = "1s"
    max_pending_latency    = "5s"
    standard_scheduler_settings {
      target_cpu_utilization        = 0.5
      target_throughput_utilization  = 0.75
      min_instances                 = var.automatic_scaling_min_instances
      max_instances                 = var.automatic_scaling_max_instances
    }
  }
  delete_service_on_destroy = true
}
```

Variables.tf

```
variable "project_id" {
  description = "The Google Cloud Project ID."
  type       = string
  default     = "your-project-id"
}
variable "region" {
  description = "The region where resources will be deployed."
  type        = string
  default     = "us-central1"
}
variable "bucket_name" {
  description = "The name of the Google Cloud Storage bucket."
  type        = string
  default     = "your-bucket-name"
}
variable "app_version" {
  description = "The version of the application."
  type        = string
  default     = "v1"
}
variable "service_name" {
  description = "The name of the App Engine service."
  type        = string
  default     = "your-service-name"
}
variable "bucket_object_name" {
  description = "The name of the Google Cloud Storage bucket object."
  type        = string
  default     = "your-app.zip"
}
variable "source_path" {
  description = "The local path to the source code."
  type        = string
  default     = "./your-app.zip"
```

```
}

variable "entrypoint_shell" {

  description = "The shell command for the App Engine entrypoint."

  type        = string

  default     = "node ./index.js"

}

variable "environment_port" {

  description = "The port to be used in the environment variables."

  type        = string

  default     = "8080"

}

variable "runtime" {

  description = "The runtime for the App Engine."

  type        = string

  default     = "nodejs18"

}

variable "automatic_scaling_min_instances" {

  description = "The minimum number of instances for automatic scaling."

  type        = number

  default     = 1

}

variable "automatic_scaling_max_instances" {

  description = "The maximum number of instances for automatic scaling."

  type        = number

  default     = 10

}
```

Deployment Process

Follow these steps to deploy the Node.js application:

- Initialize Terraform: terraform init
- Validate the configuration: terraform validate
- Plan the deployment: terraform plan
- Apply the changes: terraform apply

Once completed, the Node.js application should be successfully deployed on Google Cloud App Engine.

Conclusion

This Terraform configuration simplifies the deployment process of a Node.js application on Google Cloud App Engine, providing a structured and reproducible approach.