**Deploy Cloud Infra with GitHub Copilot**

Here’s a practical, end-to-end, beginner-friendly workflow to use **GitHub Copilot** to scaffold and deploy **Infrastructure as Code (IaC)** on a public cloud. It’s cloud-agnostic, but I’ll show concrete examples for **AWS with Terraform** (you can swap in Azure/GCP easily—see Step 11).

**0) Prerequisites (one-time)**

* **Accounts/CLIs**
  + GitHub account with **Copilot** enabled.
  + Cloud account (AWS/Azure/GCP) + CLI installed and configured:
    - AWS: aws configure (needs an IAM user/role with least-privileged access).
    - Azure: az login
    - GCP: gcloud init
* **Tools**
  + **VS Code** + Git + **GitHub Copilot** extension (and Copilot Chat if available).
  + **Terraform** or **OpenTofu** (or Pulumi/Ansible if you prefer—see Step 11).
* **Security**
  + Prepare **short-lived credentials** or OIDC to avoid long-lived keys.
  + Create a *sandbox* cloud project/subscription/account to avoid surprises.

**1) Create a fresh repo and describe your goal**

1. In GitHub: **New repository** → e.g., cloud-iac-demo.
2. Clone and open in VS Code.
3. Create an empty README.md and type a clear spec at the top—Copilot uses context from open files:
4. Deploy a minimal, production-ready VPC/network, one compute instance, and an S3 bucket (or equivalent),
5. with tags, remote backend, workspaces for dev/stage/prod, GitHub Actions CI for plan/apply, and tfsec scan.

**Pro tip**: The clearer your spec in README.md, the better Copilot’s first drafts.

**2) Initialize a Terraform (or OpenTofu) project with Copilot**

Create main.tf and start typing a comment describing what you want. Pause after a few lines to let Copilot suggest a block; accept (Tab) or cycle suggestions:

**Prompt (as code comments in main.tf):**

# Terraform project for AWS:

# - Use aws provider ~> 5.0

# - Region via variable with default "us-east-1"

# - Backend: S3 with DynamoDB table for state locking

# - Enable tags for all resources

# - Create a VPC, public subnet, internet gateway, route table/association

# - Create a security group allowing SSH from my IP and HTTP from 0.0.0.0/0

# - Launch a small EC2 instance (t3.micro) with latest Amazon Linux

# - Create an S3 bucket with versioning and block public access

# - Outputs: vpc\_id, subnet\_id, instance\_public\_ip, bucket\_name

Let Copilot draft the provider, backend, and core resources. Edit as needed.

**Create variables.tf and outputs.tf similarly:**

**Prompt for variables.tf:**

# Define variables:

# region (default us-east-1)

# project\_name (default "iac-demo")

# environment (no default; dev|stage|prod)

# my\_ip\_cidr for SSH (no default)

**Prompt for outputs.tf:**

# Output vpc\_id, public\_subnet\_id, instance\_public\_ip, s3\_bucket\_name

**3) Add remote backend & state locking**

Create a one-time “bootstrap” to hold Terraform state (S3 bucket + DynamoDB). You can:

* Use a separate small bootstrap/ Terraform or
* Manually create them via console once.

**Prompt for bootstrap/main.tf:**

# Create S3 bucket for tf state (unique name) and DynamoDB table for state locking

# Bucket: versioning enabled, block public access

# DynamoDB table: "terraform-locks" with PK "LockID"

Run:

cd bootstrap

terraform init

terraform apply

Then reference this backend in the root main.tf:

terraform {

backend "s3" {

bucket = "YOUR-STATE-BUCKET"

key = "envs/${var.environment}/terraform.tfstate"

region = var.region

dynamodb\_table = "terraform-locks"

encrypt = true

}

}

**4) Separate environments with Terraform workspaces**

terraform init

terraform workspace new dev

terraform workspace select dev

Create env/dev.tfvars:

project\_name = "iac-demo"

environment = "dev"

region = "us-east-1"

my\_ip\_cidr = "YOUR.PUBLIC.IP.ADDR/32"

Now:

terraform validate

terraform plan -var-file="env/dev.tfvars"

terraform apply -var-file="env/dev.tfvars"

**5) Use Copilot to harden defaults**

Open main.tf and add comments for best practices. Copilot will suggest config:

**Prompt in main.tf:**

# Best practices:

# - Add default tags using provider-level default\_tags

# - Enforce bucket "block\_public\_acls", "block\_public\_policy", "ignore\_public\_acls", "restrict\_public\_buckets"

# - Add lifecycle rules for non-current versions after 30 days in nonprod

# - Security group: limit SSH to var.my\_ip\_cidr only

Review and accept.

**6) Add tfsec + fmt + validate locally**

Create Makefile:

**Prompt in Makefile:**

# Targets: fmt, validate, tfsec, plan, apply, destroy

# Use -var-file passed via ENV: TFVARS ?= env/dev.tfvars

Then:

make fmt

make validate

make tfsec

make plan TFVARS=env/dev.tfvars

**7) Set up GitHub Actions CI/CD with Copilot**

Create .github/workflows/iac.yml.

**Prompt in iac.yml:**

# GitHub Actions workflow:

# - Trigger on pull\_request for plan; on push to main for apply (manual approval recommended)

# - Jobs: setup terraform, terraform fmt/validate, tfsec, terraform plan

# - Use OIDC to assume an IAM role in AWS (no long-lived secrets)

# - Store plan output as artifact; require environment protection for apply

Accept/edit Copilot’s draft. Add:

* **Environment** in repo settings (dev, stage, prod) with required reviewers for apply.
* **OIDC role** in AWS that trusts GitHub (policy for specific repo + environment).  
  *(Copilot can draft the IAM trust & permissions policy if you paste a spec into a file and ask for it.)*

**8) Pull Request workflow**

* Create a feature branch: feat/vpc-and-ec2.
* Make changes, commit & push.
* Open a PR. The workflow should:
  + run fmt, validate, tfsec
  + run terraform plan and attach the plan as an artifact or PR comment.
* Review the plan. When happy, merge to main.
* The apply job runs only after **manual approval** in the target environment.

**Prompt for PR description (in the PR body):**

Draft a concise PR description summarizing infra changes, risks, costs, and rollback plan.

Copilot will generate a nice summary based on diffs and your README.

**9) Verify resources and tag hygiene**

* Use CLI to verify:
* aws ec2 describe-instances --filters "Name=tag:Project,Values=iac-demo"
* aws s3 ls | grep iac-demo
* Test SSH/HTTP if you created an instance and security group.

**10) Teardown safely**

* For dev:
* terraform destroy -var-file="env/dev.tfvars"
* Delete workspace if not needed:
* terraform workspace select default
* terraform workspace delete dev

**11) Swap to Azure or GCP (quick Copilot prompts)**

**Azure (Terraform):** in main.tf

# Azure provider azurerm ~> 4.0

# Resource group, virtual network + subnet, NSG allowing SSH/HTTP

# Linux VM (B1s) with SSH key

# Storage account with public access disabled, TLS 1.2, versioning

# Backend: azurerm (resource group + storage account + container)

# Outputs: resource\_group\_name, public\_ip, storage\_account\_name

**GCP (Terraform):**

# Google provider ~> 5.0

# VPC, subnet, firewall (allow ssh/http)

# Compute Engine e2-micro, minimal boot disk

# GCS bucket with uniform bucket-level access and versioning

# Backend: gcs (bucket and prefix)

# Outputs: network\_self\_link, instance\_external\_ip, bucket\_name

Copilot will draft most of this if you paste the prompt into main.tf and a README.md section.

**12) Alternative stacks with Copilot**

* **OpenTofu**: same workflow as Terraform.
* **Pulumi (TypeScript/Python/C#)**:
  + Ask Copilot to “init Pulumi TypeScript project for AWS VPC + EC2 + S3, with config per stack and GitHub Actions using OIDC.”
* **Ansible**:
  + Use Ansible to provision on top of compute:
  + # Prompt in playbook.yml:
  + # - Install Docker on target EC2 and run Nginx container
  + # - Idempotent, handlers for service restart, variables for AMI family differences
* **Crossplane** on Kubernetes: “Install Crossplane on EKS/AKS/GKE and define an XRD for a ‘Network+Instance’ composite; generate composition and claim.”

**13) Prompt patterns that work great with Copilot**

Use these **verbatim** inside files or in Copilot Chat:

* **Scaffold a provider & backend**
  + *“Write Terraform provider + remote backend (S3+DynamoDB) with variables for region/project/environment and default tags.”*
* **Secure S3/Azure Storage/GCS**
  + *“Harden storage bucket: block all public access, enable versioning, add lifecycle for noncurrent versions in dev only.”*
* **Network + instance**
  + *“Create VPC/VNet with one public subnet, IGW/route table, SG/NSG for SSH from my\_ip\_cidr, and a tiny Linux VM.”*
* **Pipelines**
  + *“Generate GitHub Actions that runs fmt, validate, tfsec, plan on PR; apply on main with environment approval and OIDC role.”*
* **Docs**
  + *“Write a README section explaining how to set up state backend, workspaces, and env tfvars.”*

If a suggestion is *almost right*, accept and edit. If it’s off, **give more context** (paste a small spec or example).

**14) Guardrails & best practices**

* **Review everything** Copilot proposes; treat it like a junior pair-programmer.
* **Policies & scanning**
  + Add **tfsec** (or checkov), **terraform validate**, and optionally **OPA/Conftest** policies.
* **Least privilege**
  + Limit the OIDC-assumed role to specific repo, env, and actions.
* **Environments**
  + Separate state per environment; consider separate accounts/subscriptions for prod.
* **Costs**
  + Prefer free-tier-friendly sizes (t3.micro, B1s, e2-micro) and lifecycle rules.
* **Secrets**
  + Prefer **OIDC** over long-lived keys; if secrets are needed, store them in GitHub **Environments**.

**15) Minimal working example (AWS) you can generate with Copilot**

**main.tf (skeleton you can paste + let Copilot complete):**

terraform {

required\_version = ">= 1.7.0"

required\_providers {

aws = { source = "hashicorp/aws", version = "~> 5.0" }

}

backend "s3" {}

}

provider "aws" {

region = var.region

default\_tags {

tags = {

Project = var.project\_name

Environment = var.environment

ManagedBy = "terraform"

}

}

}

# VPC, subnet, IGW, route table, SG...

# EC2 instance...

# S3 bucket hardened...

**variables.tf:**

variable "region" { type = string, default = "us-east-1" }

variable "project\_name" { type = string, default = "iac-demo" }

variable "environment" { type = string }

variable "my\_ip\_cidr" { type = string }

**Commands:**

terraform init

terraform workspace new dev

terraform plan -var-file="env/dev.tfvars"

terraform apply -var-file="env/dev.tfvars"