**Managing infrastructure as code with Terraform, Cloud Build, and GitOps**

The [State of DevOps](https://cloud.google.com/devops) reports identified capabilities that drive software delivery performance. This tutorial will help you with the following capabilities:

* [Version control](https://cloud.google.com/solutions/devops/devops-tech-version-control)
* [Continuous integration](https://cloud.google.com/solutions/devops/devops-tech-continuous-integration)
* [Continuous delivery](https://cloud.google.com/solutions/devops/devops-tech-continuous-delivery)
* [Continuous testing](https://cloud.google.com/solutions/devops/devops-tech-test-automation)

**Architecture**

To demonstrate how this tutorial applies GitOps practices for managing Terraform executions, consider the following architecture diagram. Note that it uses GitHub branches—dev and prod—to represent actual environments. These environments are defined by Virtual Private Cloud (VPC) networks—dev and prod, respectively—into a Google Cloud project.

**Note:** For simplicity, we implimented only **dev** and **prod** environments using VPCs. You can extend this behavior to deploy to more environments and to create projects under your [organization hierarchy](https://cloud.google.com/resource-manager/docs/cloud-platform-resource-hierarchy) if needed.

The process starts when you push Terraform code to either the dev or prod branch. In this scenario, Cloud Build triggers and then applies Terraform manifests to achieve the state you want in the respective environment. On the other hand, when you push Terraform code to any other branch—for example, to a feature branch—Cloud Build runs to execute terraform plan, but nothing is applied to any environment.

Ideally, either developers or operators must make infrastructure proposals to [non-protected branches](https://help.github.com/en/articles/about-protected-branches) and then submit them through [pull requests](https://help.github.com/en/articles/about-pull-requests). The [Cloud Build GitHub app](https://github.com/marketplace/google-cloud-build), discussed later in this tutorial, automatically triggers the build jobs and links the terraform plan reports to these pull requests. This way, you can discuss and review the potential changes with collaborators and add follow-up commits before changes are merged into the base branch.

If no concerns are raised, you must first merge the changes to the dev branch. This merge triggers an infrastructure deployment to the dev environment, allowing you to test this environment. After you have tested and are confident about what was deployed, you must merge the dev branch into the prod branch to trigger the infrastructure installation to the production environment.

**Objectives**

* Set up your GitHub repository.
* Configure Terraform to store state in a Cloud Storage bucket.
* Grant permissions to your Cloud Build service account.
* Connect Cloud Build to your GitHub repository.
* Change your environment configuration in a feature branch.
* Promote changes to the development environment.
* Promote changes to the production environment.

**Costs**

This tutorial uses the following billable components of Google Cloud:

* [Cloud Build](https://cloud.google.com/build/pricing)
* [Cloud Storage](https://cloud.google.com/storage/pricing)
* [Compute Engine](https://cloud.google.com/compute/pricing)

To generate a cost estimate based on your projected usage, use the [pricing calculator](https://cloud.google.com/products/calculator).

When you finish this handson, you can avoid continued billing by deleting the resources you created. For more information, see [Clean up](https://cloud.google.com/architecture/managing-infrastructure-as-code#clean-up).

**Before you begin**

1. In the Google Cloud console, on the project selector page, select or [create a Google Cloud project](https://cloud.google.com/resource-manager/docs/creating-managing-projects).

**Note**: If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

[Go to project selector](https://console.cloud.google.com/projectselector2/home/dashboard)

1. Make sure that billing is enabled for your Cloud project. Learn how to [check if billing is enabled on a project](https://cloud.google.com/billing/docs/how-to/verify-billing-enabled).
2. In the Google Cloud console, activate Cloud Shell.

[Activate Cloud Shell](https://console.cloud.google.com/?cloudshell=true)

At the bottom of the Google Cloud console, a [Cloud Shell](https://cloud.google.com/shell/docs/features) session starts and displays a command-line prompt. Cloud Shell is a shell environment with the Google Cloud CLI already installed and with values already set for your current project. It can take a few seconds for the session to initialize.

1. In Cloud Shell, get the ID of the project you just selected:

gcloud config get-value project

If this command doesn't return the project ID, configure Cloud Shell to use your project. Replace *PROJECT\_ID* with your project ID.

gcloud config set project *PROJECT\_ID*

1. Enable the required APIs:

gcloud services enable cloudbuild.googleapis.com compute.googleapis.com

This step might take a few minutes to finish.

1. If you've never used Git in Cloud Shell, configure it with your name and email address:
2. git config --global user.email "*YOUR\_EMAIL\_ADDRESS*"
3. git config --global user.name "*YOUR\_NAME*"

Git uses this information to identify you as the author of the commits that you create in Cloud Shell.

**Setting up your GitHub repository**

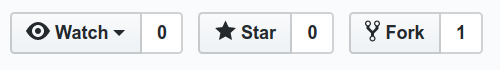
In this tutorial, you use a single Git repository to define your cloud infrastructure. You orchestrate this infrastructure by having different branches corresponding to different environments:

* The dev branch contains the latest changes that are applied to the development environment.
* The prod branch contains the latest changes that are applied to the production environment.

With this infrastructure, you can always reference the repository to know what configuration is expected in each environment and to propose new changes by first merging them into the dev environment. You then promote the changes by merging the dev branch into the subsequent prod branch.

To get started, you fork the [solutions-terraform-cloudbuild-gitops](https://github.com/GoogleCloudPlatform/solutions-terraform-cloudbuild-gitops.git) repository.

1. On GitHub, navigate to <https://github.com/GoogleCloudPlatform/solutions-terraform-cloudbuild-gitops.git>.
2. In the top-right corner of the page, click **Fork**.



Now you have a copy of the solutions-terraform-cloudbuild-gitops repository with source files.

1. In Cloud Shell, clone this forked repository, replacing *YOUR\_GITHUB\_USERNAME* with your GitHub username:
2. cd ~
3. git clone https://github.com/*YOUR\_GITHUB\_USERNAME*/solutions-terraform-cloudbuild-gitops.git
4. cd ~/solutions-terraform-cloudbuild-gitops

The code in this repository is structured as follows:

* The environments/ folder contains subfolders that represent environments, such as dev and prod, which provide logical separation between workloads at different stages of maturity, development and production, respectively. Although it's a good practice to have these environments as similar as possible, each subfolder has its own Terraform configuration to ensure they can have unique settings as necessary.
* The modules/ folder contains inline Terraform modules. These modules represent logical groupings of related resources and are used to share code across different environments.
* The cloudbuild.yaml file is a build configuration file that contains instructions for Cloud Build, such as how to perform tasks based on a set of steps. This file specifies a conditional execution depending on the branch Cloud Build is fetching the code from, for example:
  + For dev and prod branches, the following steps are executed:
    1. terraform init
    2. terraform plan
    3. terraform apply
  + For any other branch, the following steps are executed:
    1. terraform init for all environments subfolders
    2. terraform plan for all environments subfolders

To ensure that the changes being proposed are appropriate for every environment, terraform init and terraform plan are run for all environments subfolders. Before merging the pull request, you can review the plans to make sure that access isn't being granted to an unauthorized entity, for example.

**Configuring Terraform to store state in a Cloud Storage bucket**

By default, Terraform stores [state](https://www.terraform.io/docs/state/) locally in a file named terraform.tfstate. This default configuration can make Terraform usage difficult for teams, especially when many users run Terraform at the same time and each machine has its own understanding of the current infrastructure.

To help you avoid such issues, this section configures a [remote state](https://www.terraform.io/docs/state/remote.html) that points to a Cloud Storage bucket. Remote state is a feature of [backends](https://www.terraform.io/docs/backends)

# Copyright 2019 Google LLC  
#  
# Licensed under the Apache License, Version 2.0 (the "License");  
# you may not use this file except in compliance with the License.  
# You may obtain a copy of the License at  
#  
#     https://www.apache.org/licenses/LICENSE-2.0  
#  
# Unless required by applicable law or agreed to in writing, software  
# distributed under the License is distributed on an "AS IS" BASIS,  
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
# See the License for the specific language governing permissions and  
# limitations under the License.  
  
  
terraform {  
  backend "gcs" {  
    bucket = "PROJECT\_ID-tfstate"  
    prefix = "env/dev"  
  }  
}

In the following steps, you create a Cloud Storage bucket and change a few files to point to your new bucket and your Google Cloud project.

1. In Cloud Shell, create the Cloud Storage bucket:

PROJECT\_ID=$(gcloud config get-value project)  
gsutil mb gs://${PROJECT\_ID}-tfstate

1. Enable [Object Versioning](https://cloud.google.com/storage/docs/object-versioning) to keep the history of your deployments:

gsutil versioning set on gs://${PROJECT\_ID}-tfstate

Enabling Object Versioning increases [storage costs](https://cloud.google.com/storage/pricing), which you can mitigate by configuring [Object Lifecycle Management](https://cloud.google.com/storage/docs/lifecycle) to delete old state versions.

1. Replace the PROJECT\_ID placeholder with the project ID in both the terraform.tfvars and backend.tf files:
2. cd ~/solutions-terraform-cloudbuild-gitops
3. sed -i s/PROJECT\_ID/$PROJECT\_ID/g environments/\*/terraform.tfvars
4. sed -i s/PROJECT\_ID/$PROJECT\_ID/g environments/\*/backend.tf

On OS X/MacOS, you might need to add two quotation marks ("") after sed -i, as follows:

cd ~/solutions-terraform-cloudbuild-gitops

sed -i "" s/PROJECT\_ID/$PROJECT\_ID/g environments/\*/terraform.tfvars

sed -i "" s/PROJECT\_ID/$PROJECT\_ID/g environments/\*/backend.tf

1. Check whether all files were updated:

git status

The output looks like this:

On branch dev

Your branch is up-to-date with 'origin/dev'.

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: environments/dev/backend.tf

modified: environments/dev/terraform.tfvars

modified: environments/prod/backend.tf

modified: environments/prod/terraform.tfvars

no changes added to commit (use "git add" and/or "git commit -a")

1. Commit and push your changes:

git add --all  
git commit -m "Update project IDs and buckets"  
git push origin dev

Depending on your GitHub configuration, you will have to authenticate to push the preceding changes.

**Granting permissions to your Cloud Build service account**

To allow [Cloud Build service account](https://cloud.google.com/build/docs/securing-builds/set-service-account-permissions) to run Terraform scripts with the goal of managing Google Cloud resources, you need to grant it appropriate access to your project. For simplicity, [project editor](https://cloud.google.com/iam/docs/understanding-roles#basic) access is granted in this tutorial. But when the project editor role has a wide-range permission, in production environments you must follow your company's IT security best practices, usually providing [least-privileged access](https://cloud.google.com/docs/enterprise/best-practices-for-enterprise-organizations#control-access).

1. In Cloud Shell, retrieve the email for your project's Cloud Build service account:

CLOUDBUILD\_SA="$(gcloud projects describe $PROJECT\_ID \  
    --format 'value(projectNumber)')@cloudbuild.gserviceaccount.com"

1. Grant the required access to your Cloud Build service account:

gcloud projects add-iam-policy-binding $PROJECT\_ID \  
    --member serviceAccount:$CLOUDBUILD\_SA --role roles/editor

**Directly connecting Cloud Build to your GitHub repository**

This section shows you how to install the [Cloud Build GitHub app](https://github.com/marketplace/google-cloud-build). This installation allows you to connect your GitHub repository with your Google Cloud project so that Cloud Build can automatically apply your Terraform manifests each time you create a new branch or push code to GitHub.

The following steps provide instructions for installing the app only for the solutions-terraform-cloudbuild-gitops repository, but you can choose to install the app for more or all of your repositories.

1. Go to the GitHub Marketplace page for the Cloud Build app:

[Open the Cloud Build app page](https://github.com/marketplace/google-cloud-build)

* + If this is your first time configuring an app in GitHub: Click **Setup with Google Cloud Build** at the bottom of the page. Then click **Grant this app access to your GitHub account**.
  + If this is not the first time configuring an app in GitHub: Click **Configure access**. The **Applications** page of your personal account opens.

1. Click **Configure** in the Cloud Build row.
2. Select **Only select repositories**, then select **solutions-terraform-cloudbuild-gitops** to connect to the repository.
3. Click **Save** or **Install**—the button label changes depending on your workflow. You are redirected to Google Cloud to continue the installation.
4. Sign in with your Google Cloud account. If requested, authorize Cloud Build integration with GitHub.
5. On the **Cloud Build** page, select your project. A wizard appears.
6. In the **Select repository** section, select your GitHub account and the **solutions-terraform-cloudbuild-gitops** repository.
7. If you agree with the terms and conditions, select the checkbox, then click **Connect**.
8. In the **Create a trigger** section, click **Create a trigger**:
   * Add a trigger name, such as push-to-branch. Note this trigger name because you will need it later.
   * In the **Event** section, select **Push to a branch**.
   * In the **Source** section, select .\* in the **Branch** field.
   * Click **Create**.

The Cloud Build GitHub app is now configured, and your GitHub repository is linked to your Google Cloud project. From now on, changes to the GitHub repository trigger Cloud Build executions, which report the results back to GitHub by using [GitHub Checks](https://developer.github.com/v3/checks/).

**Changing your environment configuration in a new feature branch**

By now, you have most of your environment configured. So it's time to make some code changes in your development environment.

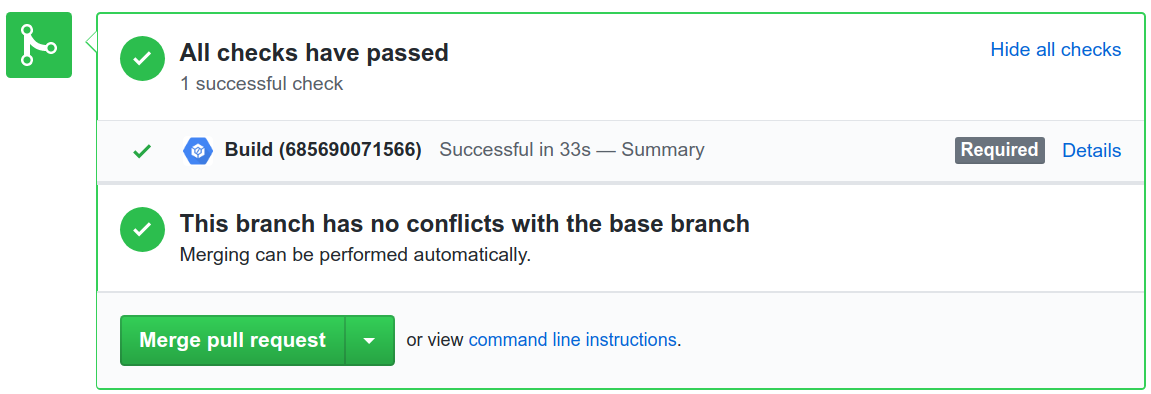
1. On GitHub, navigate to the main page of your forked repository.
2. https://github.com/*YOUR\_GITHUB\_USERNAME*/solutions-terraform-cloudbuild-gitops
3. Make sure you are in the dev branch.
4. To open the file for editing, go to the modules/firewall/main.tf file and click the pencil icon.
5. On line 30, fix the "http-server2" typo in target\_tags field.

The value must be "http-server".

1. Add a commit message at the bottom of the page, such as "Fixing http firewall target", and select **Create a new branch for this commit and start a pull request**.
2. Click **Propose changes**.
3. On the following page, click **Create pull request** to open a new pull request with your change.

After your pull request is open, a Cloud Build job is automatically initiated.

1. Click **Show all checks** and wait for the check to become green.



1. Click **Details** to see more information, including the output of the terraform plan at **View more details on Google Cloud Build** link.

Don't merge your pull request yet.

Note that the Cloud Build job ran the pipeline defined in the cloudbuild.yaml file. As discussed previously, this pipeline has different behaviors depending on the branch being fetched. The build checks whether the $BRANCH\_NAME variable matches any environment folder. If so, Cloud Build executes terraform plan for that environment. Otherwise, Cloud Build executes terraform plan for all environments to make sure that the proposed change is appropriate for all of them. If any of these plans fail to execute, the build fails.

[cloudbuild.yaml](https://github.com/GoogleCloudPlatform/solutions-terraform-cloudbuild-gitops/blob/HEAD/cloudbuild.yaml)

[View on GitHub](https://github.com/GoogleCloudPlatform/solutions-terraform-cloudbuild-gitops/blob/HEAD/cloudbuild.yaml)

- id: 'tf plan'  
  name: 'hashicorp/terraform:1.0.0'  
  entrypoint: 'sh'  
  args:   
  - '-c'  
  - |   
      if [ -d "environments/$BRANCH\_NAME/" ]; then  
        cd environments/$BRANCH\_NAME  
        terraform plan  
      else  
        for dir in environments/\*/  
        do   
          cd ${dir}     
          env=${dir%\*/}  
          env=${env#\*/}    
          echo ""  
          echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TERRAFOM PLAN \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"  
          echo "\*\*\*\*\*\*\* At environment: ${env} \*\*\*\*\*\*\*\*"  
          echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"  
          terraform plan || exit 1  
          cd ../../  
        done  
      fi

Similarly, the terraform apply command runs for environment branches, but it is completely ignored in any other case. In this section, you have submitted a code change to a new branch, so no infrastructure deployments were applied to your Google Cloud project.

[cloudbuild.yaml](https://github.com/GoogleCloudPlatform/solutions-terraform-cloudbuild-gitops/blob/HEAD/cloudbuild.yaml)

[View on GitHub](https://github.com/GoogleCloudPlatform/solutions-terraform-cloudbuild-gitops/blob/HEAD/cloudbuild.yaml)

- id: 'tf apply'  
  name: 'hashicorp/terraform:1.0.0'  
  entrypoint: 'sh'  
  args:   
  - '-c'  
  - |   
      if [ -d "environments/$BRANCH\_NAME/" ]; then  
        cd environments/$BRANCH\_NAME        
        terraform apply -auto-approve  
      else  
        echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* SKIPPING APPLYING \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"  
        echo "Branch '$BRANCH\_NAME' does not represent an oficial environment."  
        echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"  
      fi

**Enforcing Cloud Build execution success before merging branches**

To make sure merges can be applied only when respective Cloud Build executions are successful, proceed with the following steps:

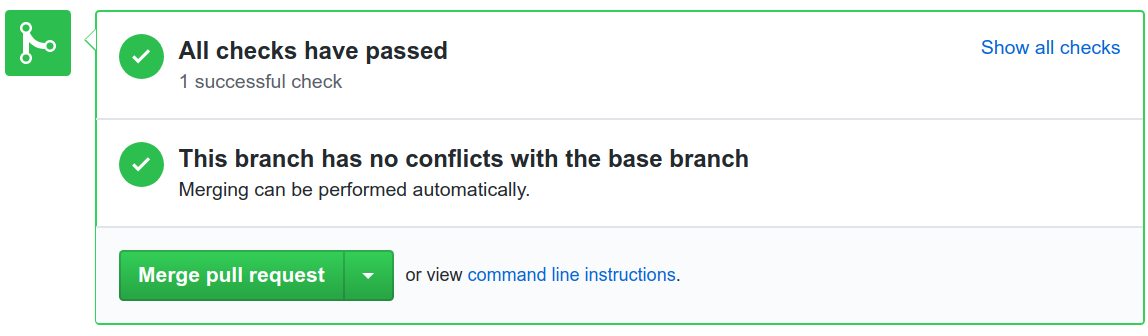
1. On GitHub, navigate to the main page of your forked repository.
2. https://github.com/*YOUR\_GITHUB\_USERNAME*/solutions-terraform-cloudbuild-gitops
3. Under your repository name, click **Settings**.
4. In the left menu, click **Branches**.
5. Under **Branch protection rules**, click **Add rule**.
6. In **Branch name pattern**, type dev.
7. In the **Protect matching branches** section, select **Require status checks to pass before merging**.
8. Search for your Cloud Build trigger name created previously.
9. Click **Create**.
10. Repeat steps 3–7, setting **Branch name pattern** to prod.

This configuration is important to [protect](https://help.github.com/en/articles/about-protected-branches) both the dev and prod branches. Meaning, commits must first be pushed to another branch, and only then they can be merged to the protected branch. In this tutorial, the protection requires that the Cloud Build execution be successful for the merge to be allowed.

**Promoting changes to the development environment**

You have a pull request waiting to be merged. It's time to apply the state you want to your dev environment.

1. On GitHub, navigate to the main page of your forked repository.
2. https://github.com/*YOUR\_GITHUB\_USERNAME*/solutions-terraform-cloudbuild-gitops
3. Under your repository name, click **Pull requests**.
4. Click the pull request you just created.
5. Click **Merge pull request**, and then click **Confirm merge**.



1. Check that a new Cloud Build has been triggered:

[Go to the Cloud Build page](https://console.cloud.google.com/cloud-build/builds)

1. Open the build and check the logs.

When the build finishes, you see something like this:

Step #3 - "tf apply": external\_ip = *EXTERNAL\_IP\_VALUE*

Step #3 - "tf apply": firewall\_rule = dev-allow-http

Step #3 - "tf apply": instance\_name = dev-apache2-instance

Step #3 - "tf apply": network = dev

Step #3 - "tf apply": subnet = dev-subnet-01

1. Copy *EXTERNAL\_IP\_VALUE* and open the address in a web browser.
2. http://*EXTERNAL\_IP\_VALUE*

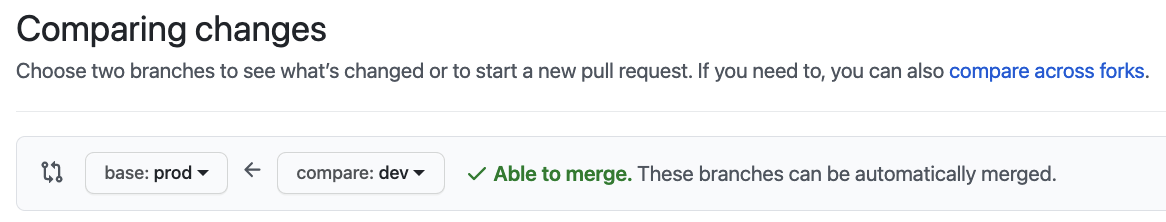
This provisioning might take a few seconds to boot the VM and to propagate the firewall rule. Eventually, you see **Environment: dev** in the web browser.

1. Navigate to your Terraform state file in your Cloud Storage bucket.
2. https://storage.cloud.google.com/*PROJECT\_ID*-tfstate/env/dev/default.tfstate

**Promoting changes to the production environment**

Now that you have your development environment fully tested, you can promote your infrastructure code to production.

1. On GitHub, navigate to the main page of your forked repository.
2. https://github.com/*YOUR\_GITHUB\_USERNAME*/solutions-terraform-cloudbuild-gitops
3. Under your repository name, click **Pull requests**.
4. Click **New pull request**.
5. For the **base repository**, select your just-forked repository.
6. For **base**, select prod from your own base repository. For **compare**, select dev.



1. Click **Create pull request**.
2. For **title**, enter a title such as Promoting networking changes, and then click **Create pull request**.
3. Review the proposed changes, including the terraform plan details from Cloud Build, and then click **Merge pull request**.
4. Click **Confirm merge**.
5. In the Google Cloud console, open the **Build History** page to see your changes being applied to the production environment:

[Go to the Cloud Build page](https://console.cloud.google.com/cloud-build/builds)

1. Wait for the build to finish, and then check the logs.

At the end of the logs, you see something like this:

Step #3 - "tf apply": external\_ip = *EXTERNAL\_IP\_VALUE*

Step #3 - "tf apply": firewall\_rule = prod-allow-http

Step #3 - "tf apply": instance\_name = prod-apache2-instance

Step #3 - "tf apply": network = prod

Step #3 - "tf apply": subnet = prod-subnet-01

1. Copy *EXTERNAL\_IP\_VALUE* and open the address in a web browser.
2. http://*EXTERNAL\_IP\_VALUE*

This provisioning might take a few seconds to boot the VM and to propagate the firewall rule. Eventually, you see **Environment: prod** in the web browser.

1. Navigate to your Terraform state file in your Cloud Storage bucket.
2. https://storage.cloud.google.com/*PROJECT\_ID*-tfstate/env/prod/default.tfstate

You have successfully configured a serverless infrastructure-as-code pipeline on Cloud Build. In the future, you might want to try the following:

* Add deployments for separate use cases.
* Create additional environments to reflect your needs.
* Use a project per environment instead of a VPC per environment.

**Clean up**

After you've finished the tutorial, clean up the resources you created on Google Cloud so you won't be billed for them in the future.

**Deleting the project**

**Caution**: Deleting a project has the following effects:

* + **Everything in the project is deleted.** If you used an existing project for this tutorial, when you delete it, you also delete any other work you've done in the project.
  + **Custom project IDs are lost.** When you created this project, you might have created a custom project ID that you want to use in the future. To preserve the URLs that use the project ID, such as an **appspot.com** URL, delete selected resources inside the project instead of deleting the whole project.

If you plan to explore multiple tutorials and quickstarts, reusing projects can help you avoid exceeding project quota limits.

1. In the Google Cloud console, go to the **Manage resources** page.

[Go to Manage resources](https://console.cloud.google.com/iam-admin/projects)

1. In the project list, select the project that you want to delete, and then click **Delete**.
2. In the dialog, type the project ID, and then click **Shut down** to delete the project.

**Deleting the GitHub repository**

To avoid blocking new pull requests on your GitHub repository, you can delete your branch protection rules:

1. In GitHub, navigate to the main page of your forked repository.
2. Under your repository name, click **Settings**.
3. In the left menu, click **Branches**.
4. Under the **Branch protection rules** section, click the **Delete** button for both dev and prod rows.

Optionally, you can completely uninstall the Cloud Build app from GitHub:

1. Go to your GitHub **Applications** settings.

[Go to the GitHub applications page](https://github.com/settings/installations)

1. In the **Installed GitHub Apps** tab, click **Configure** in the **Cloud Build** row. Then, in the **Danger zone** section, click the **Uninstall** button in the **Uninstall Google Cloud Builder** row.

At the top of the page, you see a message saying "You're all set. A job has been queued to uninstall Google Cloud Build."

1. In the **Authorized GitHub Apps** tab, click the **Revoke** button in the **Google Cloud Build** row, then **I understand, revoke access** in the popup.

If you don't want to keep your GitHub repository:

1. In GitHub, go to the main page of your forked repository.
2. Under your repository name, click **Settings**.
3. Scroll down to the **Danger Zone**.
4. Click **Delete this repository**, and follow the confirmation steps.