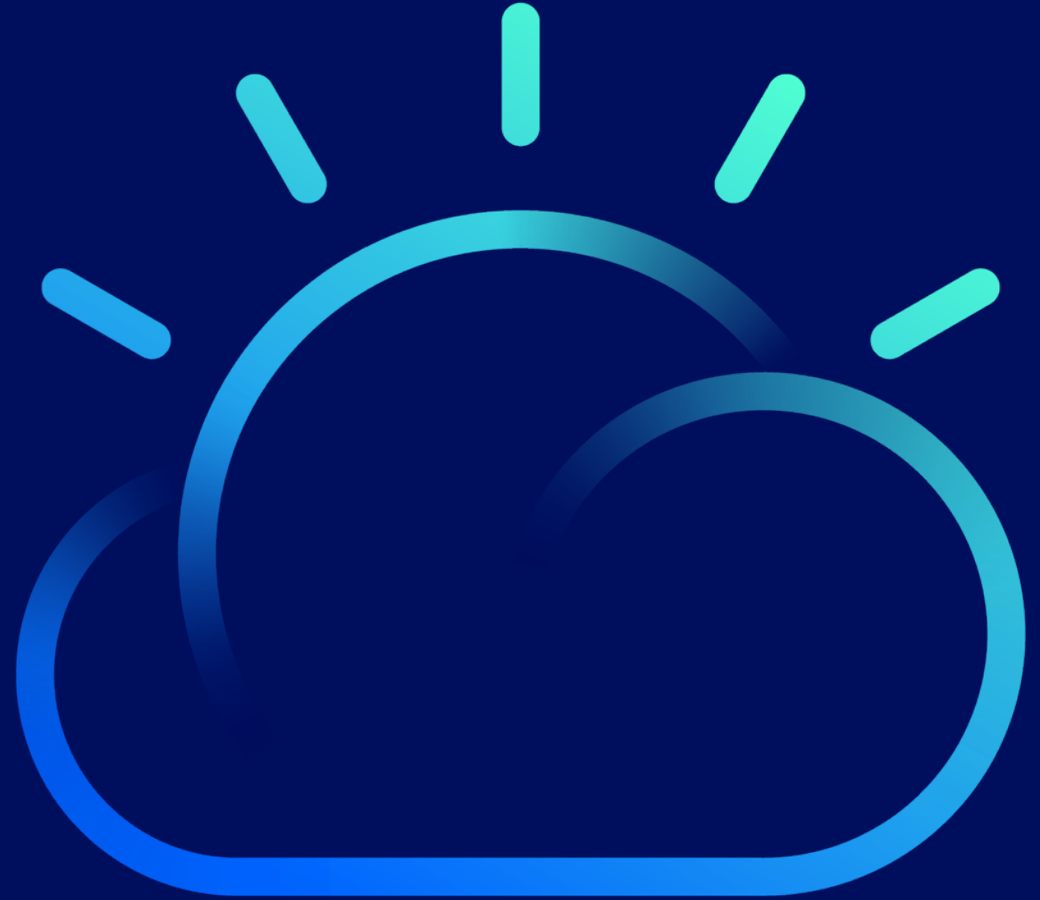


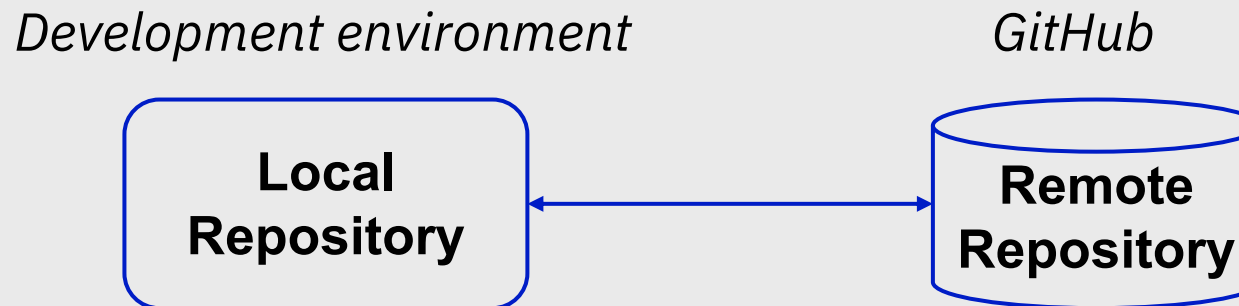
Working with GIT

Collaborative Infrastructure

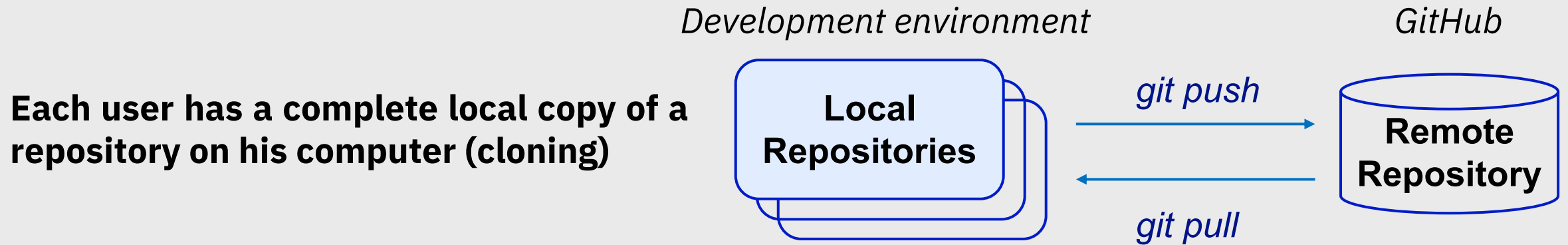


Why using a version control system such as Git ?

- Access to all versions of all terraform files at any time
- You don't lose a piece of code from a previous state of your environment
- Collaborate and work as a team without interfering with each other
- Terraform plan gives an opportunity for team members to review what has been done by each other

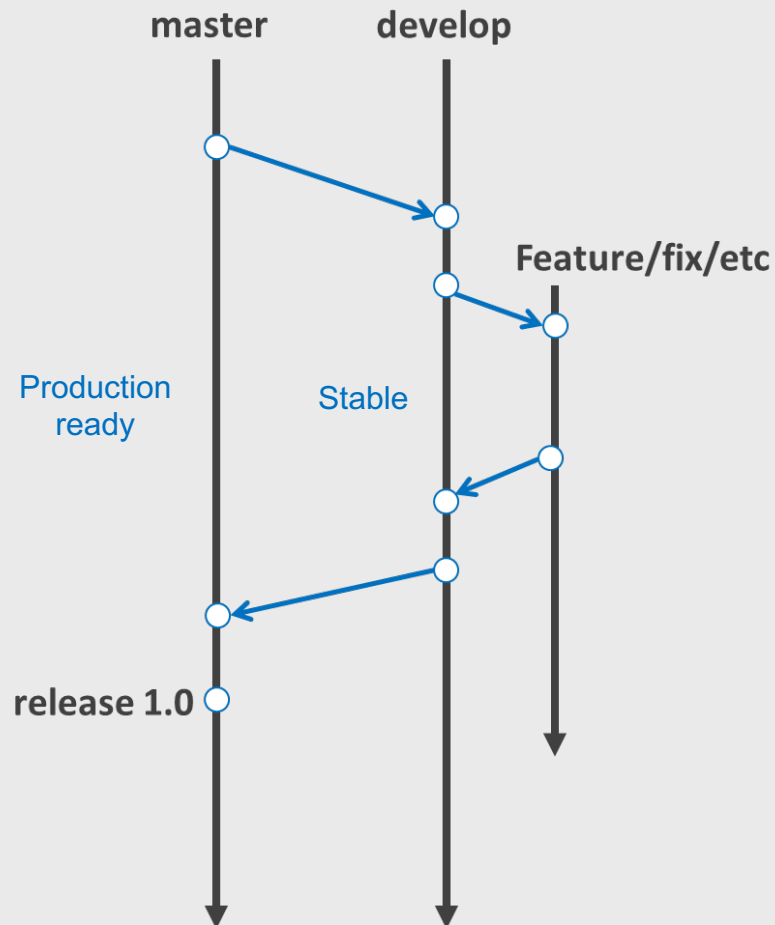


Distributed version control system



- The code is developed on each local computer
- GitHub is a central remote point hosting repositories and enabling users to obtain, alter and integrate changes through Git
- You have to setup a Git account as an hosting utility
- You can create a repository locally with the command `git.init`
- Or you can create a repository on GitHub (web ui) and clone it locally using the command `git.clone`

The concept of branches



A branch allows the user to switch between different versions of a collection of files so that he can work on different changes independently from each other.

Branches in Git are local to the repository.

The master branch is the default branch automatically created when cloning a repository

The command *git checkout* allows to select a branch (switch to a branch)

Why branches ?

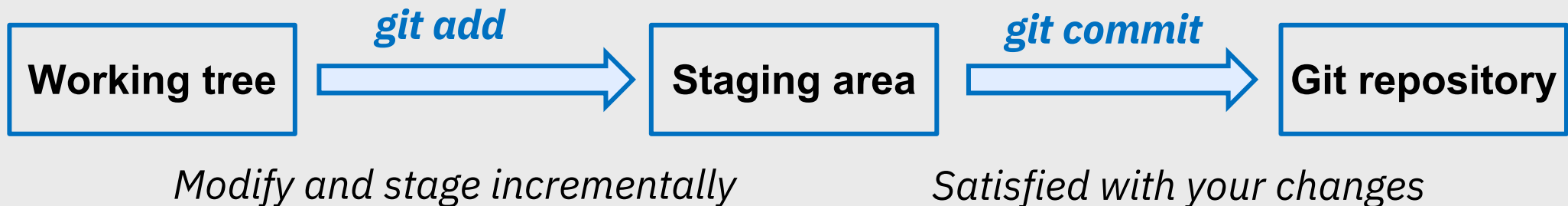
- Never commit on Master, instead uses branches created from the master and merge from your working branches
- Old code untouched until the new one works

Working tree, staging area and repository

In a local repository a working tree is a collection of files which originate from a certain version of the repository.

To persist changes in a working tree you need:

- To add selected changes to the staging area (Index) with the command *git add*
- To commit the staged changes into the git repository with the command *git commit*



A branch points to a specific commit. **HEAD** is a reference to the last commit in the currently check-out branch .

Versioning Commits

- The command *git status* retrieve the status of a file
 - ✓ Untracked: file whose changes are not monitored by Git and not declared in the gitignore file
 - ✓ Unstaged: file whose changes are being tracked by Git and changed since the last commit
 - ✓ Staged: file whose changes are being tracked by Git and ready for the commit
- Run *git add* to stage the changes made to the file
- Run *git rm* to remove files from your staging area and your working directory
- Run *git commit -m "reason for change"* to commit the staged changed
- Use *git diff* to show changes between commits or commit and working tree

Fetch, Merge and Stash

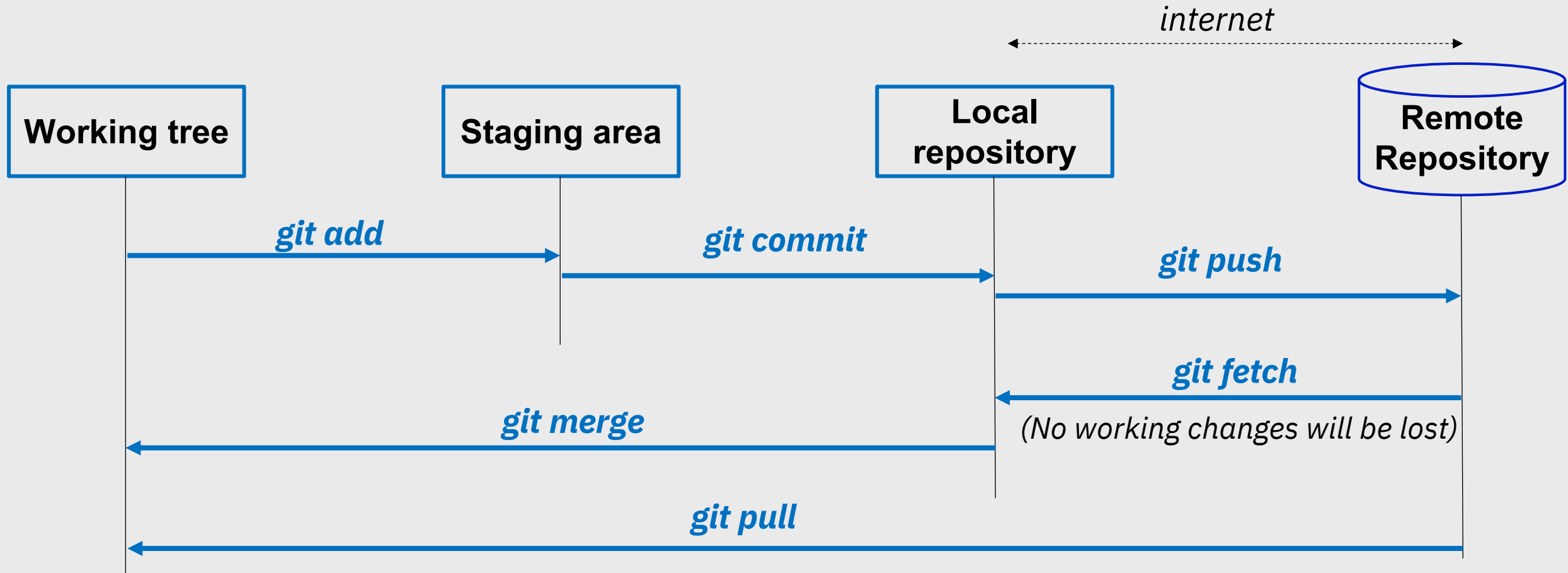
Fetch: See what everybody else has been working on. If you have conflicting files, you will need to manually **merge** them. Git fetch does not touch your working tree. It gives you the chance to decide what to do next.

Merge: When two files have been changed, merge them. After a fetch it brings the changes into your working tree. When merging branches the commit history get merged.

Pull: Fetch + merge

Stash: temporarily save uncommitted local changes and leave you with a clean working copy (*git stash*).... for example when you have to switch context to work on an urgent bug....to continue later on you can restore the saved state (*git stash pop*)

Put all together



Repository Tags

- Tags are used to identify specific significant points on a repository's history
- You tag a specific commit
- Use the command `git tag <tag name>`
- A tag is like a branch which does not change

Configure your environment

You already have a remote repository – Configure a remote repository URL

1 - Add a remote repo url to your local git configuration (create a connection to a remote repository)

➤ *git remote add <remote_name> <remote_repo_url>*

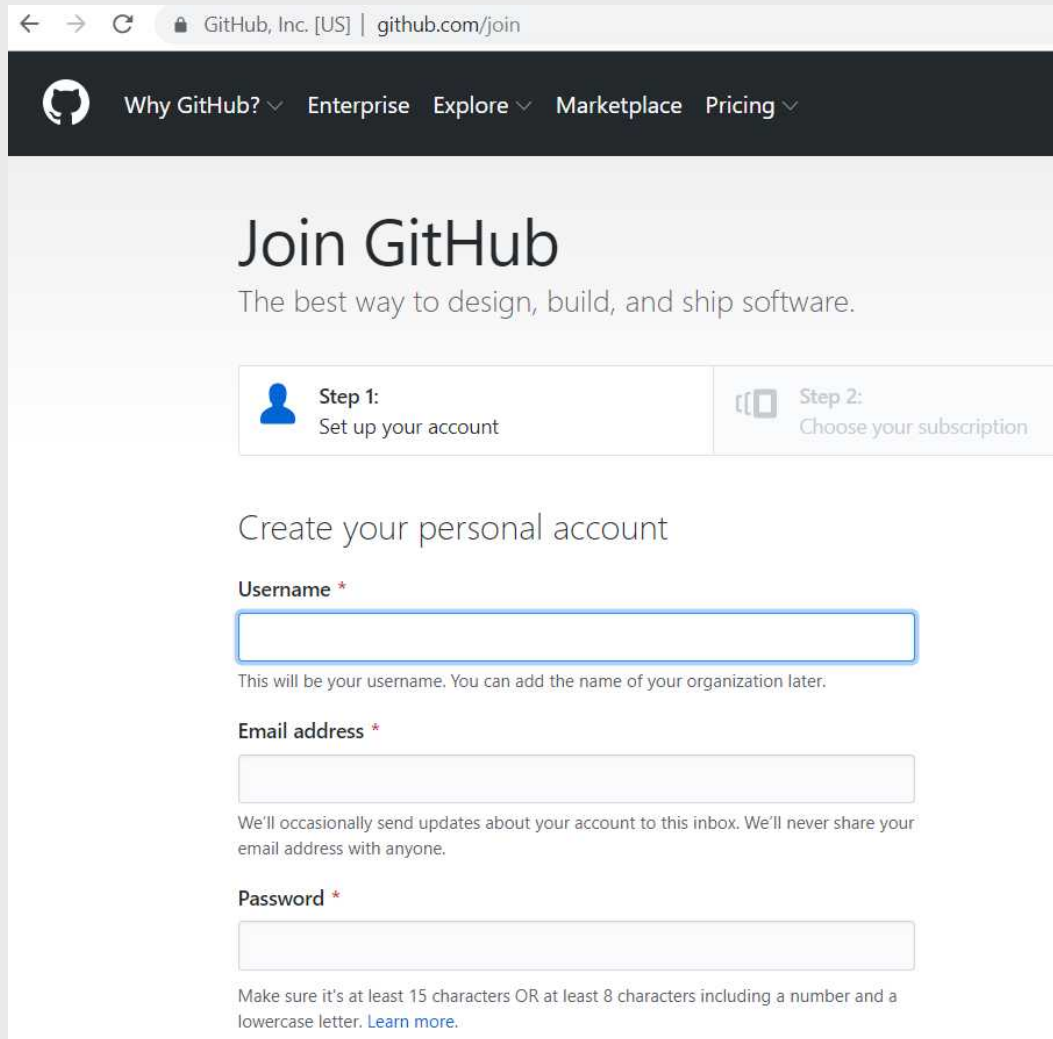
2 - Push a local branch to the remote repository

➤ *git push -u <remote_name> <local_branch_name>*

You can also configure global options such as user name to be used for all commits


➤ *git config --global user.name <name>*

Create a GitHub Account




The screenshot shows the GitHub 'Join GitHub' page. At the top, there's a navigation bar with the GitHub logo and links for 'Why GitHub?', 'Enterprise', 'Explore', 'Marketplace', and 'Pricing'. The main heading is 'Join GitHub' with the tagline 'The best way to design, build, and ship software.' Below this, there are two steps: 'Step 1: Set up your account' (active) and 'Step 2: Choose your subscription'. The 'Create your personal account' section includes three input fields: 'Username', 'Email address', and 'Password', each with a red asterisk indicating it's required. Below the 'Username' field is a note: 'This will be your username. You can add the name of your organization later.' Below the 'Email address' field is a note: 'We'll occasionally send updates about your account to this inbox. We'll never share your email address with anyone.' Below the 'Password' field is a note: 'Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)'


← → ↻ 🔒 GitHub, Inc. [US] | github.com/join

 Why GitHub? ▾ Enterprise Explore ▾ Marketplace Pricing ▾

Join GitHub

The best way to design, build, and ship software.

 **Step 1:**
Set up your account

 **Step 2:**
Choose your subscription

Create your personal account

Username *

This will be your username. You can add the name of your organization later.

Email address *

We'll occasionally send updates about your account to this inbox. We'll never share your email address with anyone.

Password *

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)

What's GitLab ?

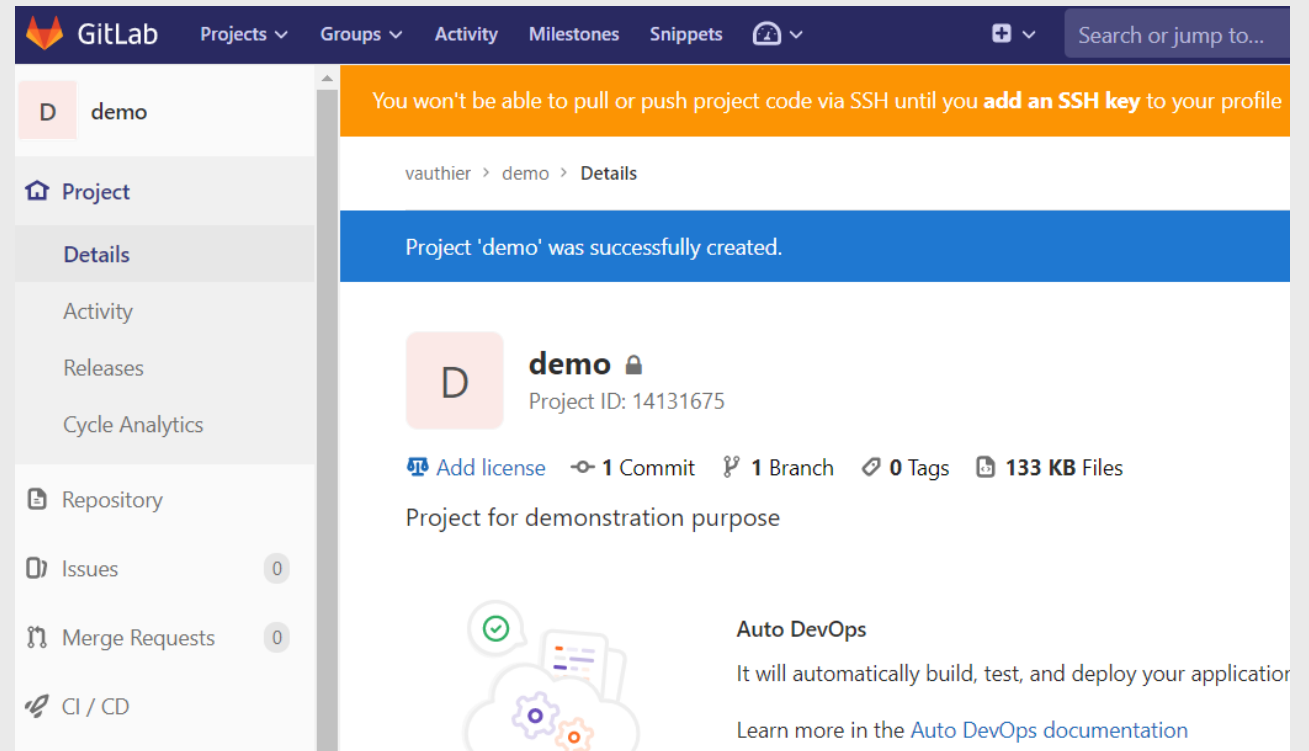
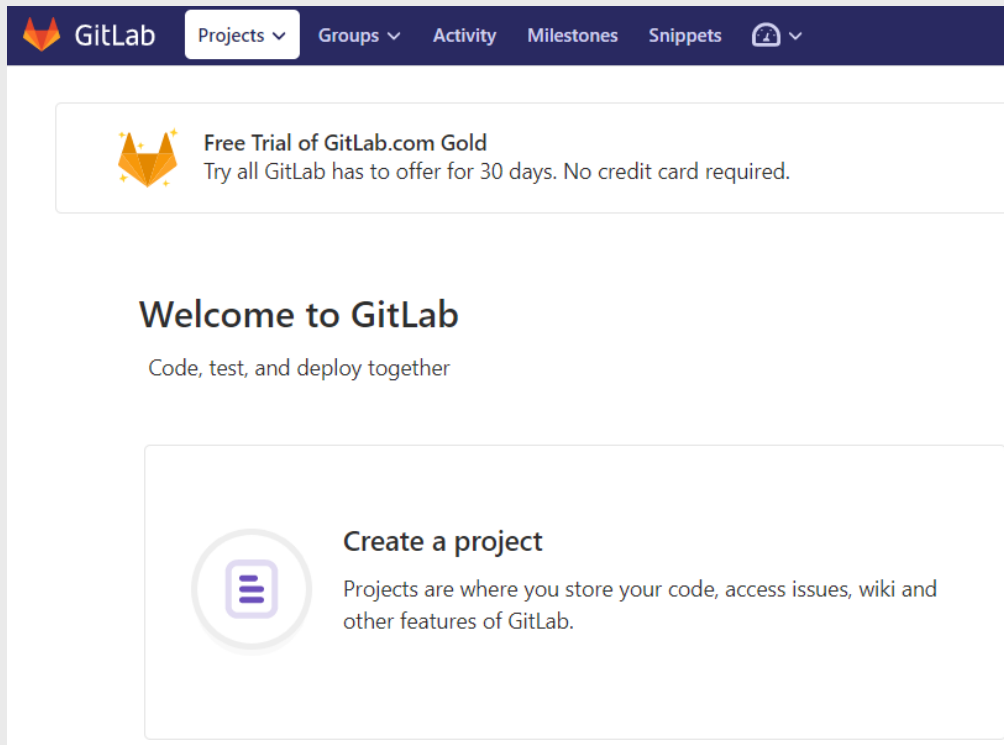
GitLab is built on top of git so that users who are contributing work to a project will have a copy of the project downloaded/checked out/cloned on their local computer.

It provides a web interface for handling many of git's more advanced workflows, and recommends a workflow for interacting with git for the best in productivity, efficiency, and ease of use.

- ✓ File browser to explore the files in your repository
- ✓ A branch viewer
- ✓ A tag viewer
- ✓ Analyze and view the commit graph
- ✓ Webinterface to make changes to code and commit it

What is a GitLab project ?

- A Git repository to host the code with branches
- Several features such as issues tracking, CI/CD features....



GitLab create a project

The screenshot shows the GitLab web interface for a project named 'demo'. The left sidebar contains navigation links: Project, Repository, Files, Commits, Branches (selected), Tags, Contributors, Graph, Compare, and Charts. The top navigation bar includes links for Projects, Groups, Activity, Milestones, Snippets, and a search bar. The main content area is titled 'vauthier > demo > Repository > Branches'. It features tabs for Overview, Active, Stale, and All, with 'Overview' selected. A filter box for 'Filter by branch name' and buttons for 'Delete merged branches' and 'New branch' are present. A message states: 'Protected branches can be managed in [project settings](#).' Below this, the 'Active branches' section shows the 'master' branch as the 'default' and 'protected', with its commit hash '7ad1c315' and the text 'Initial commit · 1 minute ago'. Action icons for refreshing and deleting are visible next to the branch entry.

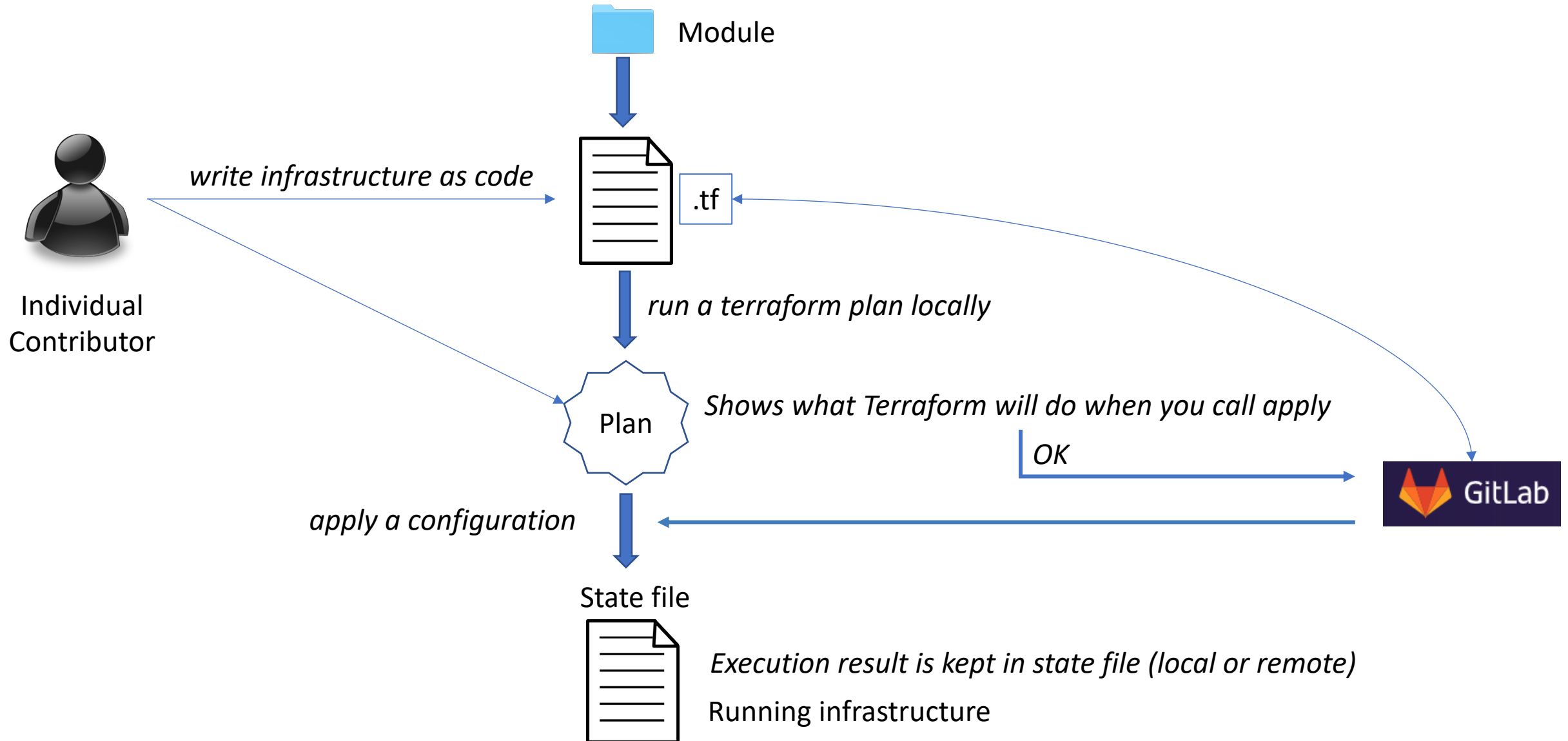
GitLab and GitHub ?

- Both, GitLab and GitHub are web-based Git repositories. A git repository is a central place where developers store, share, test and collaborate.
- Written in Ruby and Go, GitLab offers some similar features for issue tracking and project management as GitHub.
- With GitLab you can set and modify people's permissions according to their role. In GitHub, you can decide if someone gets a read or write access to a repository. With GitLab you can provide access to the issue tracker without giving permission to the source code
- GitLab offers its very own CI for free. No need to use an external CI service.
- GitLab provides service to import and export projects
- GitLab focus more on DevOps providing tools for more efficient workflows
- GitLab can be installed locally

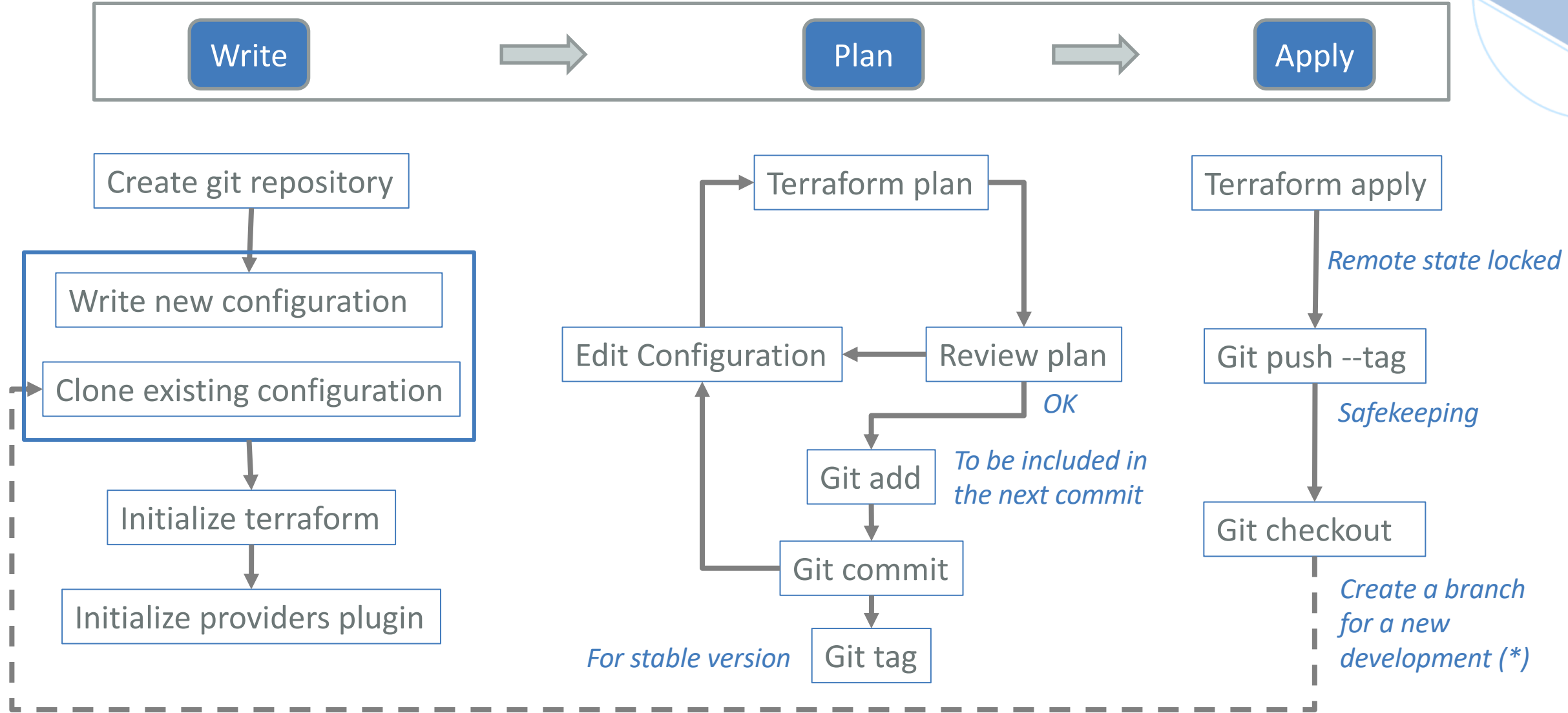
Git, Terraform and Devops

The good the bad and and the ugly

Terraform workflow working with Git



Terraform workflow ... working with Git



(*) Master is the default branch, but several branches can be used such as release, hot fix, feature dev.,... depending on the context

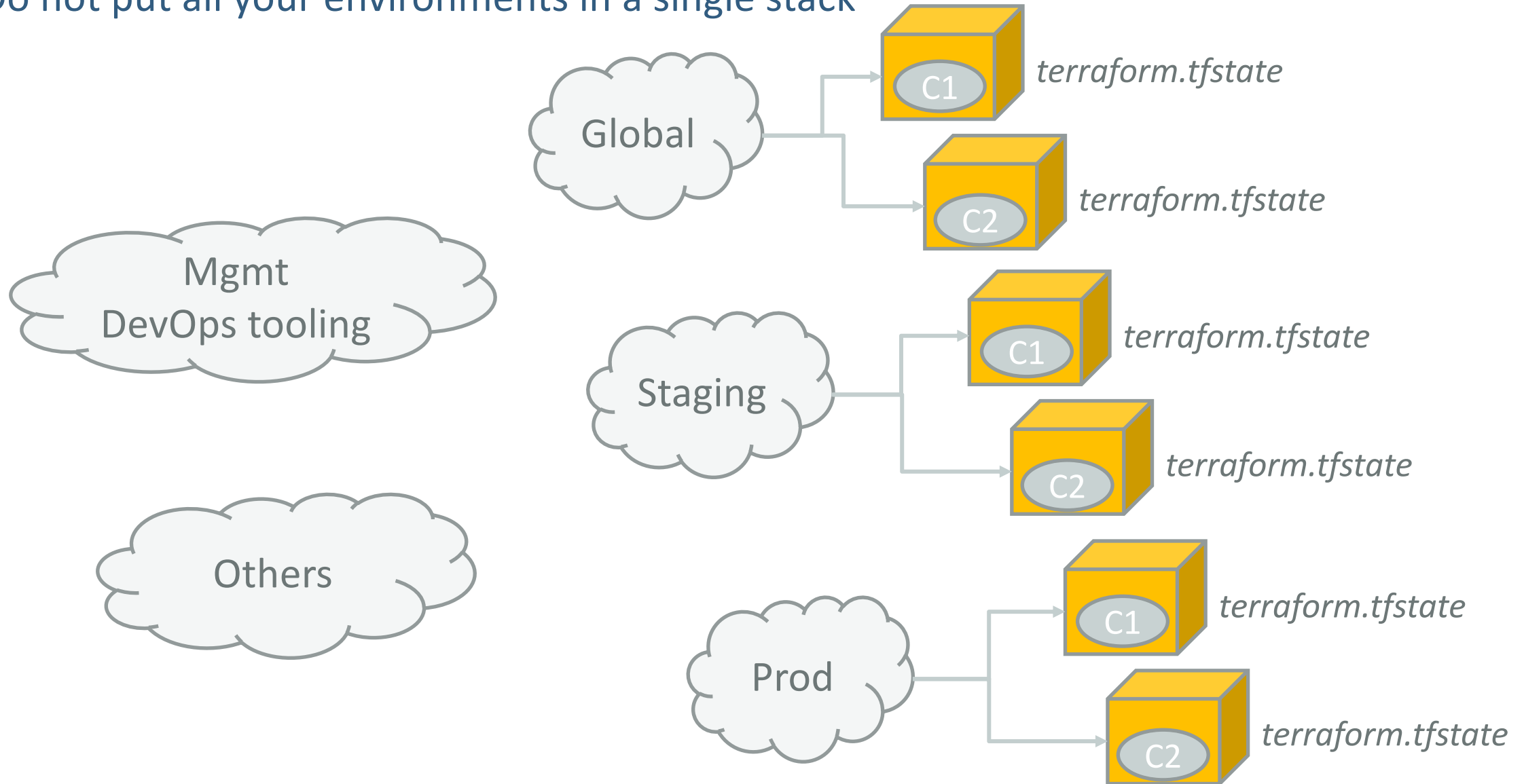
Trouble with branches

- Locking mechanism provided by Terraform backends avoid changes to overwrite in the terraform state file.... but not in the source code
- If 2 team members are deploying the same code to the same environment but from 2 different branches => conflicts can occur
- For one shared environment (*staging, prod,...*) always deploy from a single branch
- After a plan, review the changes. Then when all the tests are passed merge all your changes in the master branch
- Terraform is a declarative language and control over deployment is limited when applying changes. Be prepared to address efficiently situations where something goes wrong. Always test your changes in a staging environment before deploying in production.

Environments isolation

Do not put all your environments in a single stack

Components (...terraform modules)



Pipeline

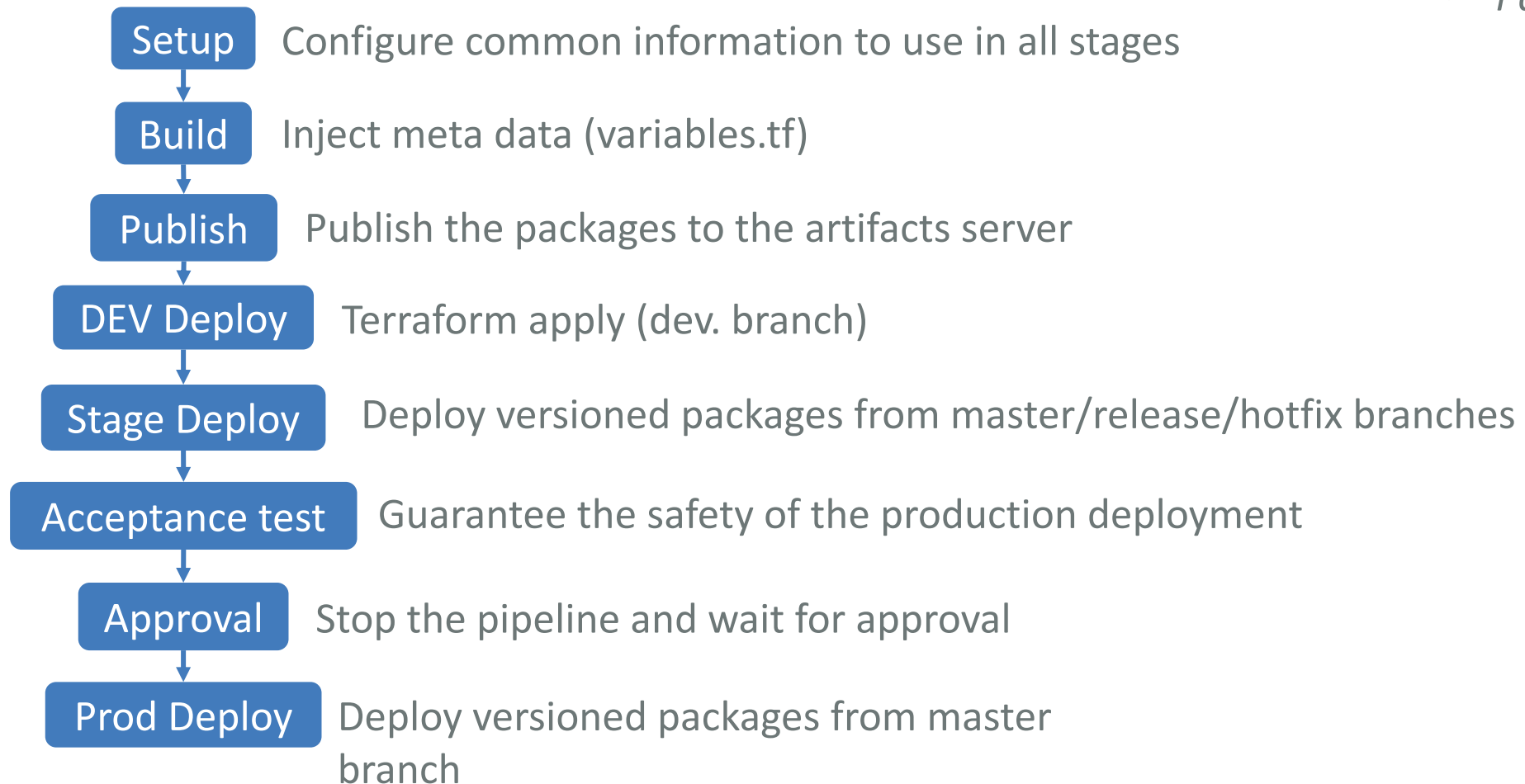


Keep code base consistent

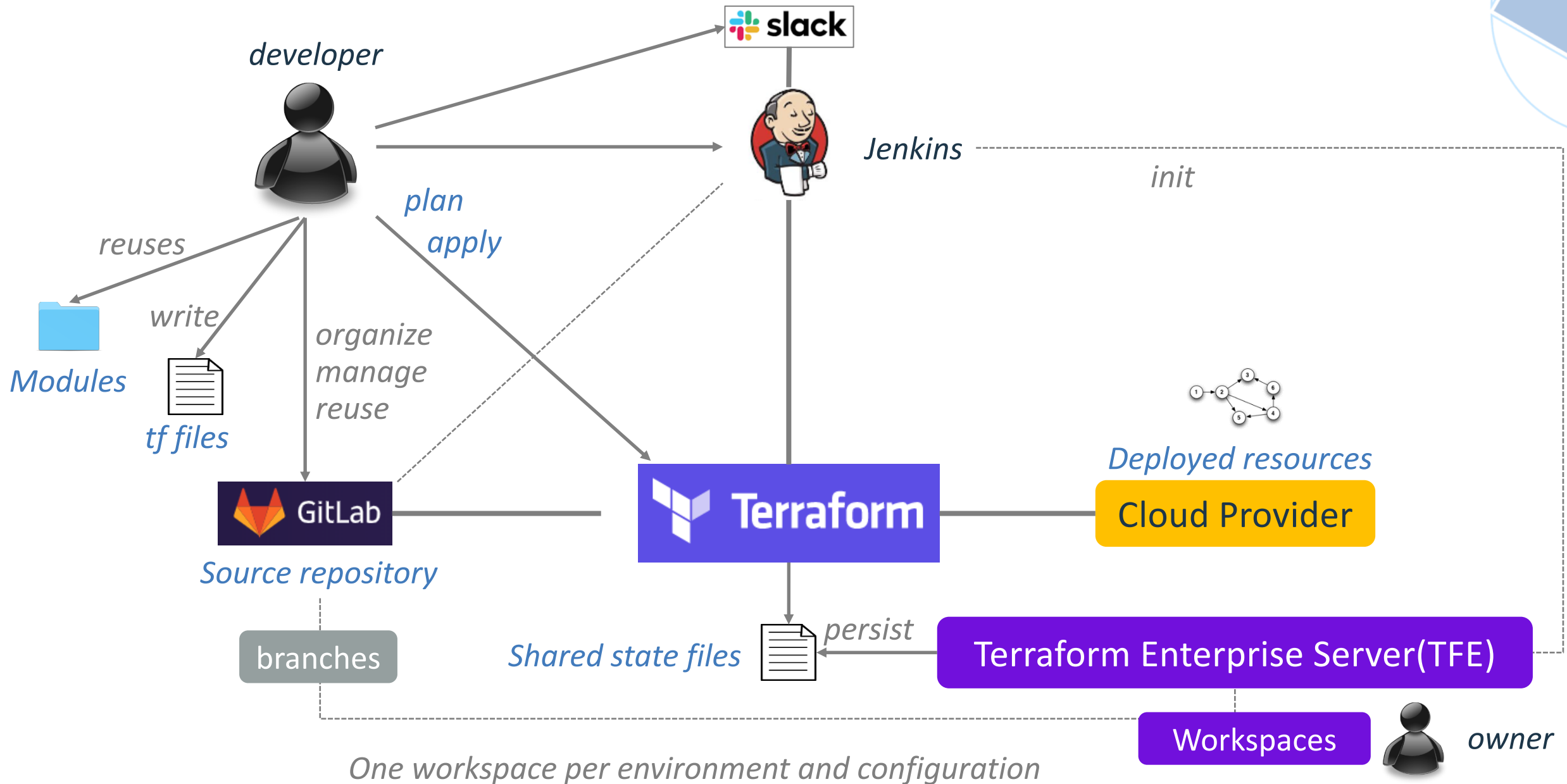
Simple: Define separate infra. component (*1 terraform state file*) for each environment

Complex: Create a single infra. component definition and promote it through a pipeline

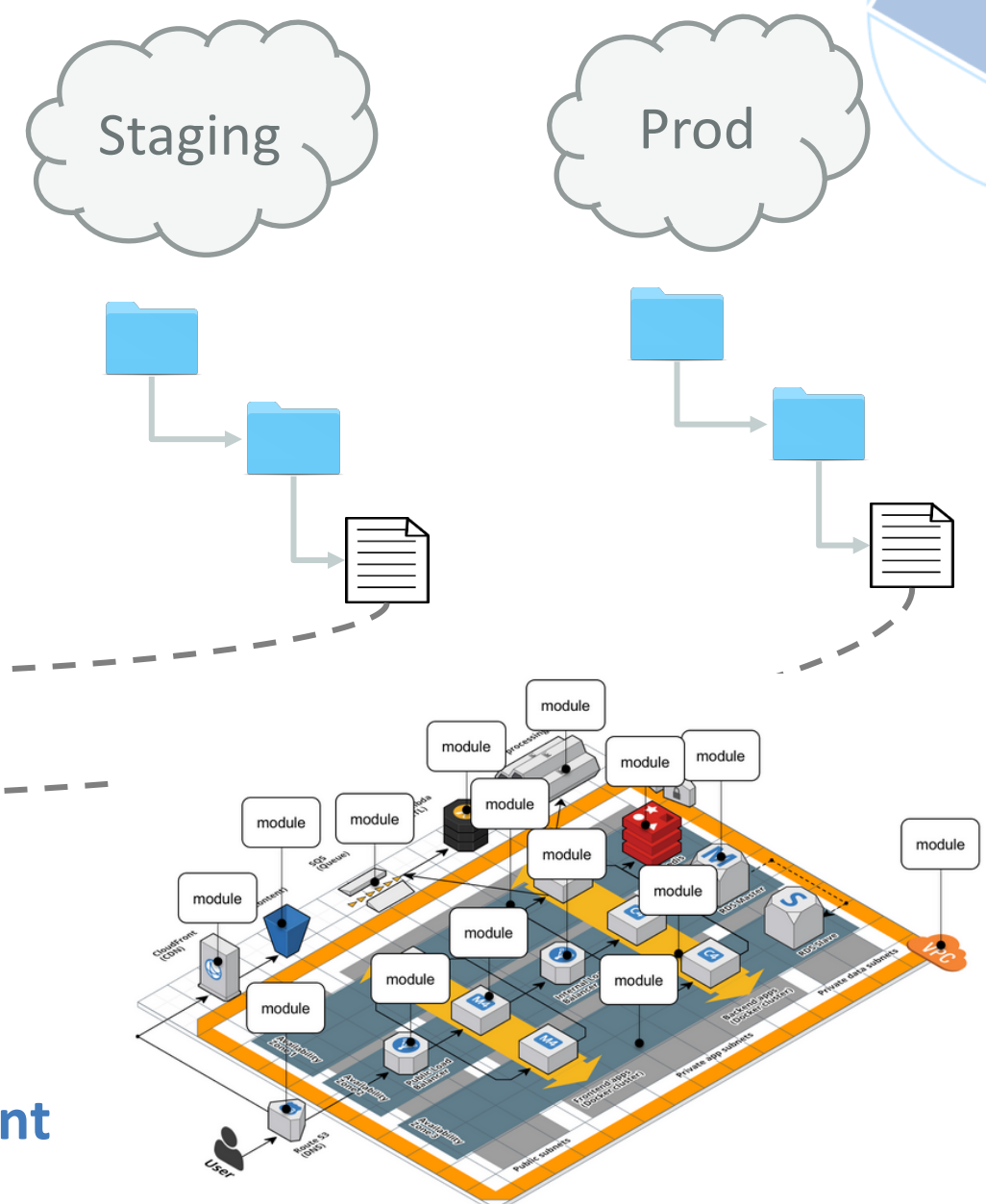
Parametrized



Pipeline tools example



The diagram illustrates the build process of a module. It begins with a folder labeled *Modules*. A sequence of arrows leads through three more folders and a file named *main.tf*. A horizontal line separates this build phase from the release phase, labeled *published & released*. Below this line, two versions of the module are shown as boxes: *V1* and *V2*. A solid arrow points from the build output to *V1*, and another solid arrow points to *V2*. Dashed arrows indicate a sequence from *V2* to *V1*, suggesting that *V1* is the previous or base version.



- ➡ You need a repository for your modules
- ➡ But you also need a live repository for infrastructure deployed in each environment

Repositories

- Modules must be maintained and versioned independently of the files for live environments which allow different environment to **simultaneously** reference different versions of the modules
- Separate terraform workspaces and repositories must be considered for live infrastructure deployments

Pipeline

Terraform project

