UNIVERSITÀ CATTOLICA del Sacro Cuore

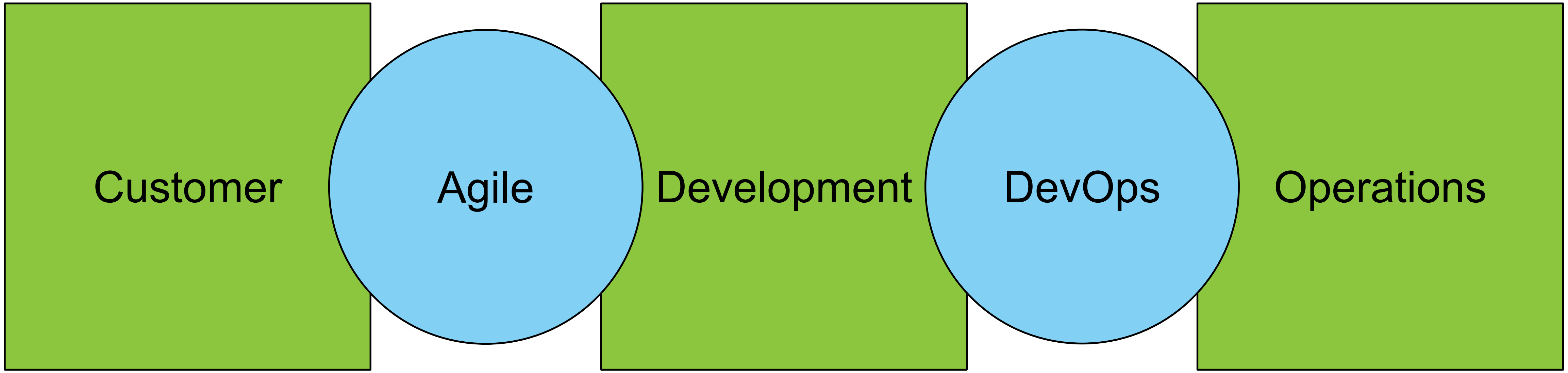
DevOps and GIT

INTRO TO DEVOPS

DevOps (*n.*) –

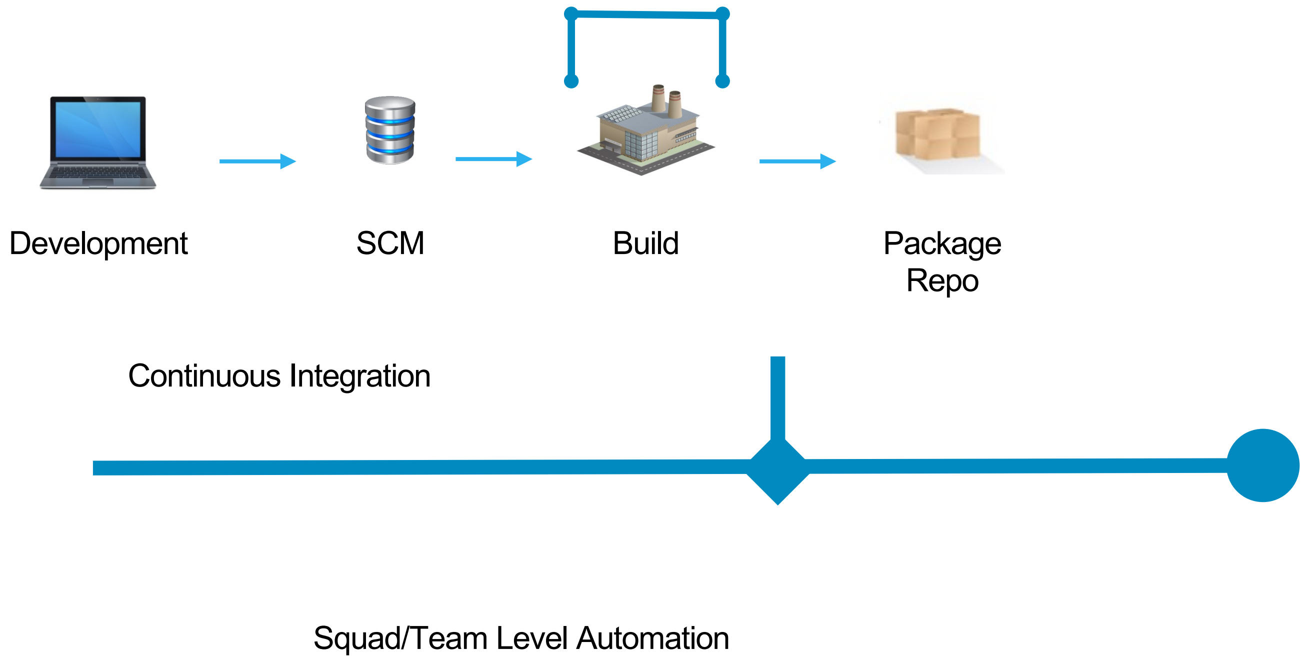
“DevOps is a philosophy, a **cultural shift** that **merges operations with development** and demands a **linked toolchain** of technologies to facilitate collaborative change. DevOps toolchains … can include dozens of non-collaborative tools, making the task of automation a technically complex and arduous one.”

Gartner



- **CI / CD**
  - Acronym for **Continuous Integration** plus **Continuous Deployment**
- **Continuous integration**
  - Creating packaged builds of the latest code changes as soon as they’re available
- **Continuous deployment**
  - Progressing each new packaged build through the deployment lifecycle stages as rapidly as possible, resulting in that package getting into production
- **Continuous delivery**
  - Continuous integration + continuous deployment
- **Delivery pipeline**
  - An automated sequence of steps to perform CI / CD

CONTINUOUS INTEGRATION



Frequently performing all of these steps in sequence:

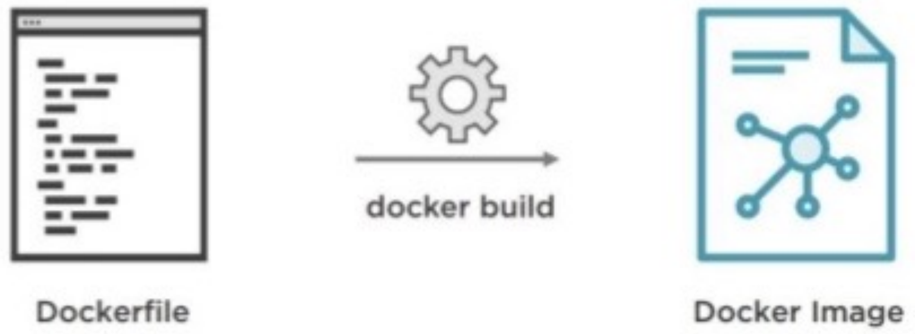
- Development
  - Rapidly implementing changes in small, tested batches
- SCM (Source Code Management)
  - Merging changes from multiple developers
  - Tools like GitHub, SVN, etc
- Build
  - Creating new deployment artifacts
  - Tools like Jenkins, Gradle, Maven, etc
- Package
  - Installing builds into runtimes
  - Releasing runtimes as immutable images
  - Cloud Foundry push, Building container images

In the package step, an immutable image is created

- An immutable image does not get changed – only used for deploying instances
- If it needs to change, you delete it and create a new one

Examples

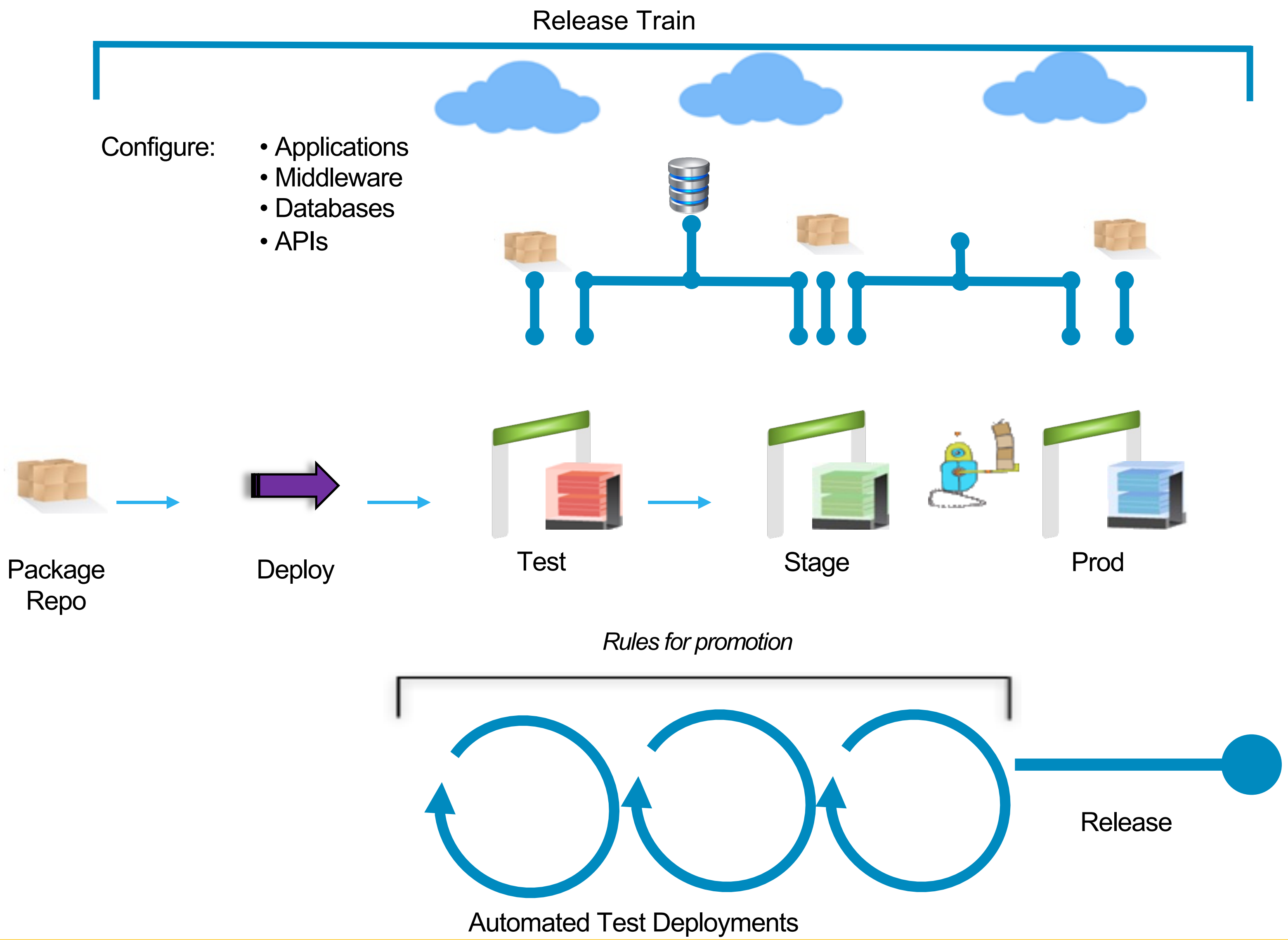
- Docker containers
  - A container image is created from a Dockerfile
  - After that, it is only deployed as a container, not changed
  - If you need to make changes, make a new container image by creating a new build and running the Dockerfile again



CONTINUOUS DEPLOYMENT

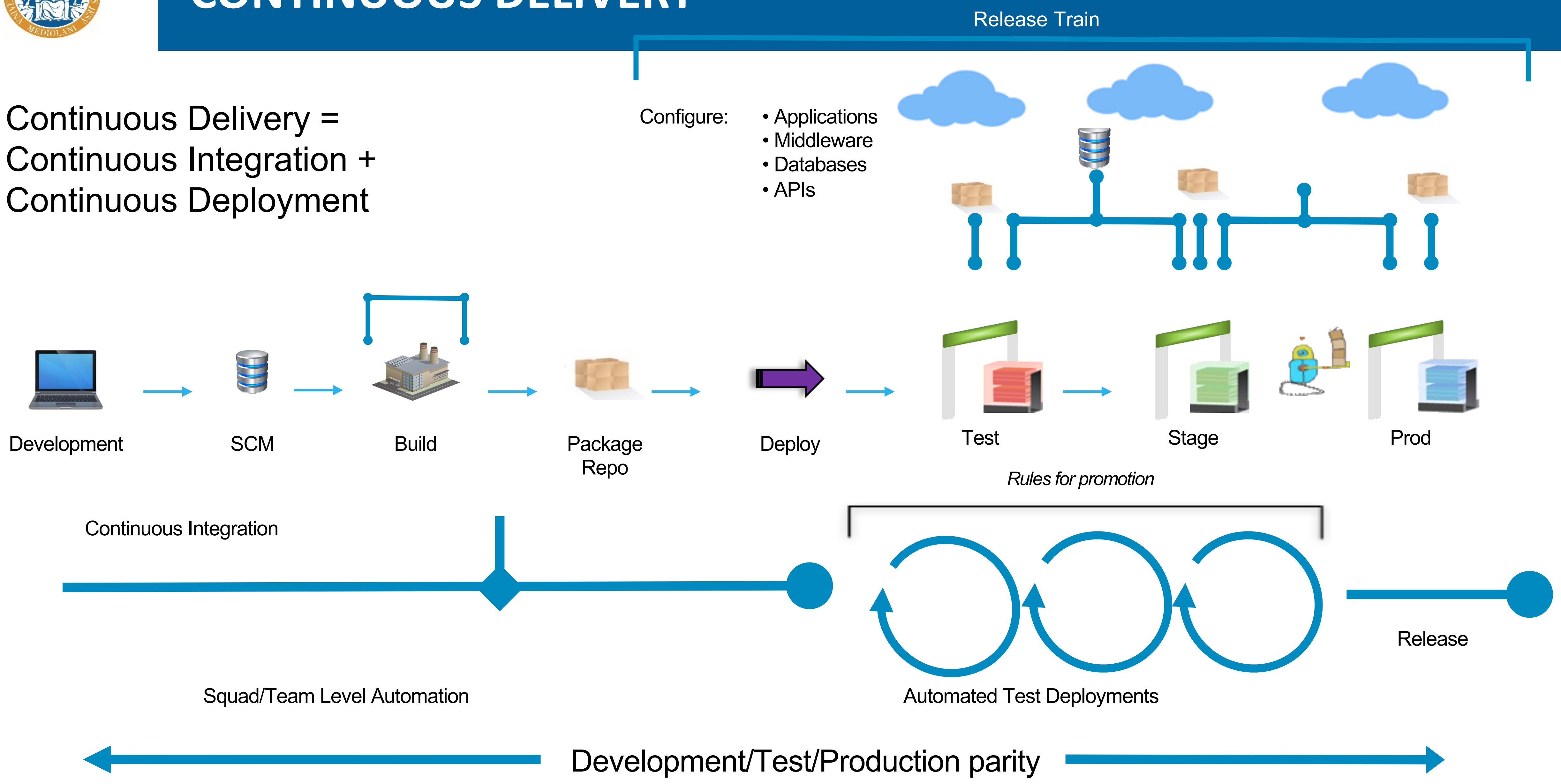
Rapidly progressing the latest packaged build through the test lifecycle stages and into production

- Deploy to Test
  - Perform functional testing
  - Automated use of test tools
- Deploy to Stage
  - Rehearse production deployment
  - Perform integration testing
- Deploy to Prod
  - Make the build available to users



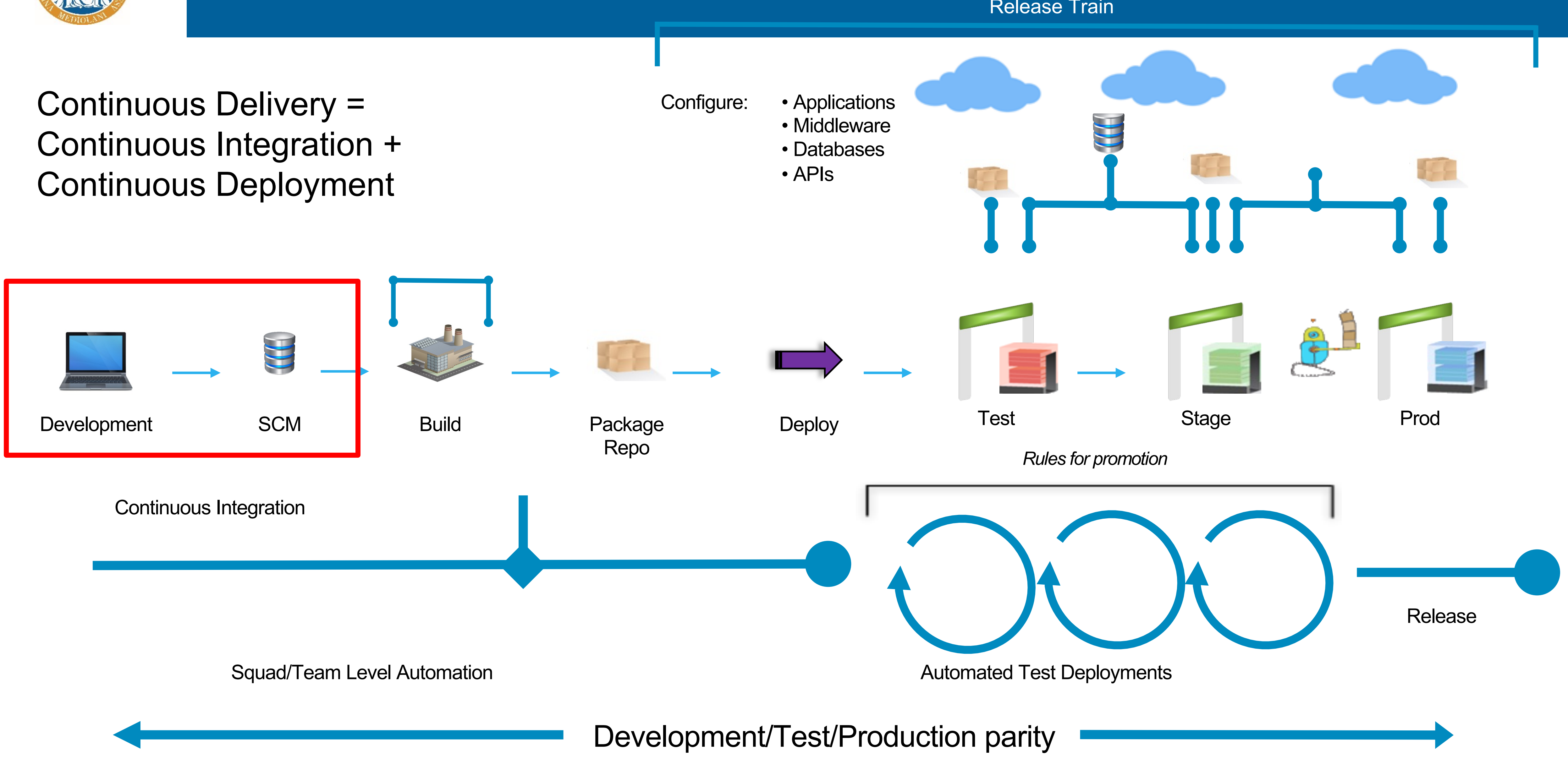
CONTINUOUS DELIVERY

Continuous Delivery =
Continuous Integration +
Continuous Deployment



CONTINUOUS DELIVERY

Continuous Delivery =
Continuous Integration +
Continuous Deployment



GIT

Git is an open-source, distributed version control system – a tool to manage files and changes to files

Git went G.A. in 2005. It is maintained by the Linux foundation

- Store your & organize source files
- Take snapshots of files – capturing changes to your source code as versions over time
- Restore earlier versions of files from snapshots
- Work on multiple versions of a file in parallel
- Work on different parts of a file in parallel
- Note that Git can be used for all filetypes: XML, PowerPoint, MS-Word Spreadsheets, graphics files, etc.

WHY GIT?

- Improved parallel development
- Faster SCM tooling performance
- Simplified merge and faster release cycle
- Seamless integration into an open CI/CD pipeline
- Strong support for “non-linear development”: you can work on different parts of the application concurrently
- Cloud integration
- Supported by all modern tools/IDEs
- One Version Control System for both enterprise & distributed applications

GIT LOCAL VS GITHUB OR MANAGED SERVICE

Git is the version control tool (SCM) that tracks changes to your files

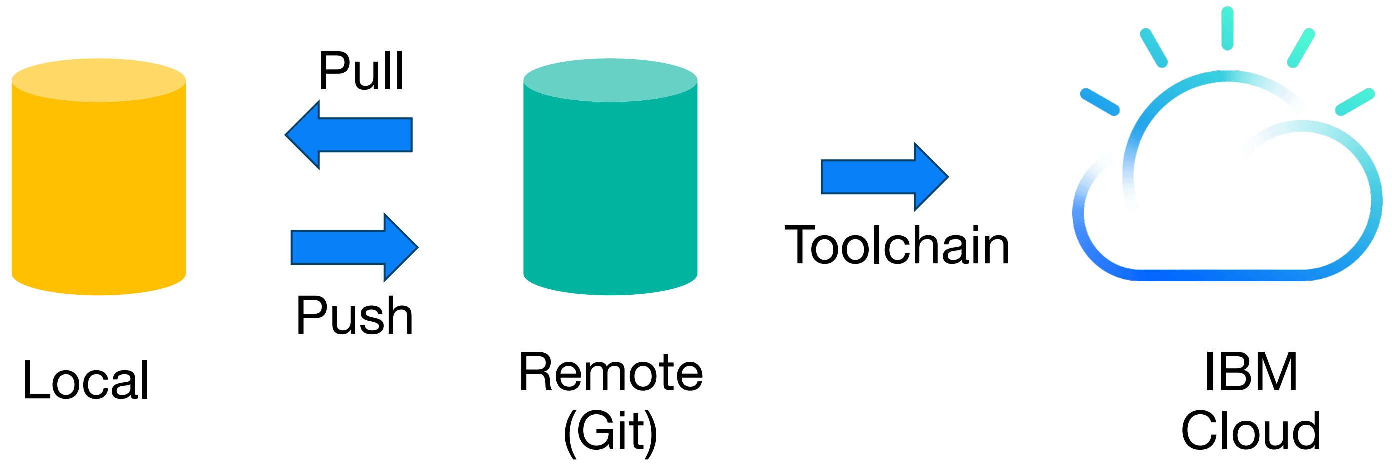
- Git Installs locally and manages version control and file sharing:
  - Via a desktop interface (GUI and/or Command-line)
  - Using a Repository to manage the history of file changes

GitHub, GitLab and **Bitbucket** are web-based **Git hosting products** – which provide:

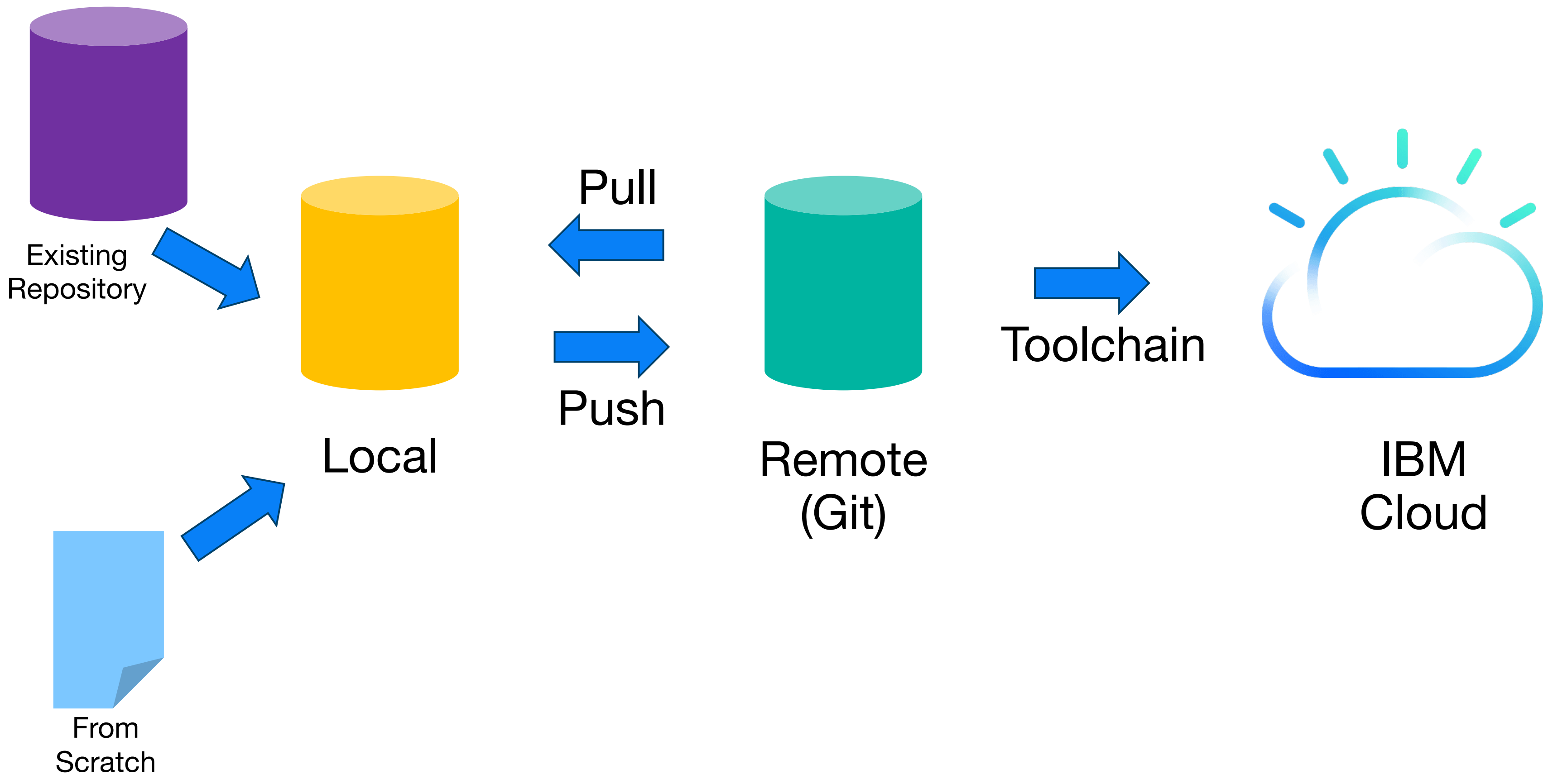
- A remote Git Version Control System
  - Free space for open-source projects, "for-pay" space for private projects
- Features for collaborative production-quality project work:
  - Work-item management and organizational tools
  - Problem-tracking, Branch protection, etc.

- **GitHub:** a web-based hosting service for version control using Git. GitHub is owned by Microsoft: https://github.com/
- **GitLab** is a cloud-based, DevOps lifecycle tool providing CI/CD pipeline features. GitLab is owned by GitLab, Inc. https://about.gitlab.com/
- **Bitbucket** is a web-based source-control-management Repository owned by Atlassian. Bitbucket offers commercial & free Git Repository time? https://bitbucket.org

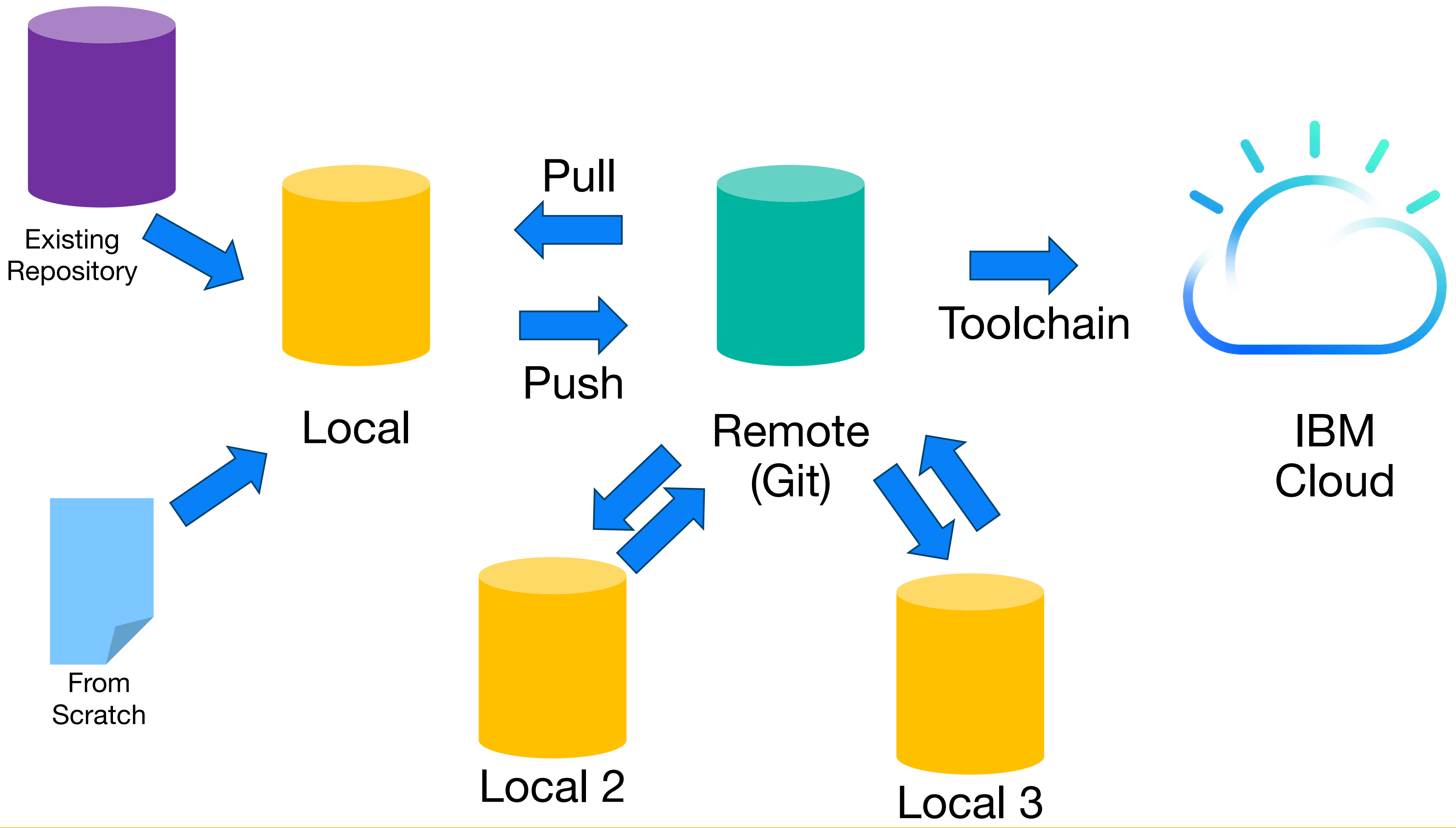
GIT & CLOUD

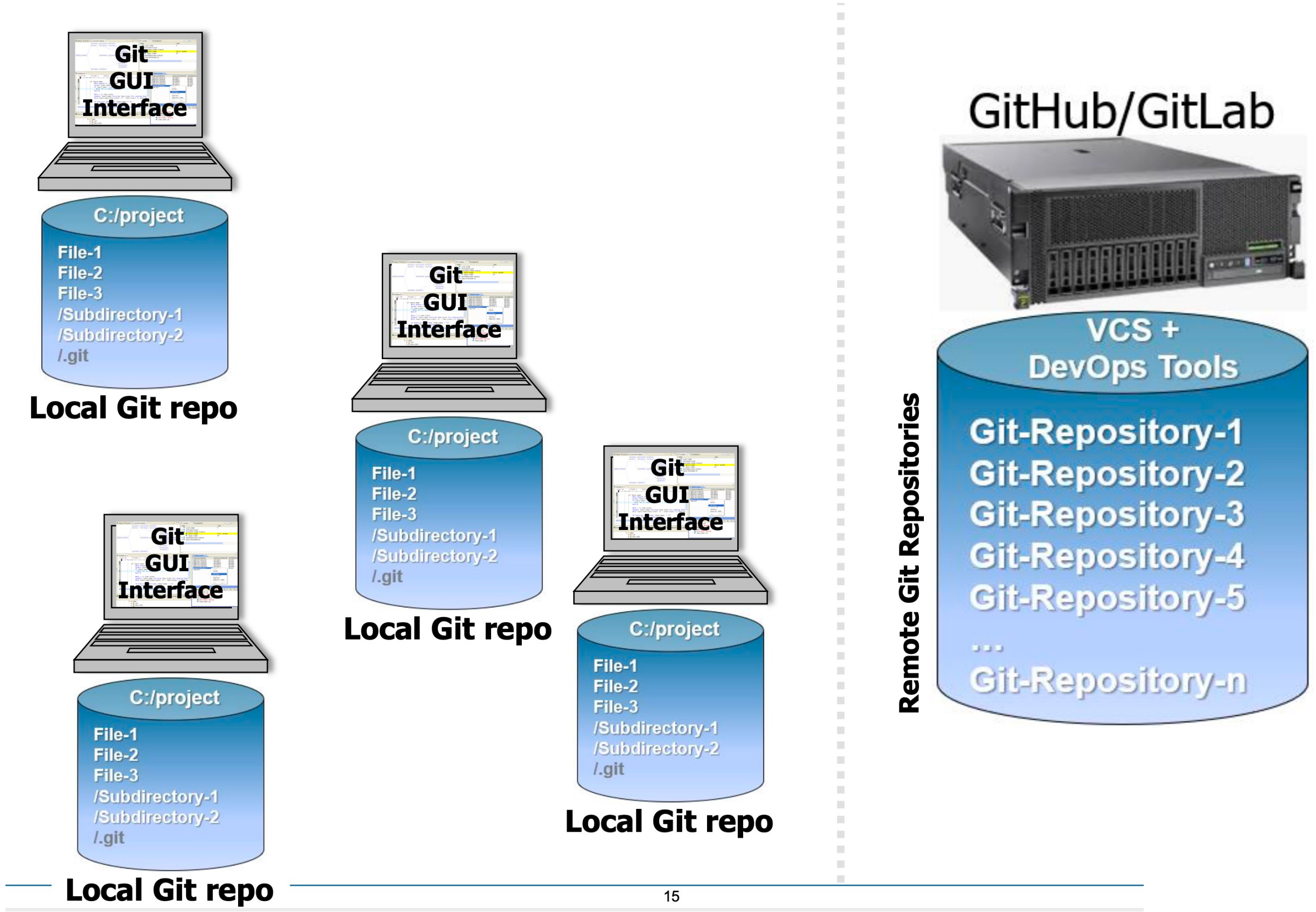


GIT & CLOUD

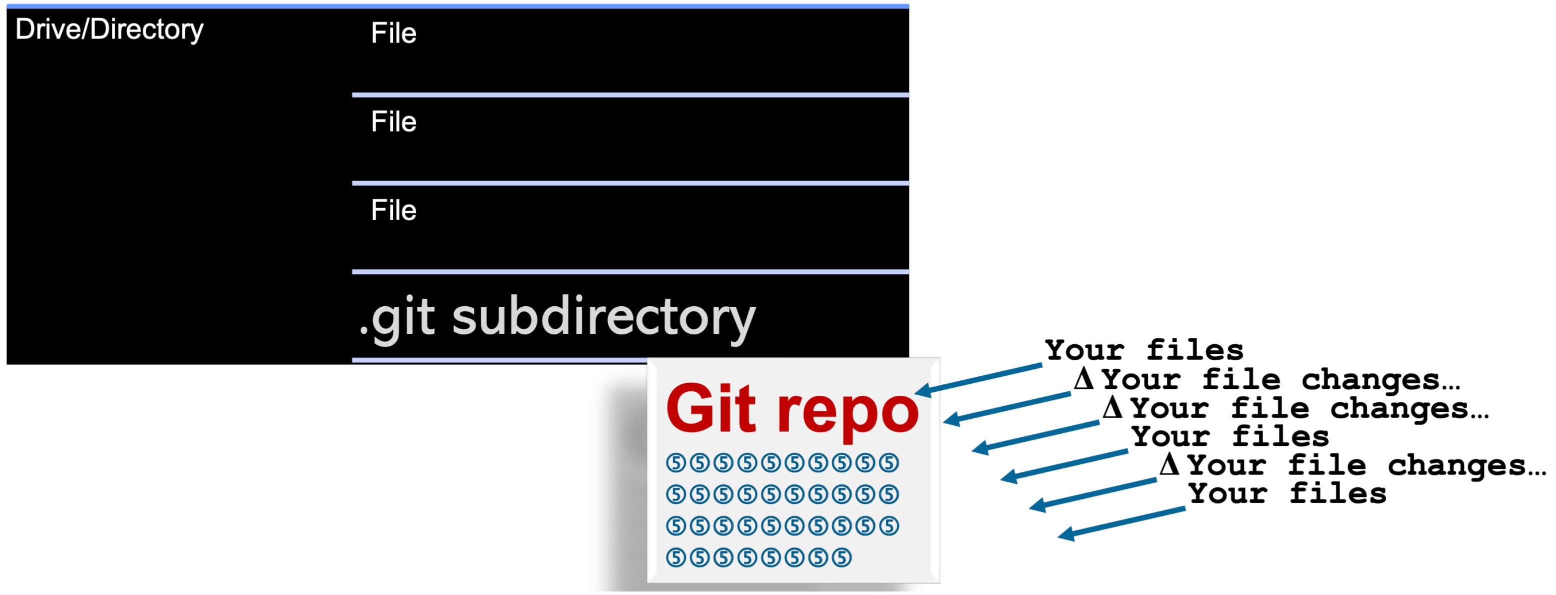

Existing Repository
From Scratch
Local
Pull
Push
Remote (Git)
Toolchain
IBM Cloud

GIT & CLOUD

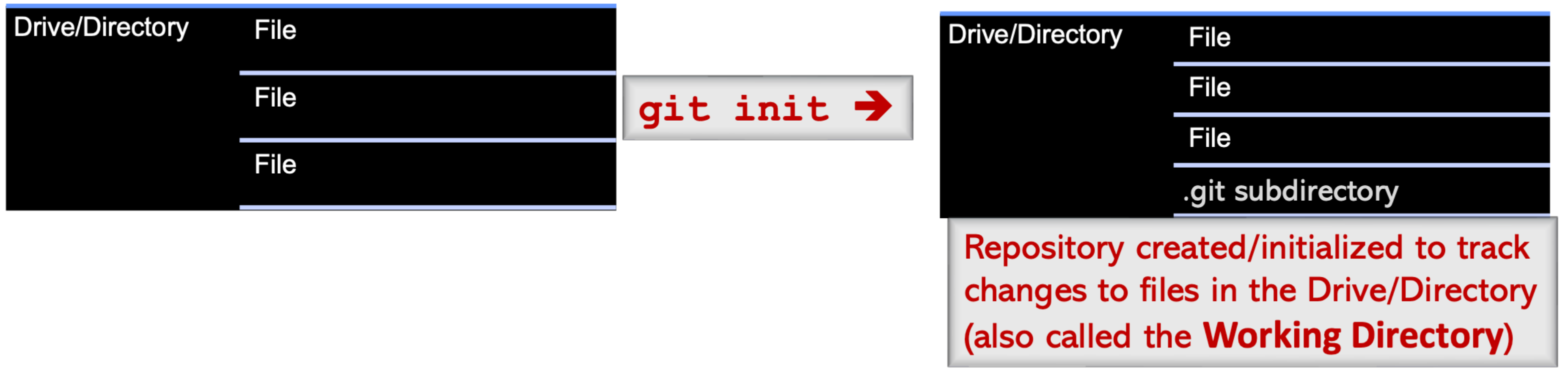

Existing
Repository
From
Scratch
Local
Pull
Push
Remote
(Git)
Toolchain
IBM
Cloud
Local 2
Local 3


Git GUI Interface
C:/project
File-1
File-2
File-3
/Subdirectory-1
/Subdirectory-2
/.git
Local Git repo
Git GUI Interface
C:/project
File-1
File-2
File-3
/Subdirectory-1
/Subdirectory-2
/.git
Local Git repo
Git GUI Interface
C:/project
File-1
File-2
File-3
/Subdirectory-1
/Subdirectory-2
/.git
Local Git repo
Git GUI Interface
C:/project
File-1
File-2
File-3
/Subdirectory-1
/Subdirectory-2
/.git
Local Git repo
Remote Git Repositories
GitHub/GitLab
VCS + DevOps Tools
Git-Repository-1
Git-Repository-2
Git-Repository-3
Git-Repository-4
Git-Repository-5
…
Git-Repository-n

- A Git **Repository** (“repo”) is a hidden **folder** named **.git** under the project drive:/directory that tracks changes to files
  - This **.git** subdirectory contains the metadata Git uses to track file changes - building a “history of changes” over time
  - If you delete the **.git** folder, then you delete your Repository



- You turn a directory into a Git Repository using the command: **git init**
  - After initializing a Repository, Git creates that **.git** subdirectory



Repository created/initialized to track changes to files in the Drive/Directory (also called the **Working Directory**)

Notes:

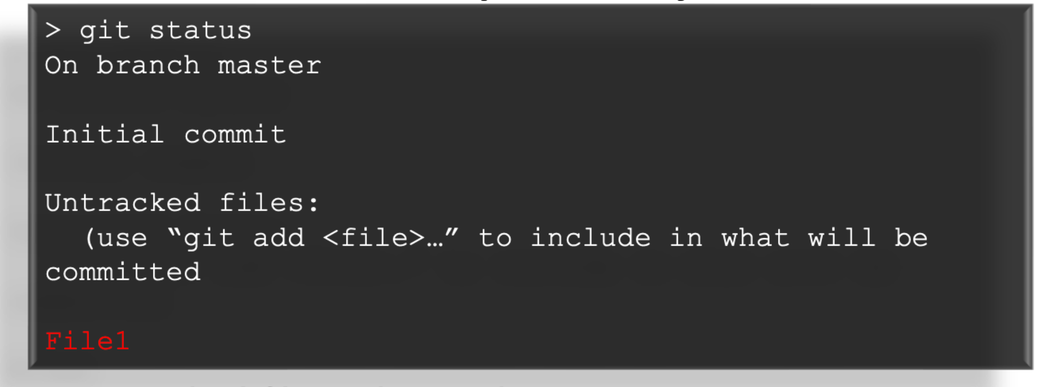
- You can create a local Repository anywhere – including your local workstation
- However, an Enterprise Git implementation is usually done from a Server
- You can name your Repository using: `git init <reponame>`

The initialized directory is referred to as a **Working Directory –** some people refer to it as the **“Working Tree”**

1. When working on project tasks you make changes to files in the **Working Directory**
2. All files in the **Working Directory** are "untracked" by Git until you add them to the Git Repository using `git add`

To obtain a status of the files in your Working Directory versus files committed to the Repository use the command:

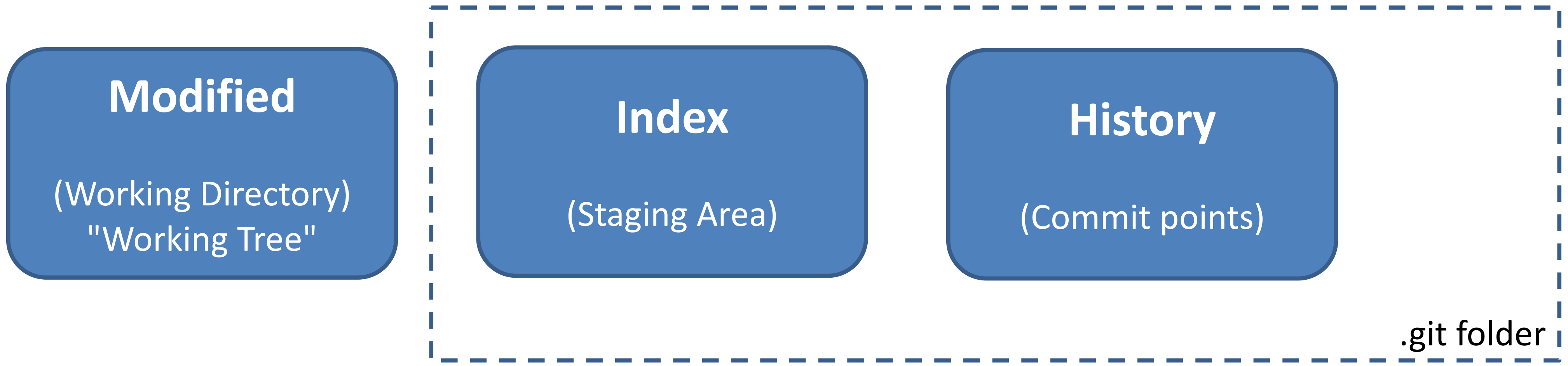
`git status`



```
> git status
On branch master

Initial commit

Untracked files:
   (use “git add <file>…” to include in what will be
committed

File1
```

An untracked file in the Working-Directory

- The output of `git status` can be referred to as the *“Working Tree status“*



Your files and folders exist in one three states:

1. Modified – Files that are being edited reside in the **Working Directory**. This is where you add/delete/copy/edit. Files in the Working Directory are not known to Git

2. Index – Files that are ready to be committed or saved are “Staged” and added to the Git Index

3. History – Files that have been saved/committed create Commit Points in the Repository History

Assuming that your Working Directory contains source files you might:

1. Initialize Git – Create a Repository

2. Configure user-name/email

– Allow Git to include your name, email as part of a Commit Point

3. Modify files

– Make changes to files in the Working Directory.

4. Stage files

– Prepare to Commit your work by Staging files

5. Commit the Staged files

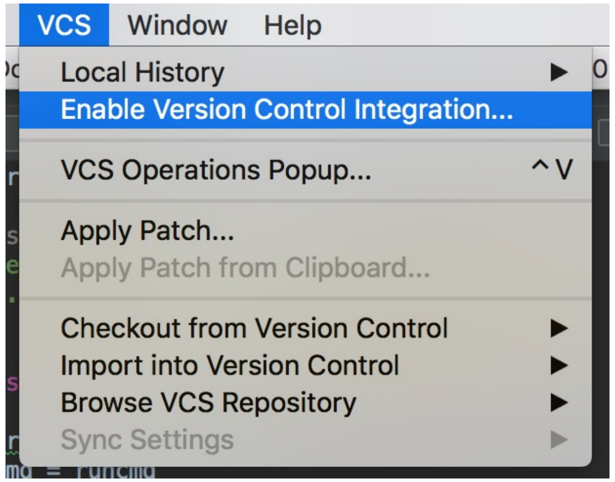
– Save the staged files – store a snapshot of the files in your Git Repository

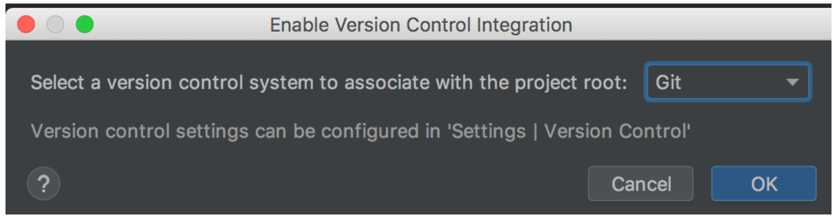
6. Create a .gitignore file

- Tell git which files/folders in the Working Directory to ignore

Note: Typically in Enterprise projects you will check code out of a Git Repository and work on the code in a “branch” created for your project team

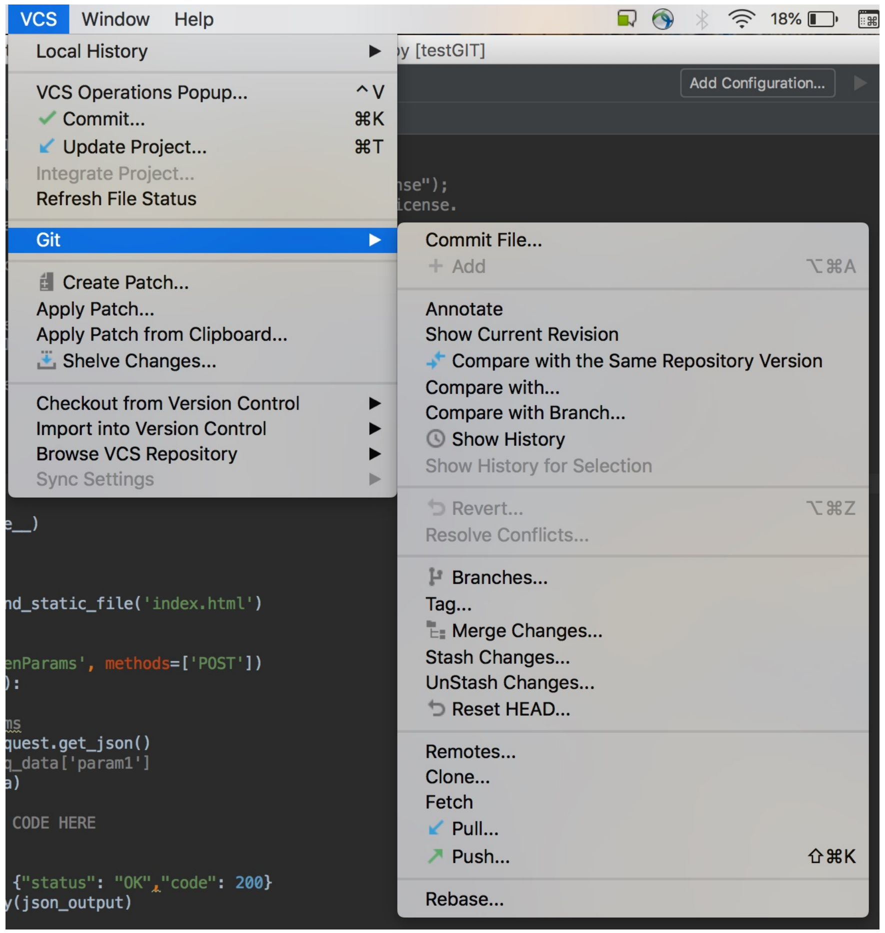
VERSIONING IN PYCHARM 1/3


VCS
Window
Help
Local History
Enable Version Control Integration...
VCS Operations Popup... ^V
Apply Patch...
Apply Patch from Clipboard...
Checkout from Version Control
Import into Version Control
Browse VCS Repository
Sync Settings



Select a version control system to associate with the project root: Git

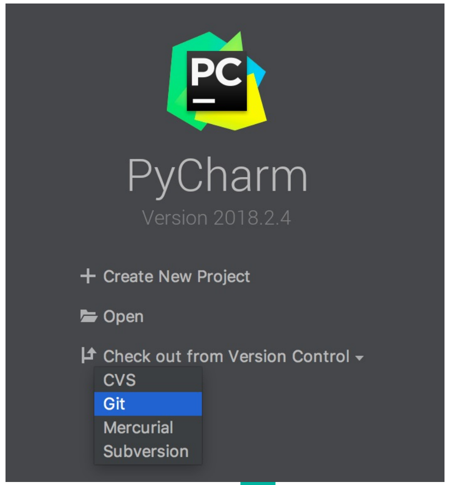
Version control settings can be configured in 'Settings | Version Control'

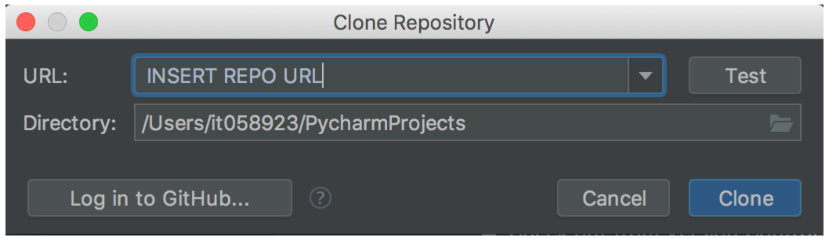


```
nse");
icense.
e__)
nd_static_file('index.html')
enParams', methods=['POST'])
):
ms
quest.get_json()
_data['param1']
a)
 CODE HERE
 {"status": "OK","code": 200}
y(json_output)
```

VERSIONING IN PYCHARM 2/3

CLONE REPO

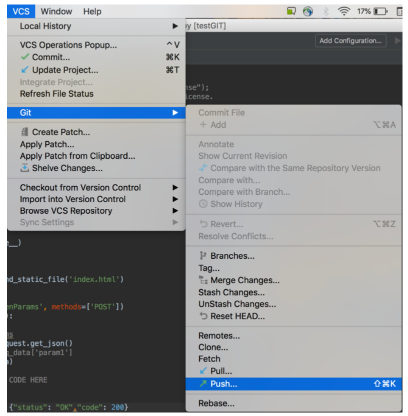


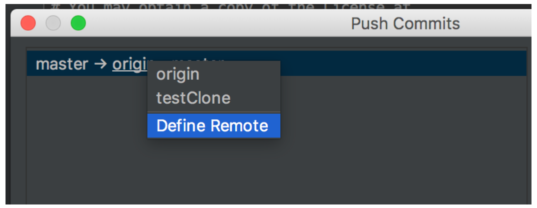

Clone Repository

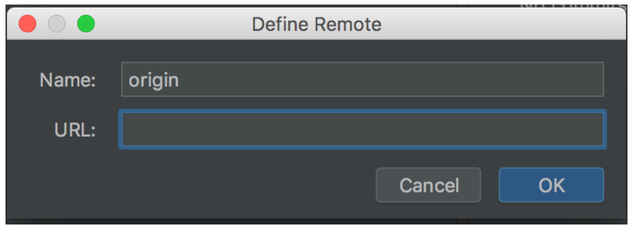
URL: INSERT REPO URL

Directory: /Users/it058923/PycharmProjects

SAVE TO OWN REPO





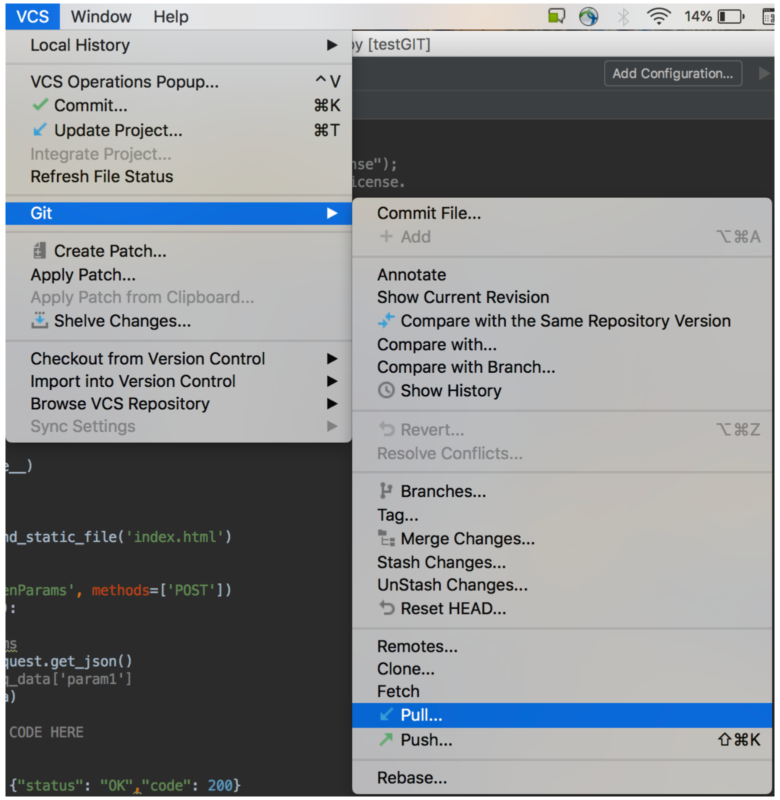

Define Remote

Name: origin

URL:

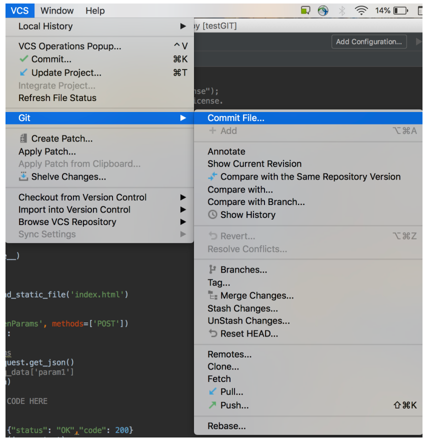
VERSIONING IN PYCHARM 3/3

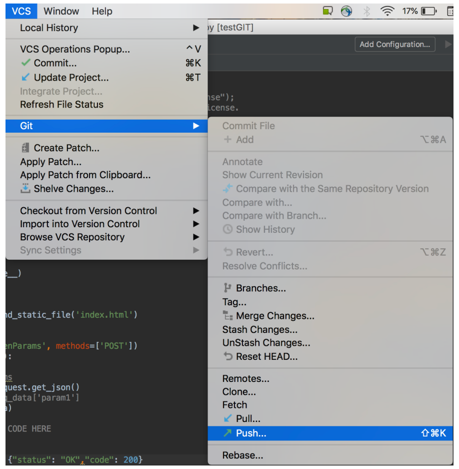
PULL (update local)



```
y [testGIT]
nse");
icense.
e__)
d_static_file('index.html')
enParams', methods=['POST'])
:
ns
quest.get_json()
_data['param1']
a)
CODE HERE
{"status": "OK","code": 200}
```

PUSH (update remote)





Commit changes -> Push Changes

RESOURCES

Git Handbook

https://guides.github.com/introduction/git-handbook/

Git Documentation

https://git-scm.com/doc

Q&A

Thanks

for your attention!

Questions?

Federico Accetta federico_accetta@it.ibm.com