

CSIT314 Software Development Methodologies



Principles and practices of continuous integration and delivery (CI/CD)

Acknowledgement: Materials from these slides are adapted from the following sources: "Continuous Integration" by Martin Fowler and GitLab documentation.

What is integration?

- ❑ Software teams often have multiple developers working on the **same codebase** at the **same time** (**independently**):
- ❑ E.g. Developer A works on feature 1 while developer B works on feature 2.
- ❑ E.g. Developer A works on class 123.java while developer B works on class 456.java
- ❑ Once they have finished, they need to **integrate** their work into the main codebase.
 - Question: What issues could arise at integration?

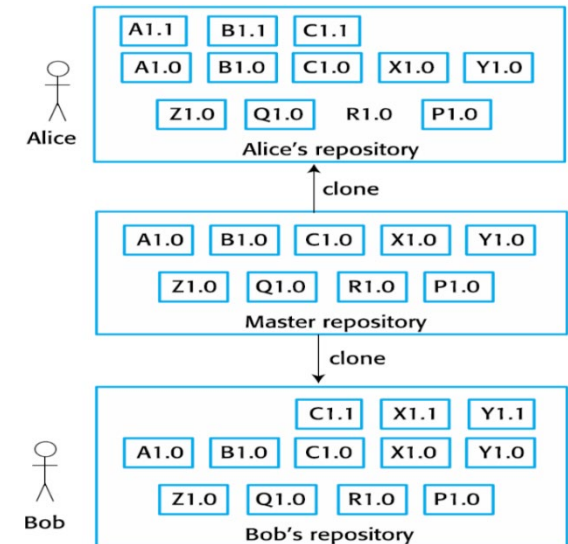
What is continuous integration (CD)?

- ❑ Continuous integration (CD) is a software development practice where developers in a team **integrate** their work **frequently**.
- ❑ Developers usually integrates several times a day.
- ❑ Each integration is verified by an **automated build**: compile the code and also run automated tests?
 - Question: Why are automated tests run?

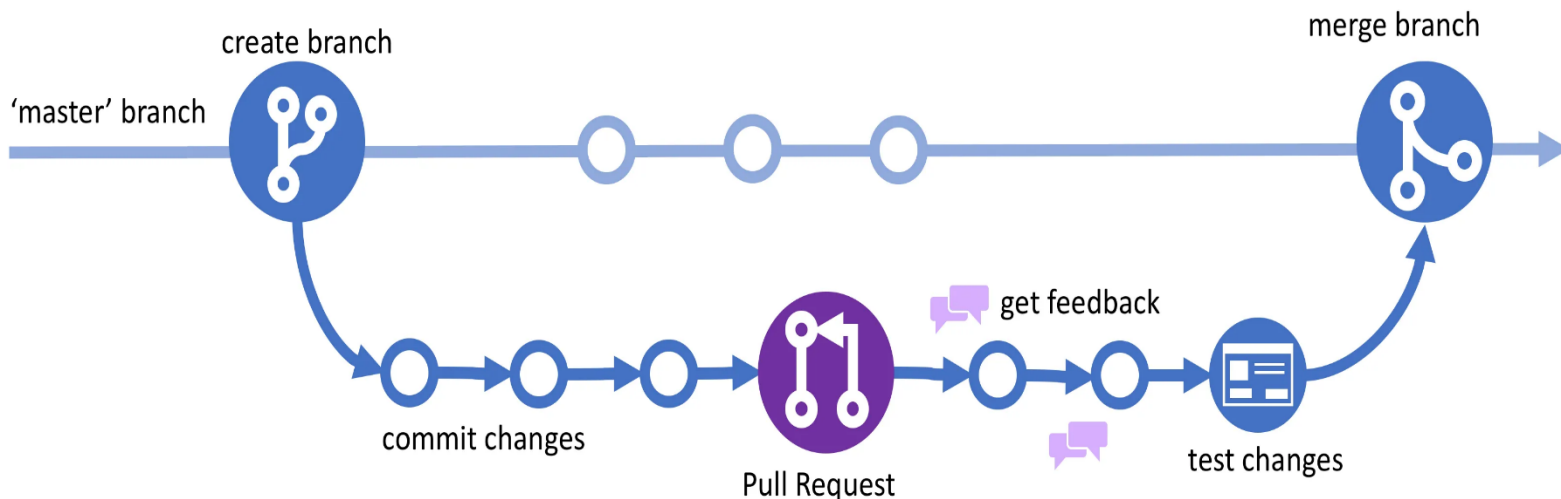
Practices of Continuous Integration

■ Maintain a Single Source Repository

- Use a version control system such as Git/GitHub
- Everything required to build the software app should be in the repository (code, test scripts, test data, properties files, database schema, third-party libraries, etc.)



GitHub Flow



Practices of Continuous Integration

- ❑ Automate the build
 - Automate the whole process of turning the source code into a running software app
 - ❑ Often include compiling, moving files around, loading schemas into the databases, etc.
 - Issuing a single command and the whole build process will run automatically.

Practices of Continuous Integration

- ❑ Make your build self-testing
 - Include automated tests as part of the build process.
 - All automated **unit tests** (refer to test-driven development) should be run as part of the continuous integration practice (e.g. JUnit, NUnit, etc.).
 - All automated **acceptance tests** should be run as part of the continuous integration practice (e.g. Selenium).

Practices of Continuous Integration

- Everyone **commits** (*integrate their changes*) to the main codebase every day (or even better, multiple times a day)
 - Integration is a way in which developers can “inform” other developers about the changes they have made: frequent integration means frequent communication!
 - By integrating their changes frequently, developers **quickly find out** if there is a **conflict** between the changes.
 - Errors, conflicts, bugs, etc. can be detected early and rectified quickly.

Practices of Continuous Integration

- Every commit should build the mainline (the main codebase) on an integration machine
 - Regular builds happen on an integration machine and only if this integration build succeeds should the commit be considered to be done.
 - Use continuous integration server (e.g. Jenkins, Travis CI, Bamboo, GitLab, etc.)

Practices of Continuous Integration

- ❑ Fix Broken Builds Immediately
 - A key part of doing a continuous build is that if the mainline build fails, it needs to be fixed right away.
 - “nobody has a higher priority task than fixing the build”

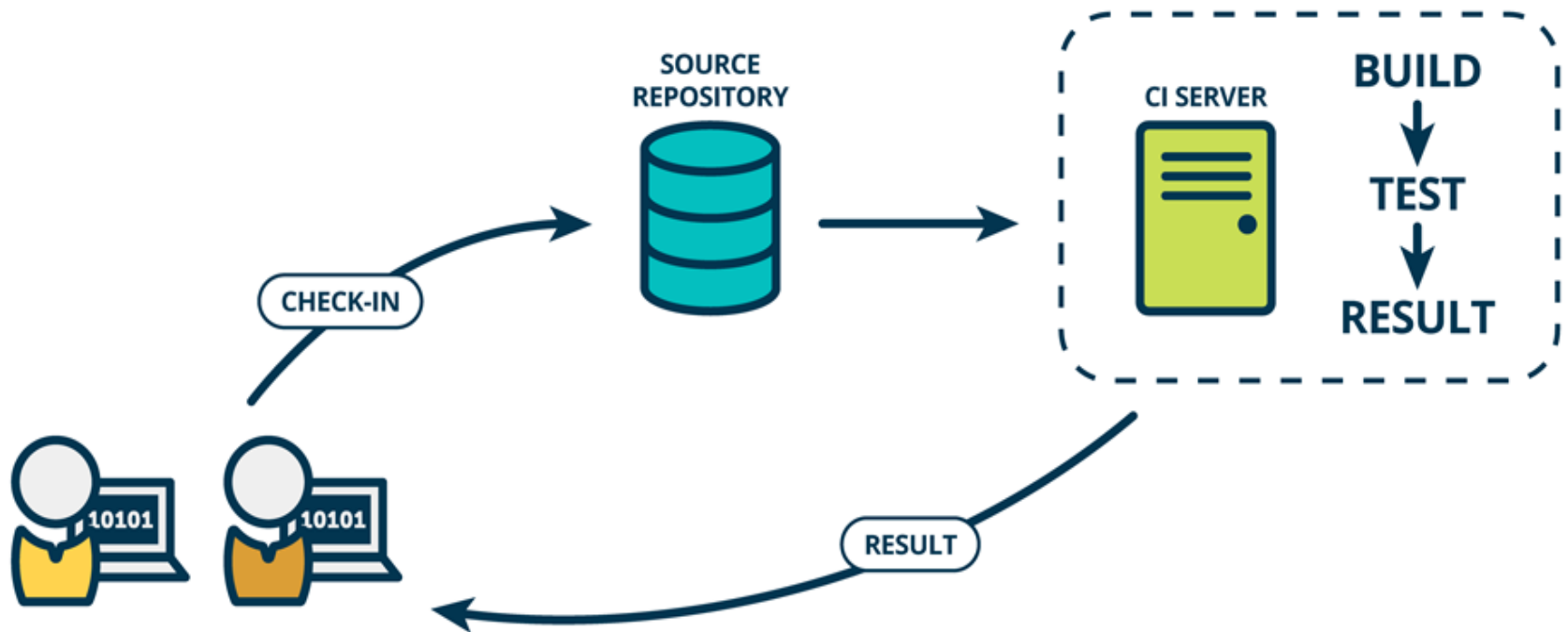
Practices of Continuous Integration

- Keep the build fast.
 - The whole point of Continuous Integration is to provide rapid feedback.
 - If the build is not fast, developers will commit less often and will be provided feedback on problems less often.

Practices of Continuous Integration

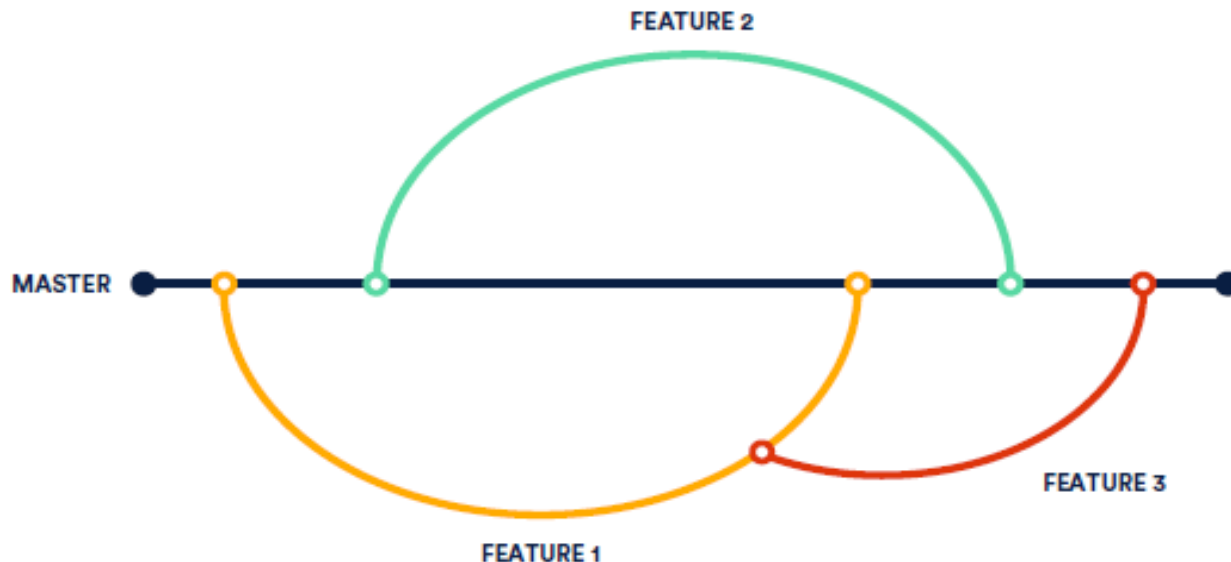
- ❑ Test in a clone of the production environment.
 - Testing in a different environment introduces risk when the system is deployed in production.

Continuous Integration cycle



Continuous Integration tools

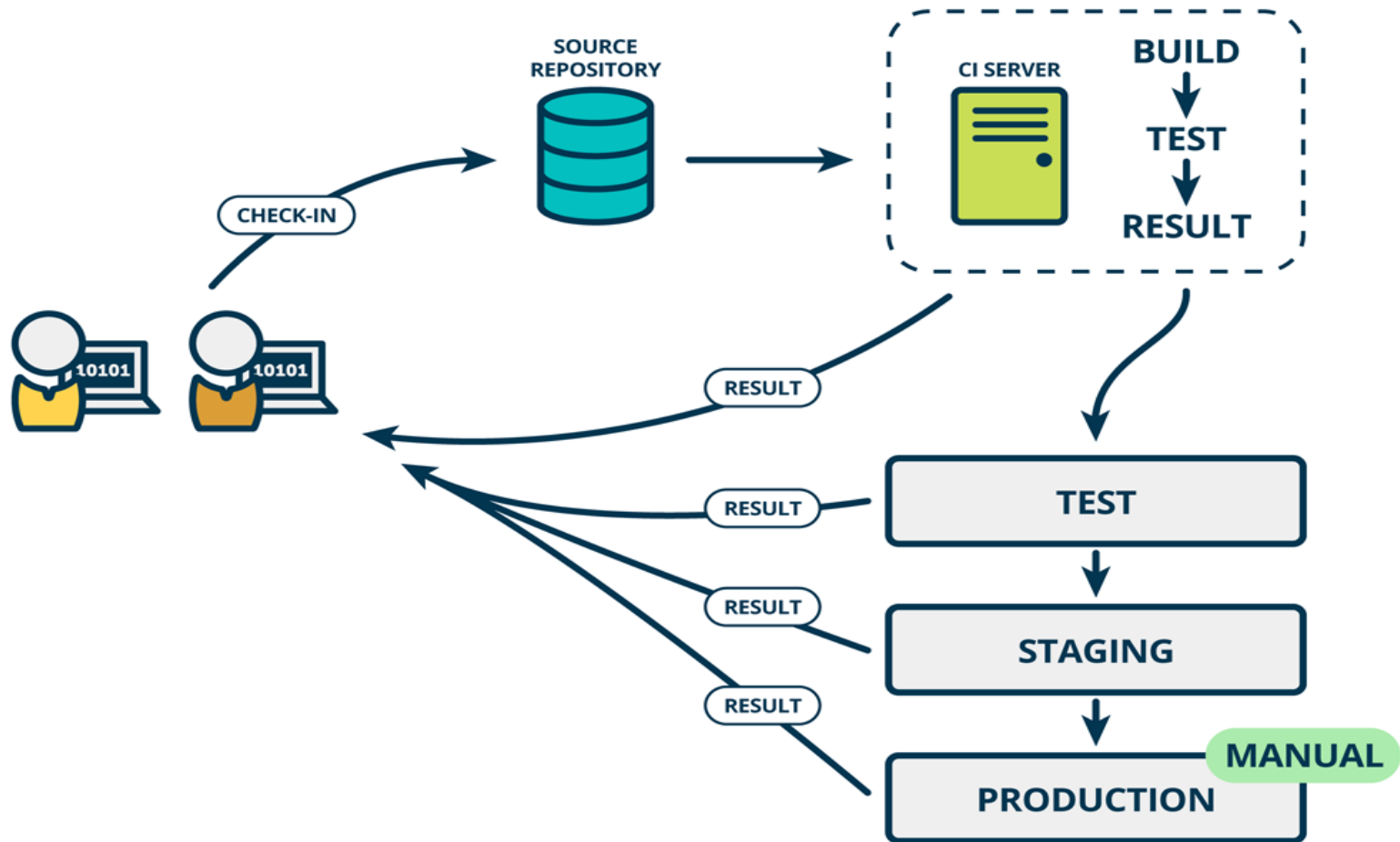
- ❑ CI tools: Jenkins, Travis CI, Bamboo, GitLab CI, etc.
- ❑ Demo CI using Bamboo:
https://youtu.be/GIfYP4HmZ_4



Continuous Delivery

- ❑ Continuous Delivery is a further step beyond Continuous Integration:
 - Each time changes are pushed to the codebase, the new code is automatically built and tested on environments that are **very similar** to production (**staging environment**)
- ❑ The staging environment addresses **non-functional** requirements such as security, load-balancing, redundancy, and scalability. These may not be covered in the development environments
- ❑ Question: why is it useful to plan for a staging environment?

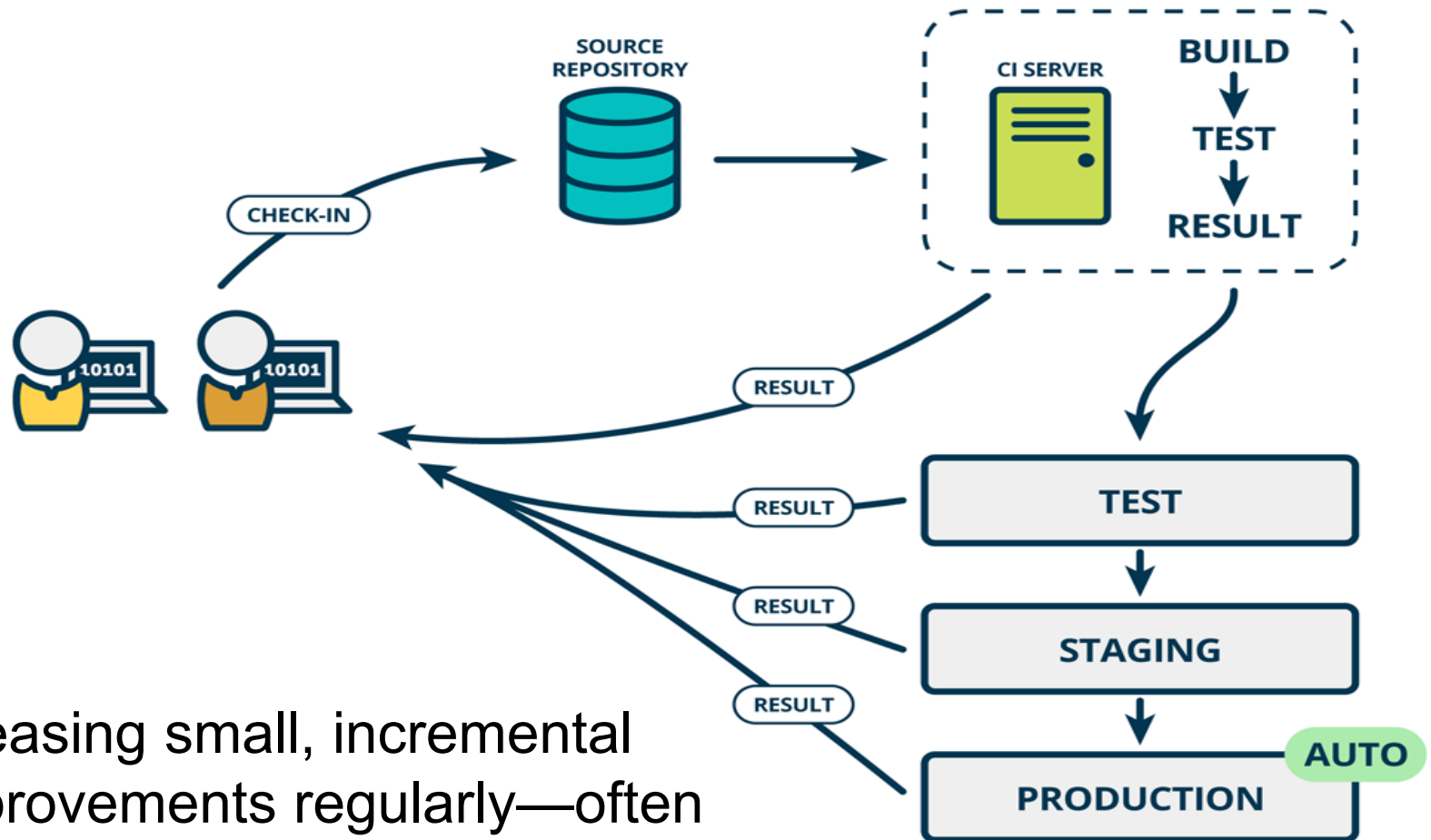
Continuous Delivery cycle



Continuous Deployment

- ❑ In Continuous Delivery, the deployment of the software application to a production environment is still done manually.
- ❑ Continuous Deployment is an additional step beyond Continuous Delivery:
 - The software application is deployed automatically.
 - Every time code changes are pushed to the codebase, it will be automatically built and tested – and if the tests are successfully, it will automatically go to production.

Continuous Deployment cycle



releasing small, incremental improvements regularly—often even several times per day.

Continuous Deployment (CD)

- ❑ Continuous Deployment demo:
 - Bamboo demo: <https://youtu.be/rG-XxVYNS4c>
 - GitHub and Azure demo:
<https://youtu.be/3WDe3l1M-3U>

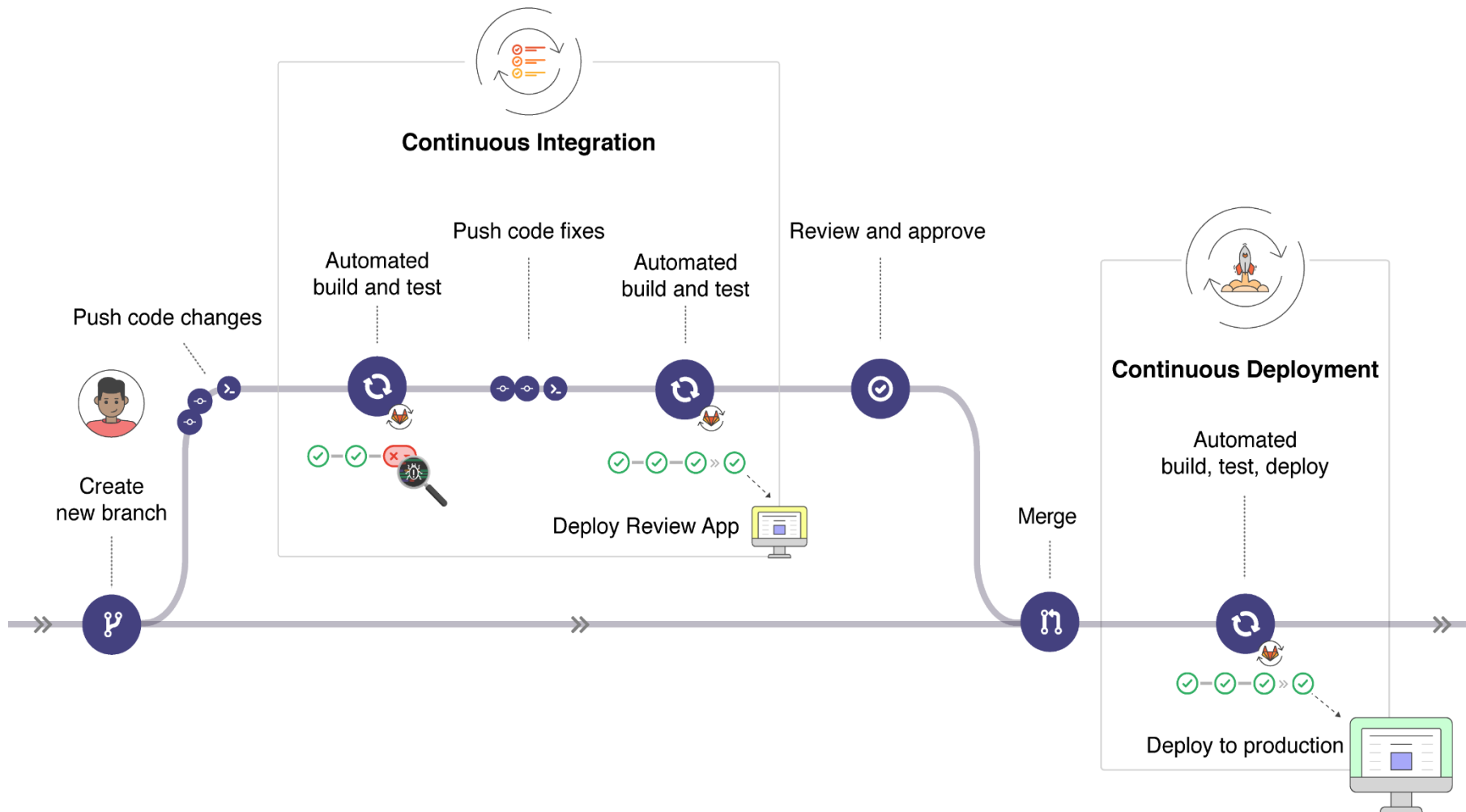
Infrastructure as Code

- ❑ What is IT infrastructure?
 - Physical machines, devices, OS, databases, and any other systems that are used to run a software application.
- ❑ Infrastructure as Code: the whole IT infrastructure can be treated as if they are software.
 - The whole IT environment can be setup, configured and changed automatically through writing code.
- ❑ Infrastructure as code is the prerequisite for common CI/CD practices

Infrastructure as Code (cont.)

- ❑ IaC model generates the same environment every time it is applied.
 - Infrastructure as Code enables software teams to test applications in production-like environments early in the development cycle
- ❑ IaC demo: https://youtu.be/k6_ZTIxI4xk

CI/CD with GitLab



CI/CD with GitLab (cont.)

- ❑ Create an account on GitLab
- ❑ Setup a project on GitLab
- ❑ Install Git on your computer:
 - <https://desktop.github.com/> or
 - <https://git-scm.com/downloads>

See a demo at:

<https://documents.uow.edu.au/~hoa/teaching/SIM/CI-CD.mp4>

CI/CD with GitLab (cont.)

□ Create a new project on GitLab



CI/CD with GitLab (cont.)

- ❑ Install GitLab Runner on your computer
 - <https://docs.gitlab.com/runner/#install-gitlab-runner>
- ❑ Register GitLab Runner with GitLab
 - <https://docs.gitlab.com/runner/register/>
- ❑ Start GitLab Runner


Administrator: Command Prompt


```
C:\GitLabRunner>gitlab-runner-windows-386.exe register
Runtime platform arch=386 os=windows pid=24916 revision=e95f89a0 version=13.4.1
Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com/):
https://gitlab.com
Please enter the gitlab-ci token for this runner:
MaYHncQz81oBonyDs-t9
Please enter the gitlab-ci description for this runner:
[eis-20019613]: MyRunner
Please enter the gitlab-ci tags for this runner (comma separated):
ci
Registering runner... succeeded runner=MaYHncQz
Please enter the executor: virtualbox, docker-ssh+machine, custom, docker-windows, docker-ssh, shell, kubernetes, docker
, parallels, ssh, docker+machine:
shell
Runner registered successfully. Feel free to start it, but if it's running already the config should be automatically re
loaded!

C:\GitLabRunner>
```


CI/CD with GitLab (cont.)

Set up a specific Runner manually

1. [Install GitLab Runner](#)
2. Specify the following URL during the Runner setup:
`https://gitlab.com/` 
3. Use the following registration token during setup:

`MaYHncQz81oBonyDs-t9` 

Reset runners registration token

4. Start the Runner!

Runners activated for this project

 **yTZG-8Sb...**  

MyRunner

#2968867

ci

Pause

Remove Runner

Available shared Runners: 15

 **9538b0ab**

gitlab-shared-runners-manager-4.gitlab.com

#157329

gitlab-org

 **0277ea0f**

shared-runners-manager-5.gitlab.com

#380986

docker gce

 **d5ae8d25**

gitlab-shared-runners-manager-5.gitlab.com

#380989

gitlab-org

CI/CD with GitLab (cont.)

- ❑ Create a CI/CD pipeline on GitLab
 - Create/add a file called “.gitlab-ci.yml” in the root folder of your project.
 - This YAML file defines the structure and order of the CI/CD pipeline and decides:
 - ❑ What will be executed using GitLab Runner
 - ❑ What to do when some specific conditions are satisfied (e.g. when a process succeeds or fails).

(See <https://docs.gitlab.com/ee/ci/yaml/> for more details)

CI/CD with GitLab (cont.)

- ▣ Let's watch a live demo of how to set up and run CI/CD with GitLab.

CI/CD with GitLab (cont.)

- GitLab CI/CD Examples for different types of applications written in different programming languages
 - <https://docs.gitlab.com/ee/ci/examples/>