Welcome to Jenkins

Introduction to Jenkins

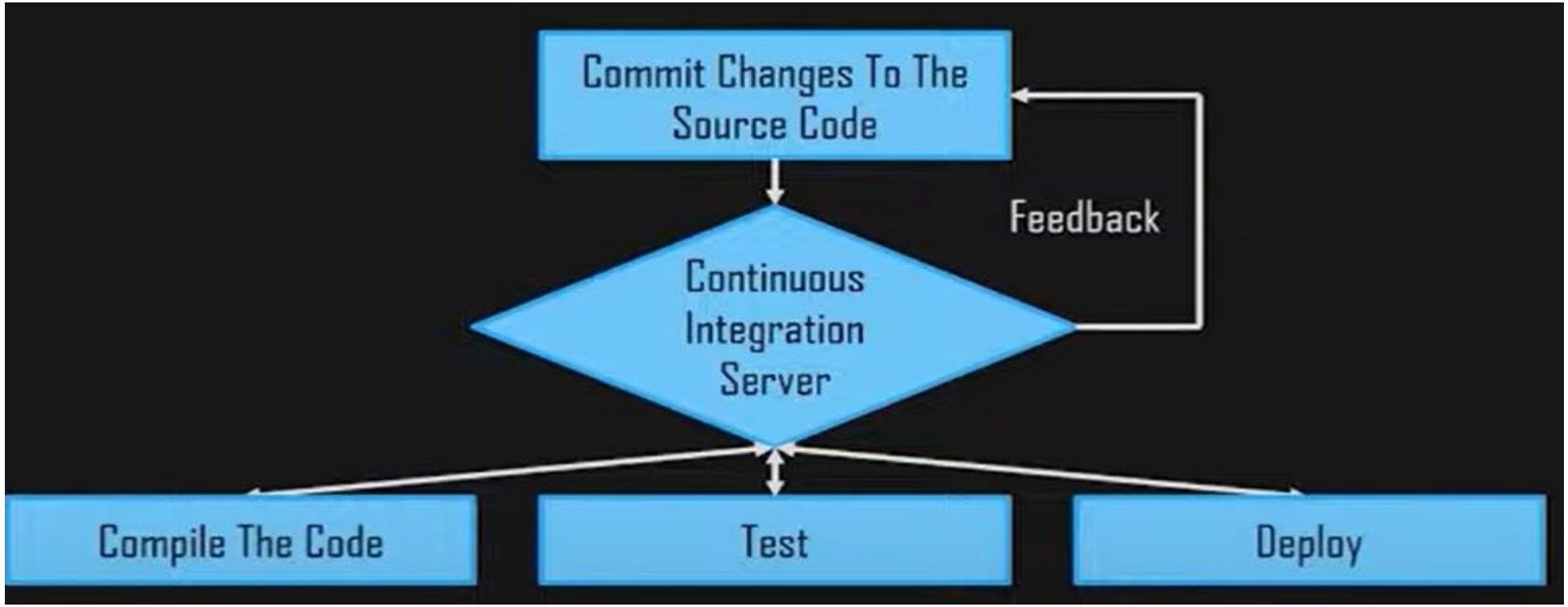
- Jenkins is a powerful application that allows continuous integration and continuous delivery of projects, regardless of the platform you are working on.
- It is a free source that can handle any kind of build or continuous integration.
- You can integrate Jenkins with a number of testing and deployment technologies
- Jenkins is a self-contained automation server which can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.
- Jenkins is written in Java with plugins built for Continuous Integration purpose.
- Jenkins is used to build and test your software projects continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build.
- It also allows you to continuously deliver your software by integrating with a large number of testing and deployment technologies.
- Continuous Integration is the most important part of DevOps that is used to integrate various DevOps stages.
- Jenkins is the most famous Continuous Integration tool.

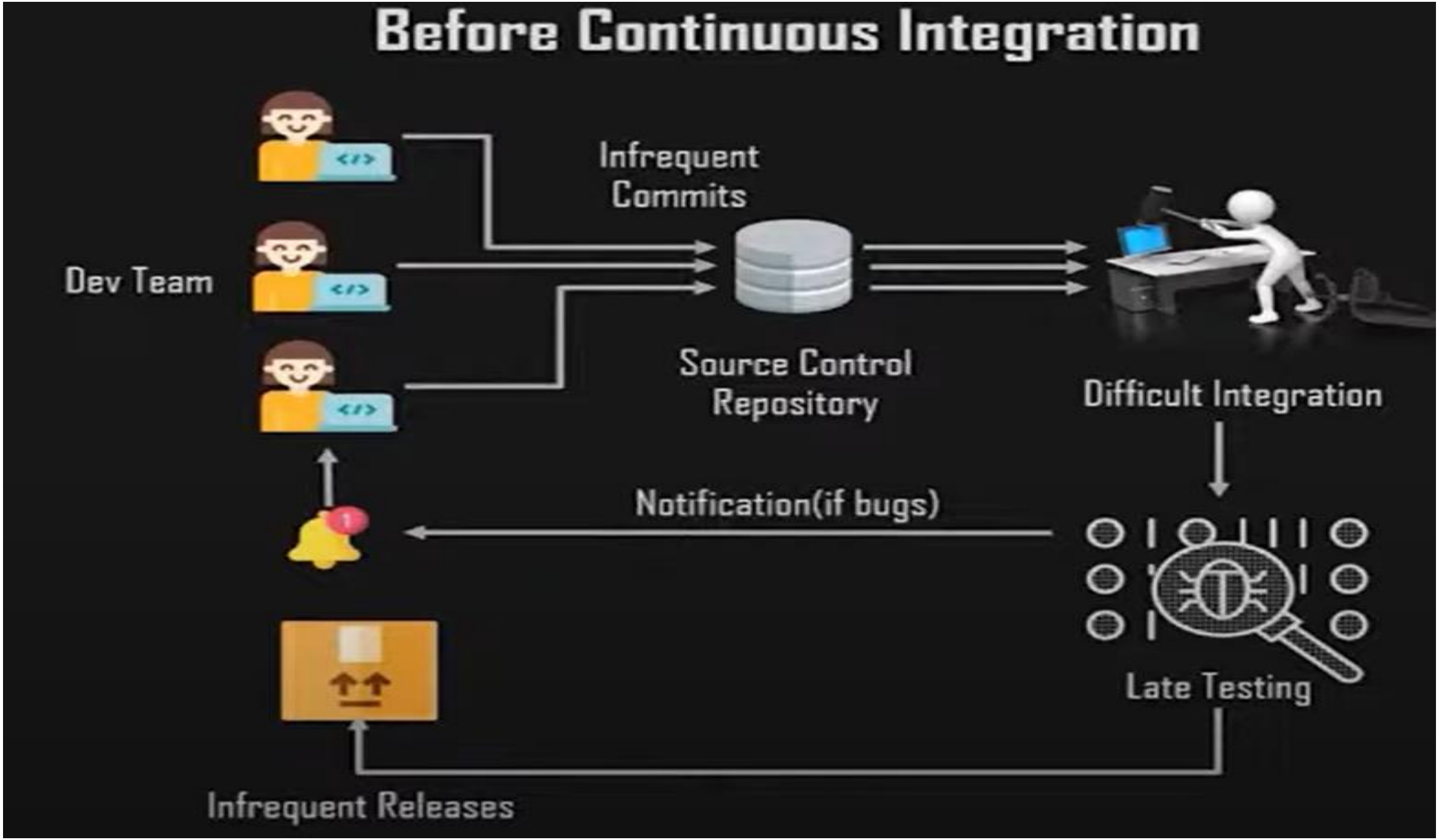
Introduction to Jenkins

- Jenkins achieves Continuous Integration with the help of plugins. Plugins allows the integration of Various DevOps stages. If you want to integrate a particular tool, you need to install the plugins for that tool. For example: Git, Maven etc.

What is Continuous Integration

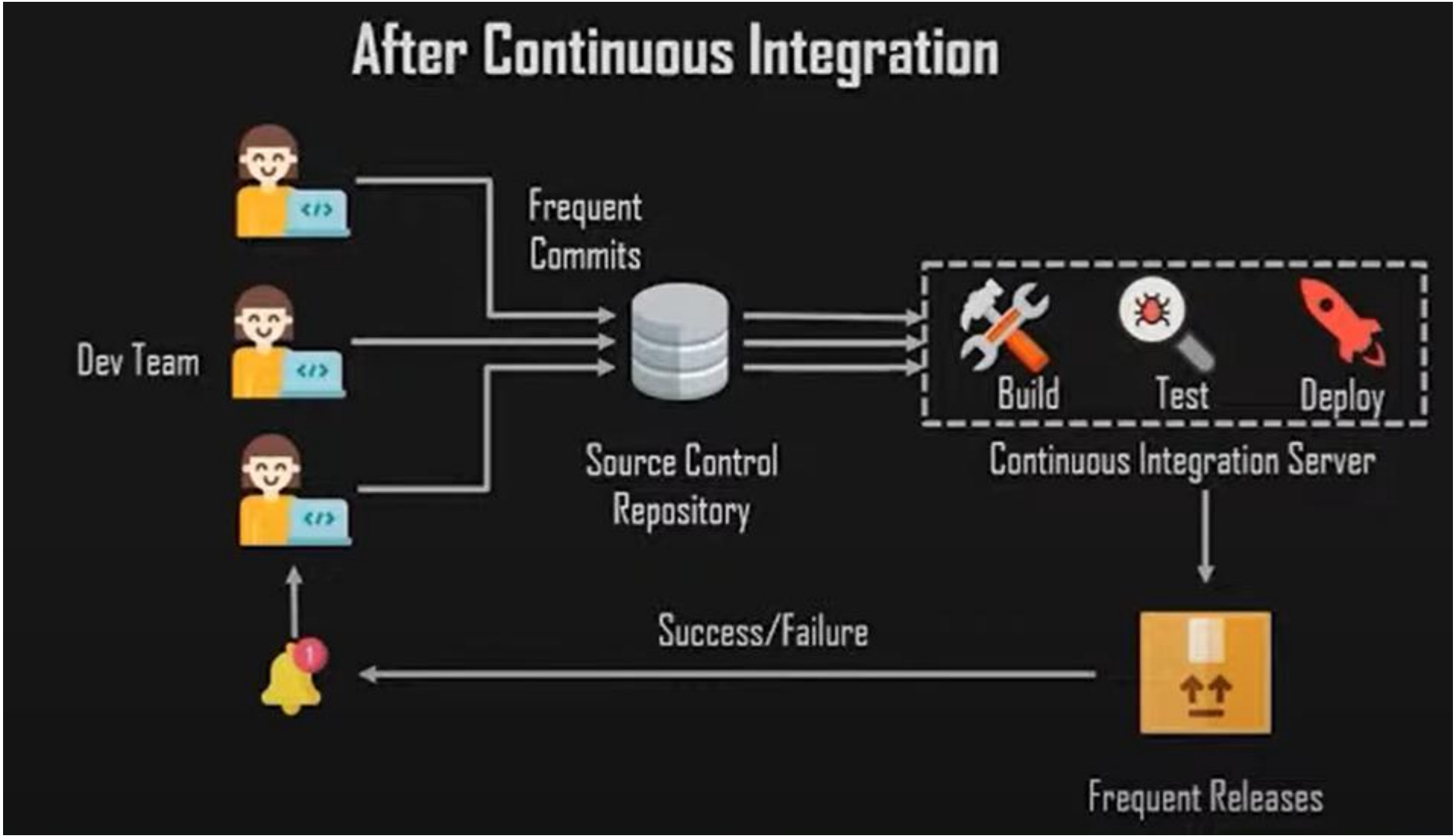
- Continuous Integration is a development practice in which the developers are required to commit changes to the source code in a shared repository several times a day or more frequently.
- Every commit made in the repository is then built. This allows the teams to detect the problems early. Apart from this, depending on the Continuous Integration tool, there are several other functions like deploying the build application on the test server, providing the concerned teams with the build and test results etc.




Before Continuous Integration
Infrequent
Commits
Dev Team
Source Control
Repository
Difficult Integration
Notification(if bugs)
Late Testing
Infrequent Releases

Issues faced before Continuous Integration

- Long Waits to test code
- Difficult to Debug
- Slow Software Delivery process
- Infrequent feedback cycles


After Continuous Integration
Frequent
Commits
Dev Team
Source Control
Repository
Build
Test
Deploy
Continuous Integration Server
Success/Failure
Frequent Releases

History of Jenkins

- The Jenkins project was started in 2004 (originally called Hudson) by Kohsuke Kawaguchi, while he worked for Sun Microsystems.
- Kohsuke was a developer at Sun and got tired of incurring the wrath of his team every time his code broke the build.
- The Hudson was renamed to Jenkins in 2011 after a dispute with Oracle, which had forked the project and claimed rights to the project name.
- The Oracle fork, Hudson, continued to be developed for a time before being donated to the Eclipse Foundation. Oracle's Hudson is no longer maintained and was announced as obsolete in February 2017.
- On April 20, 2016 version 2 was released with the Pipeline plugin enabled by default.
- The plugin allows for writing build instructions using a domain specific language based on Apache Groovy.

Other CI Tools

- Jenkins 48.11%
- TeamCity 6.83%
- Bamboo 1.76%
- Buddy
- GitLab CI
- CircleCI 5.61%
- TravisCI 1.51%

Stages in CI-CD

- The most common stages in CI-CD pipeline are
  - The Trigger
  - Code Checkout
  - Compile the code
  - Run unit tests
  - Package the code
  - Delivery or Deployment

Stages in CI-CD – The Trigger

- You can either configure your CI/CD tool to poll for changes to a Git repository, or schedule to trigger periodically
- The main reason for doing this is to make running the pipeline automatic, and friction-free.
- If you leave the pipeline to be run manually, people will sometimes forget to run it, or choose not to. It’s far better to just make this an automated step.
- There is huge confidence that comes from knowing that your code is going to be tested on every single change.

Stages in CI-CD – Code Checkout

- In this stage, the CI server will check out the code from the source code repository, such as GitHub or Bitbucket.
- The CI/CD tool usually receives information from a poll, or a scheduler, which says which specific commit triggered the pipeline.
- The pipeline then checks out the source code at a given commit point, and starts the process.
- Continuous Download

Stages in CI-CD – Compile the code

- If you’re developing in a compiled language like Java, the first thing you’ll probably need to do is compile your program.

- This means that your CI tool needs to have access to whatever build tools you need to compile your app. For example, if it’s Java, you’ll use something like Maven or Gradle.

- Ideally this stage should run in a clean, fresh environment.

- Continuous Build

Stages in CI-CD – Run tests

- The next key element of your CI/CD pipeline is unit testing. This is the stage where you configure your CI/CD tool to execute the tests that are in your codebase.

- You might use Maven or Gradle to do this in Java, or test in JavaScript.

- The aim at this point is not only to verify that all the unit tests pass, but that the tests are being maintained and enhanced as the code base grows.

- If the application is growing rapidly, but the number of tests stays the same, this isn’t great, because it could mean that there are large parts of the code base which are untested.

- Continuous Testing

Stages in CI-CD – Package the code

- Once all of the tests are passing, you can now move on to packaging the code. Exactly how you package your application depends on your programming language and target environment.

- If you're using Java, you might build a JAR file. If you're using Docker containers, you might build a Docker image.

- Whatever packaging format you choose, a good practice is that you should build the binary only once. Don't build a different binary for each environment, because this will cause your pipeline to become very complex.

- Continuous Delivery

Stages in CI-CD – Delivery or Deployment

- Finally, when the application has been tested, it can move into the delivery or deployment stage.

- At this stage, you have an artifact ready to be deployed (continuous delivery). Or, you can continue to CI/CD heaven and automatically deploy your software (continuous deployment).

- To achieve continuous deployment, you need a production environment to deploy into. If you're deploying to a public cloud, you can use the cloud provider's API to deploy your application. Or if you're using Kubernetes, you might use a Helm chart to deploy your app.

- And finally, this is the end of your pipeline!

- The next time a developer commits code into the repository, it will be run all over again.

Why Jenkins is so powerful?

- **Open-Source:** This is obviously a key reason for Jenkins' widespread appeal. It's free, so any organization can get started with it regardless of budgetary constraints. It's also easy to install: Jenkins was developed as a self-contained Java program, which means it can run on most devices and operating systems.
- **Community Support:** Because it is open-source and has been the go-to CI tool for many years, Jenkins now has a strong community behind it. This reinforces Jenkins' popularity, as it's easy to find tutorials and other resources to get the most out of it.
- **Plugins:** One of the main benefits of Jenkins is the huge catalog of plugins that can extend its functionality. At latest count, Jenkins has more than 1,800 community-developed plugins that allow any organization to adapt or enhance the base functionality to suit their own DevOps environment.
- **Distributed:** One server is enough for complex projects, so Jenkins uses a Master-Slave architecture to manage distributed builds. The main server is the 'Master', which can then distribute workload to other 'slave' servers, allowing for multiple builds and test environments to be running simultaneously. This approach could be used, for example, to build and test code on different operating systems.

Jenkins Setup

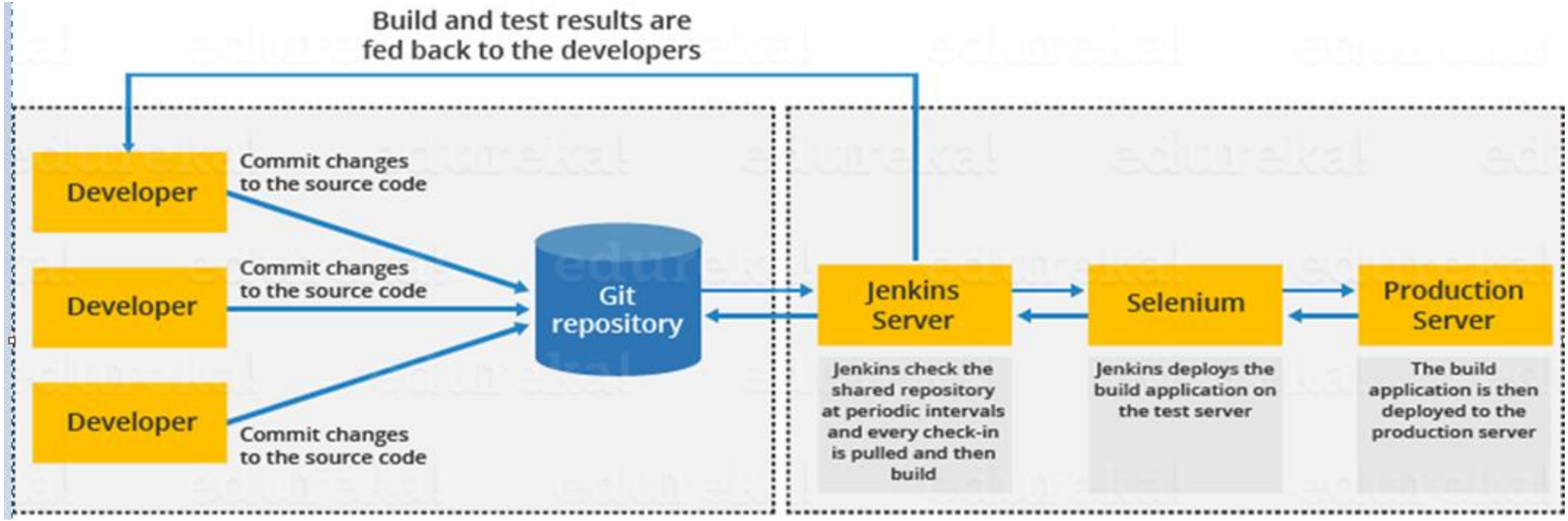
- Follow the Instructions given in below URL

Jenkins download and deployment

- Once Jenkins is installed, it runs on port no 8080 by default

    - Access the Jenkins from browser
    - Unlock Jenkins
    - Install Plugins
    - Create Admin user

Jenkins Architecture



Alternatives to setup the Jenkins

- Using Package Manager
  - Chocolatey
  - Brew
  - Yum
- Using Containers

Jenkins Job types

- Jenkins is a flexible and highly configurable automation server that can be used to automate many different types of tasks. Some common types of jobs in Jenkins include:
- **Freestyle project**: a basic type of Jenkins job that provides a wide range of options for customizing and configuring the build process.
- **Pipeline:** a type of Jenkins job that uses a Groovy script to define the entire build process, from building and testing the code to deploying it to production.
- **Multi-configuration project:** a type of Jenkins job that allows for running the same build with different configurations.
- **Folder:** a type of Jenkins job that allows you to organize and manage multiple jobs in a hierarchical structure.
- **Maven project:** a type of Jenkins job that is optimized for building and testing Maven-based projects.
- **Multi-Branch project:** a type of Jenkins job that allows for running the same build with different branches.

CRON Jobs in Jenkins

- Cron is the baked in task scheduler - run things at fixed times, repeat them etc. In fact, Jenkins uses something like the cron syntax when you are specifying specific times you want a Job to run.
- Syntactically, CRON contains 5 places for each time entry, thus a conventional one-liner syntax of the CRON template would be somehow like:

0 23 * * *
// Staged as
{Minute} {Hour} {DayOfMonth} {Month} {DayofWeek}

CRON Jobs in Jenkins

- **Minute (0-59)** – tells at what minute of the hour the Job should build.
- **Hour (0-23 With 0 is the midnight)** - tells at what hour the job should build in the 24-hour clock.
- **Day of the month (1-31)** - tells on which day of the month your job should build, e.g., 23rd of each month.
- **Month (1-12)** - tells the month when your job should build. You can number or name the month as well. i.e., April, May, etc.
- **Day of the week (0-7 With 0 or 7 both being Sunday)** - tells the day of the week when you want to build your job. It can also be a number or name like Mon etc.

- // Every day at 6 in the evening
- 0 18 * * *
- You need to make the first entry (minute) fixed if you want to work on the hour. The above expression means to run your job at the 0th minute of 18th hour DAILY without any specification of day, month or week.

CRON Jobs in Jenkins

- After every 20 minutes
  *0/20 * * * *
- After every 4 hours
  0 */4 * * *
  // OR
  0 0,4,8,12,16,20 * * *
- In the first case, it will run at 0th minute, 20 minutes after, 40 minutes after means after every 20 minutes. 20 is the offset here which tells after how many minutes this should trigger.
- Similarly, in the next example, the first entry (minute) is fixed and the second entry (hour) contains an offset which tells keep repeating it after 4 hours. In the next line, the less verbose and more readable syntax of the same CRON is also given.

- Every day at 11 in the morning and at night both
  0 11, 22 * * *
- You can define the multiple values for same entry using a comma. Here, the hour place has two values, first for 11 am and the next for 11 pm.

CRON Jobs in Jenkins

- Every Saturday and Sunday at 5 PM

  0 17 * * 6,7
- This will run every Saturday (6) and Sunday (7) at 5 pm (17).

Q & A