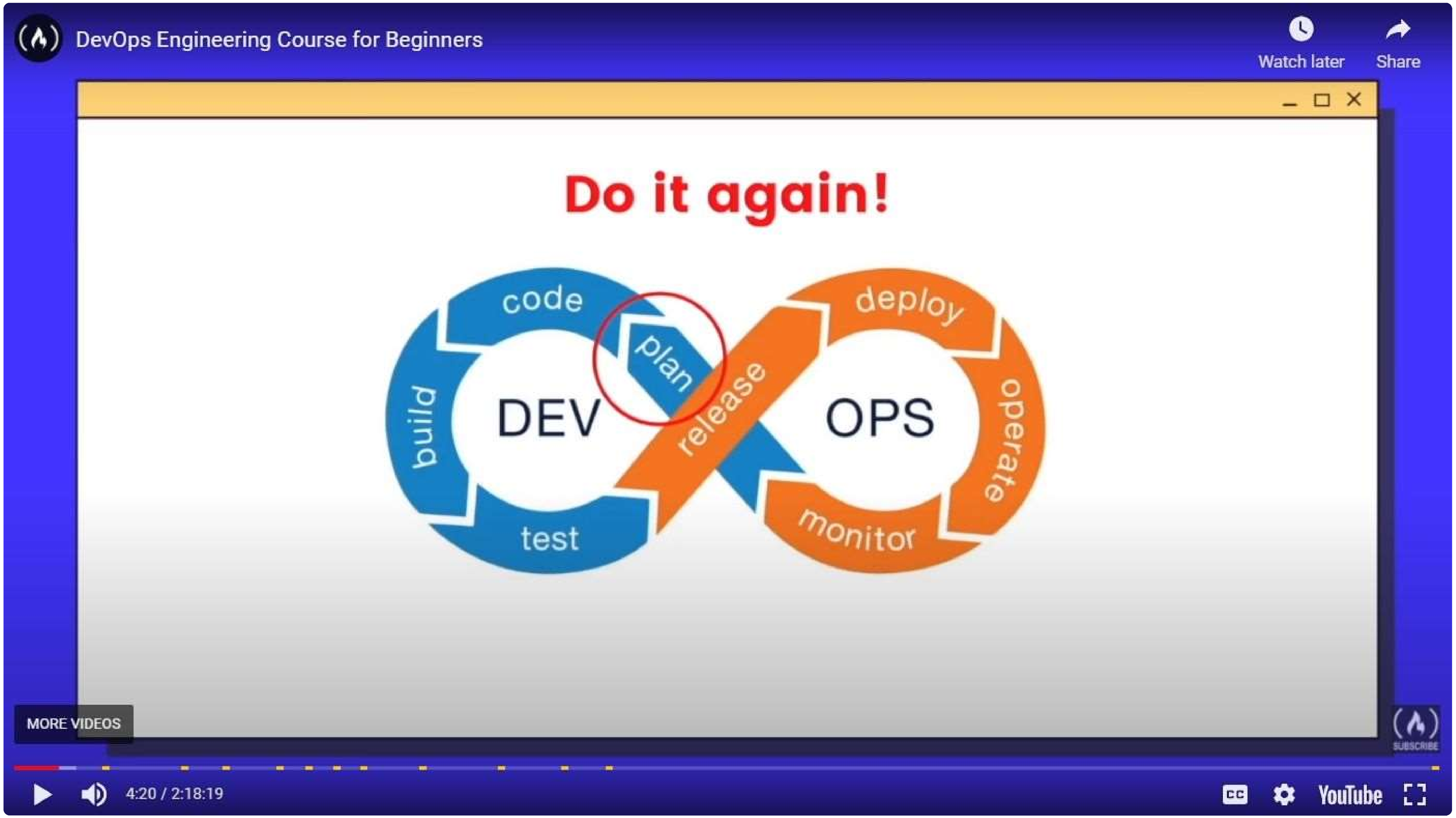
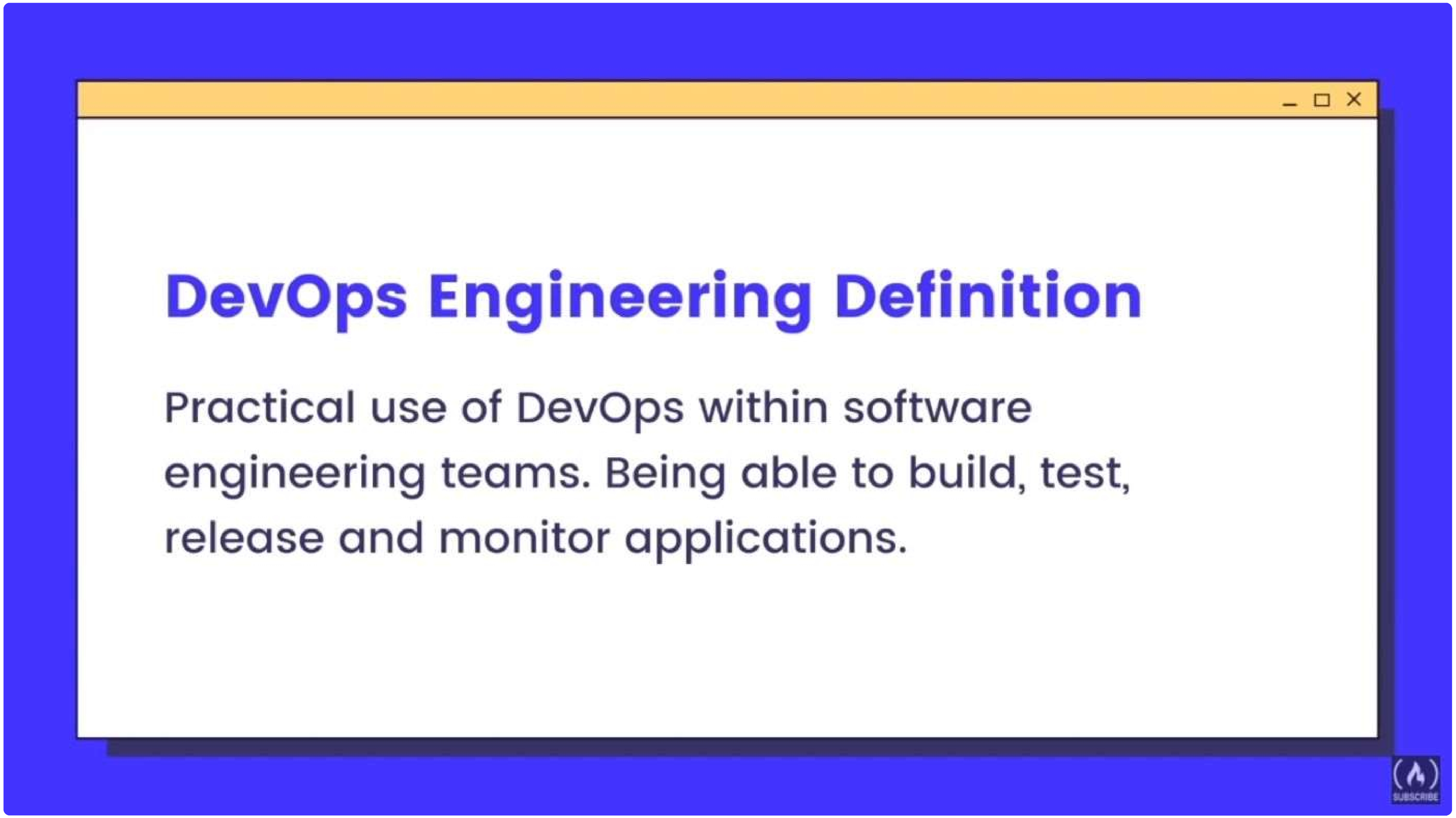

DevOps Engineering Course for Beginners
Watch later
Share
MORE VIDEOS
4:20 / 2:18:19
YouTube
SUBSCRIBE

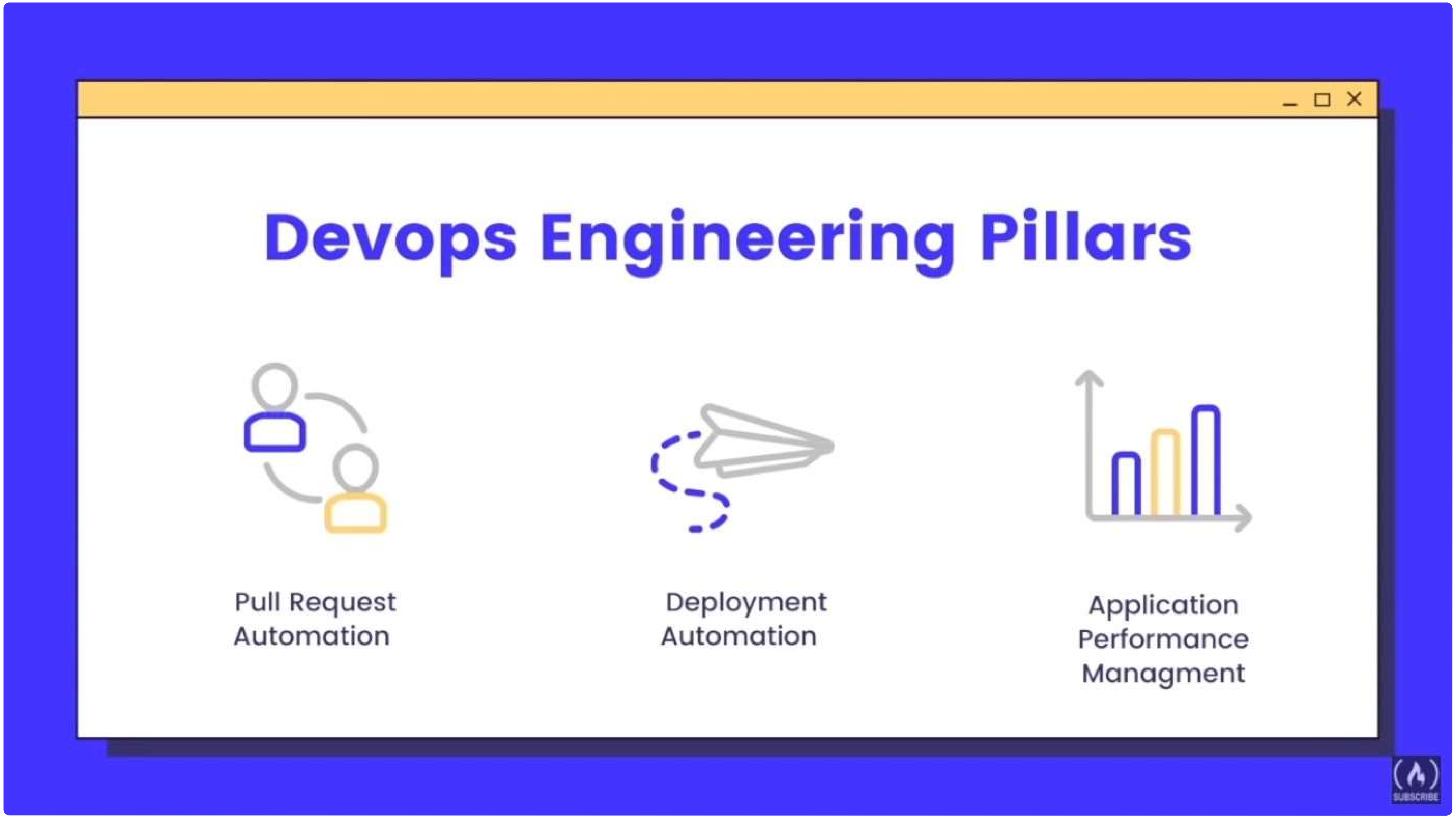
Do it again!

code
plan
build
DEV
release
test
deploy
OPS
operate
monitor



DevOps Engineering Definition

Practical use of DevOps within software engineering teams. Being able to build, test, release and monitor applications.

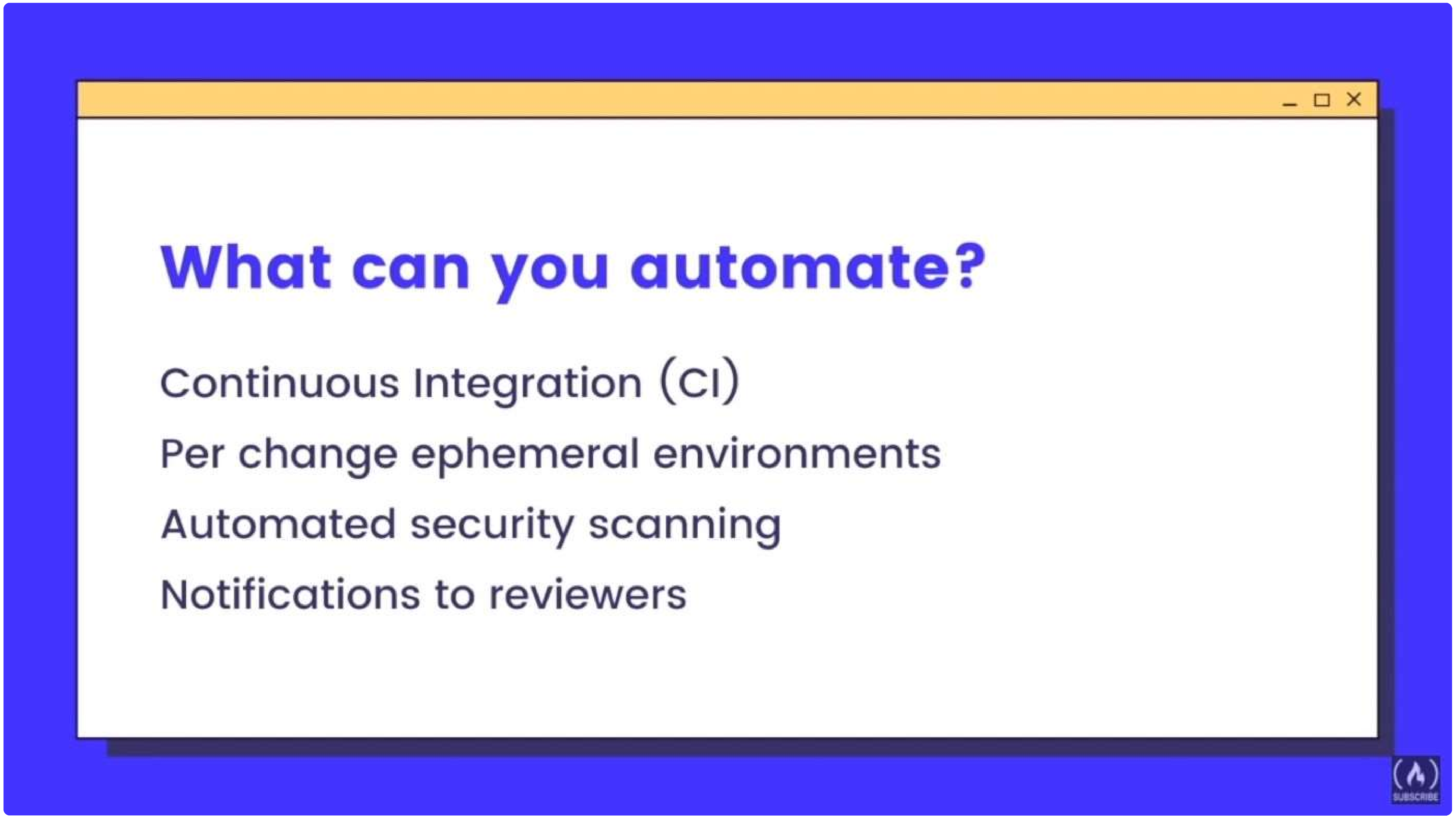


Devops Engineering Pillars

Pull Request Automation

Deployment Automation

Application Performance Managment



What can you automate?

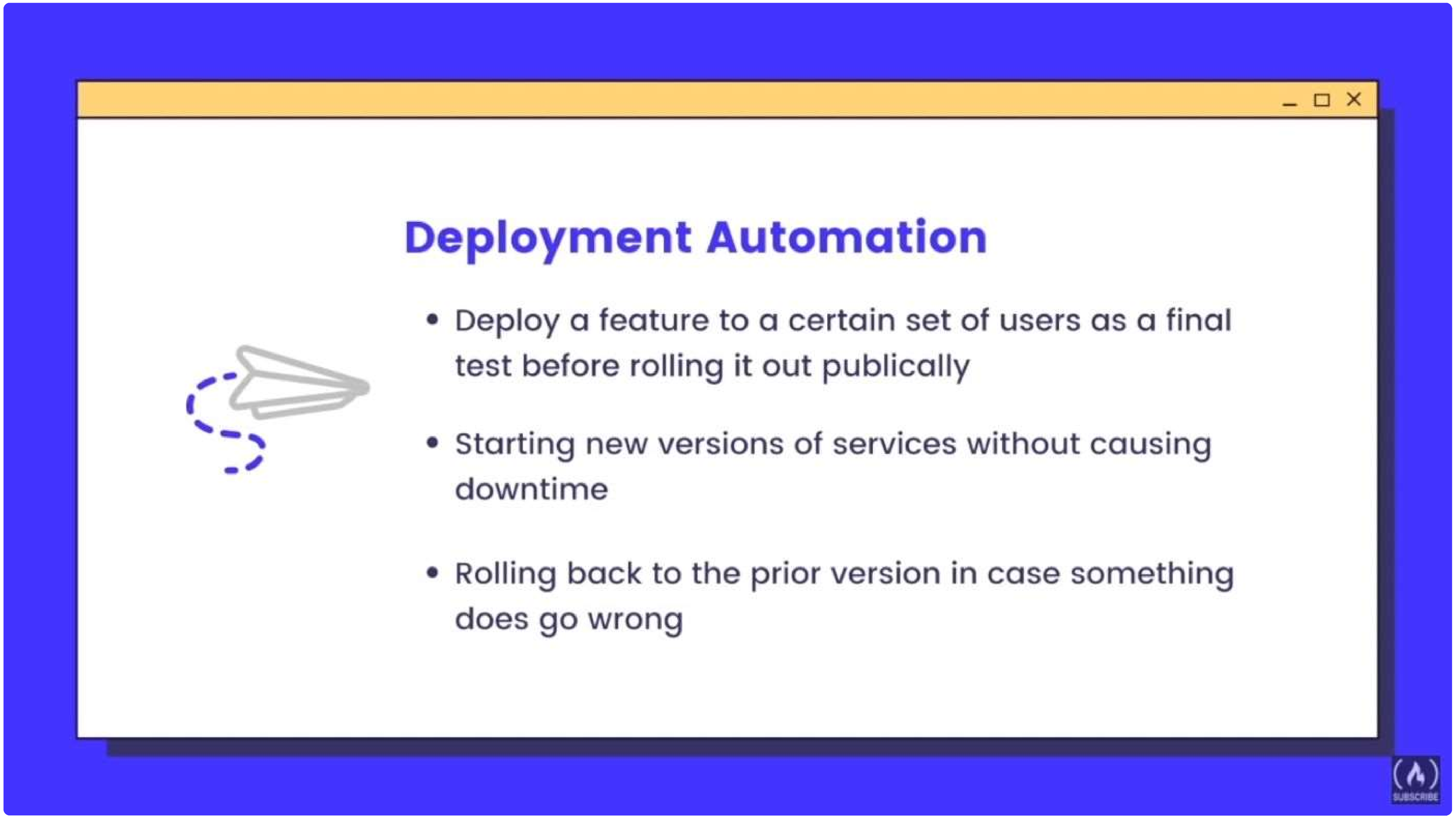
Continuous Integration (CI)

Per change ephemeral environments

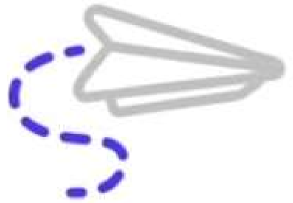
Automated security scanning

Notifications to reviewers

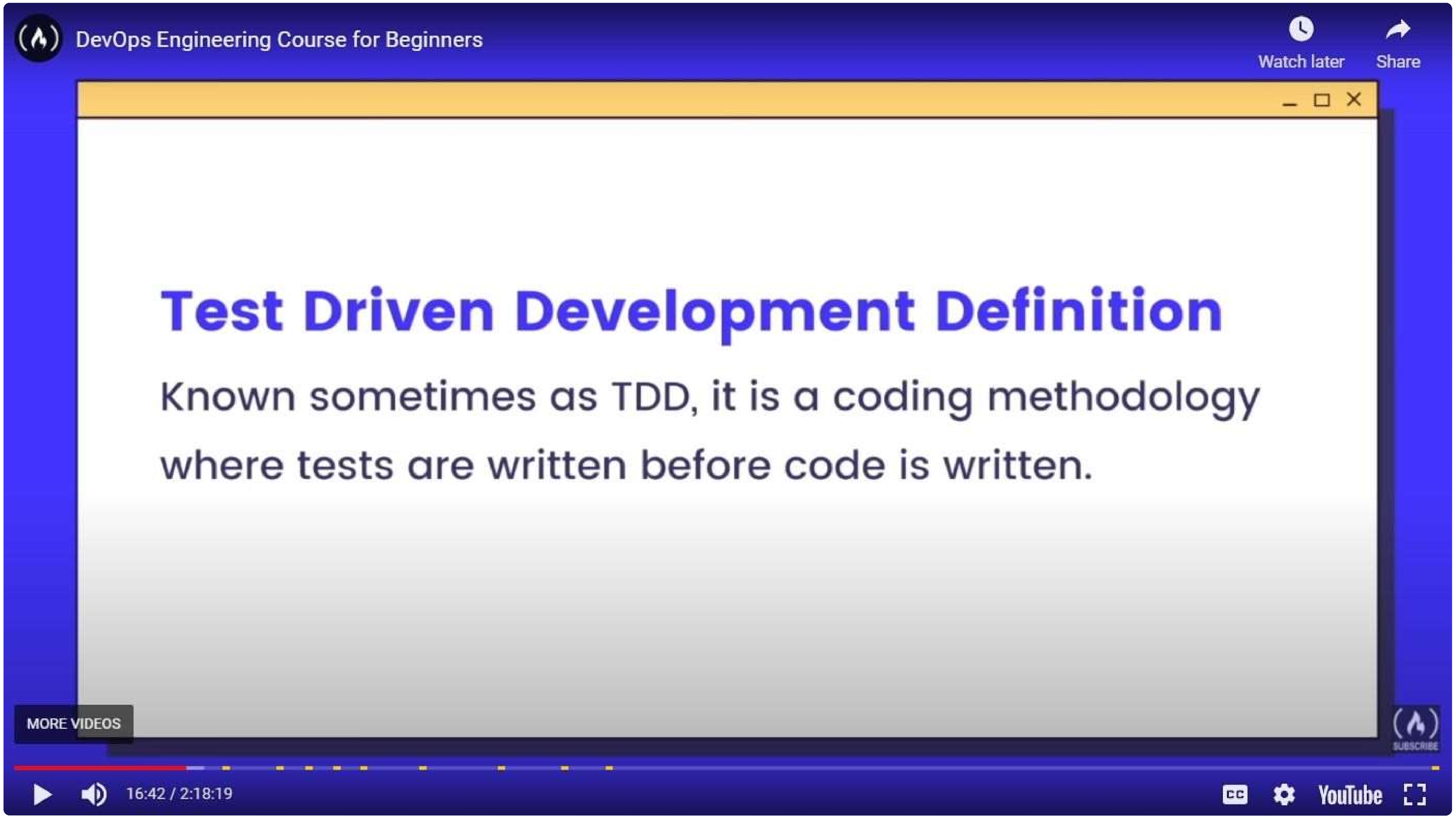
"Can you make a build in one step?"



Deployment Automation

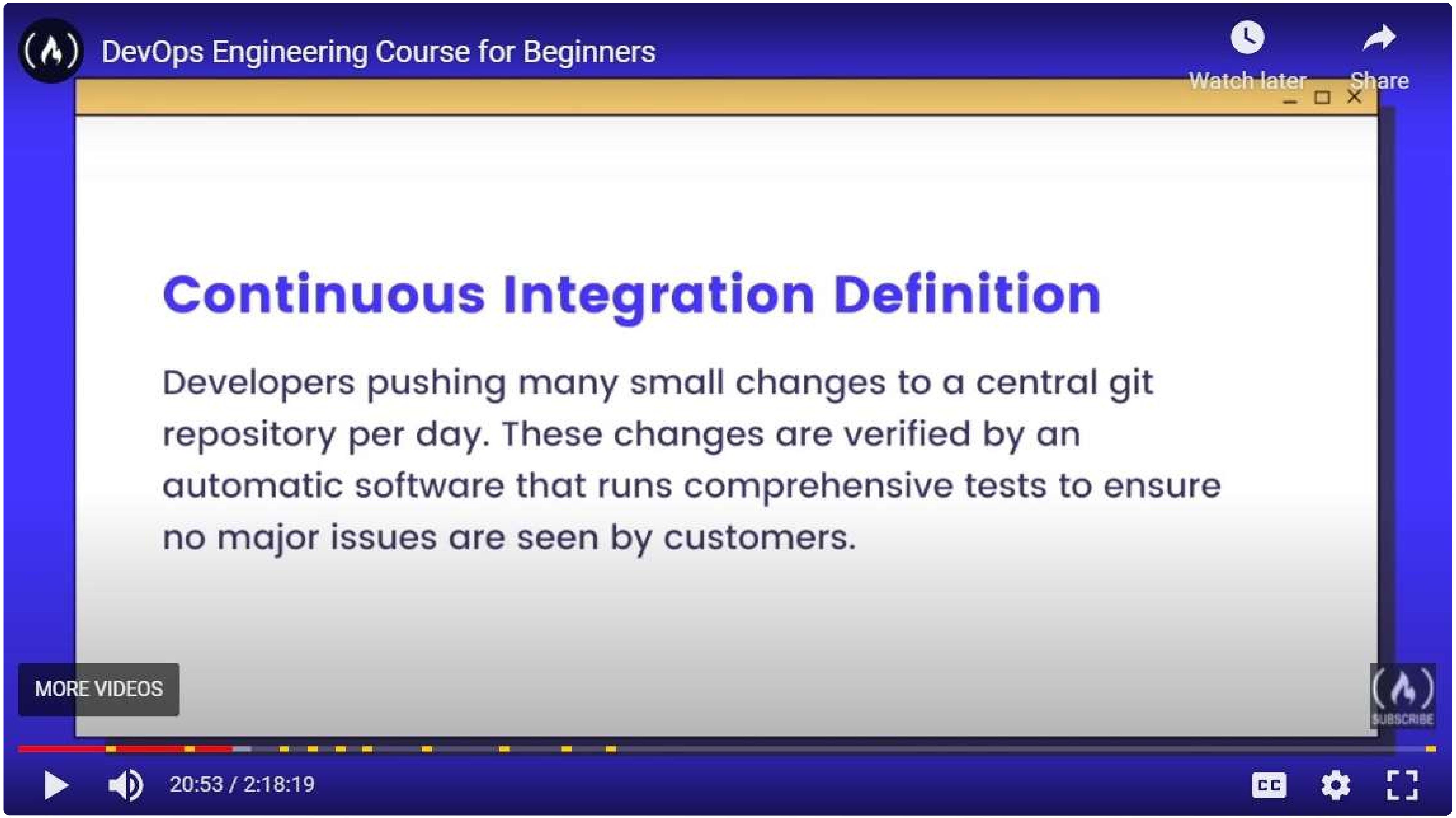


- Deploy a feature to a certain set of users as a final test before rolling it out publically
- Starting new versions of services without causing downtime
- Rolling back to the prior version in case something does go wrong



Test Driven Development Definition

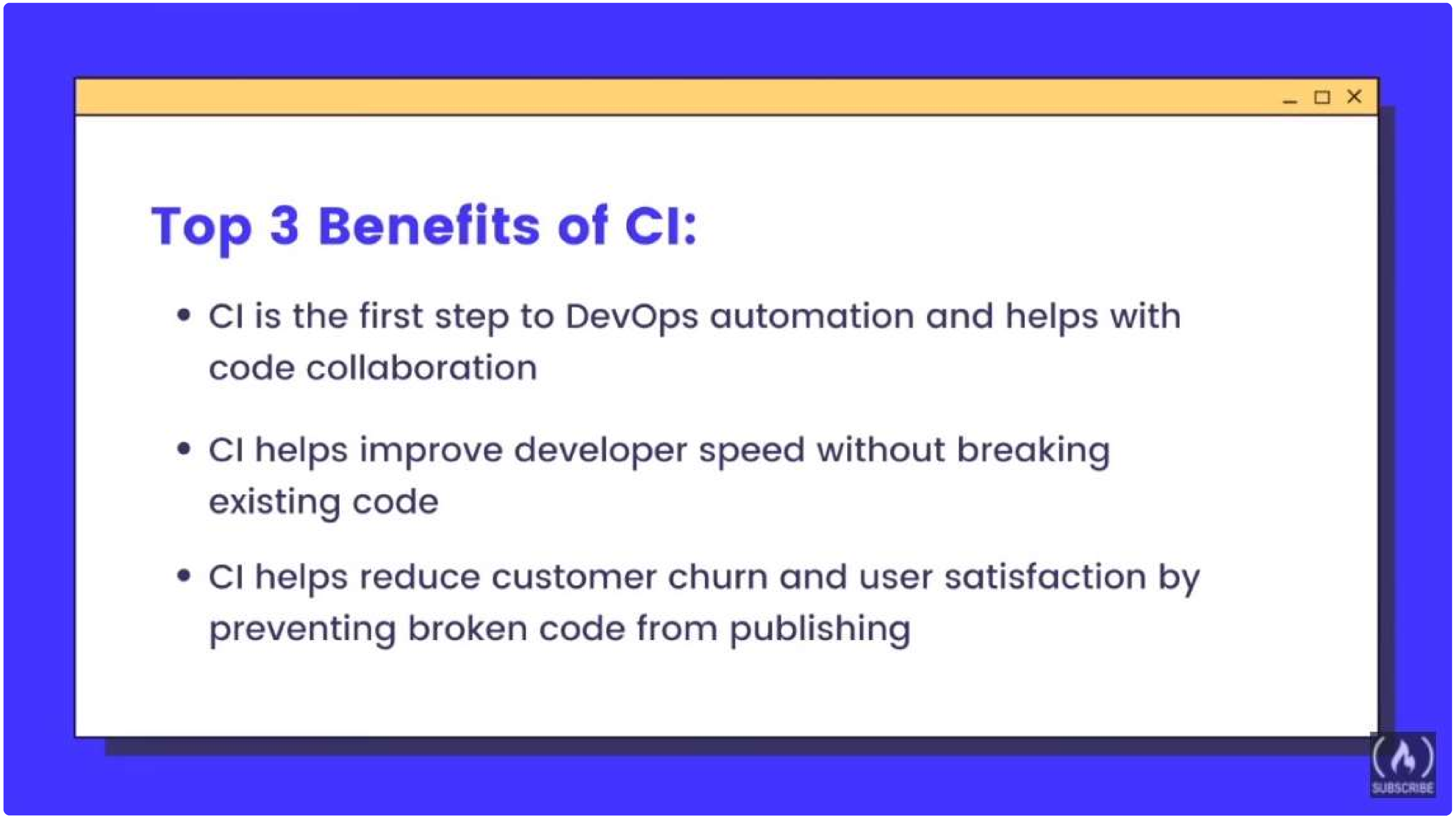
Known sometimes as TDD, it is a coding methodology where tests are written before code is written.



DevOps Engineering Course for Beginners

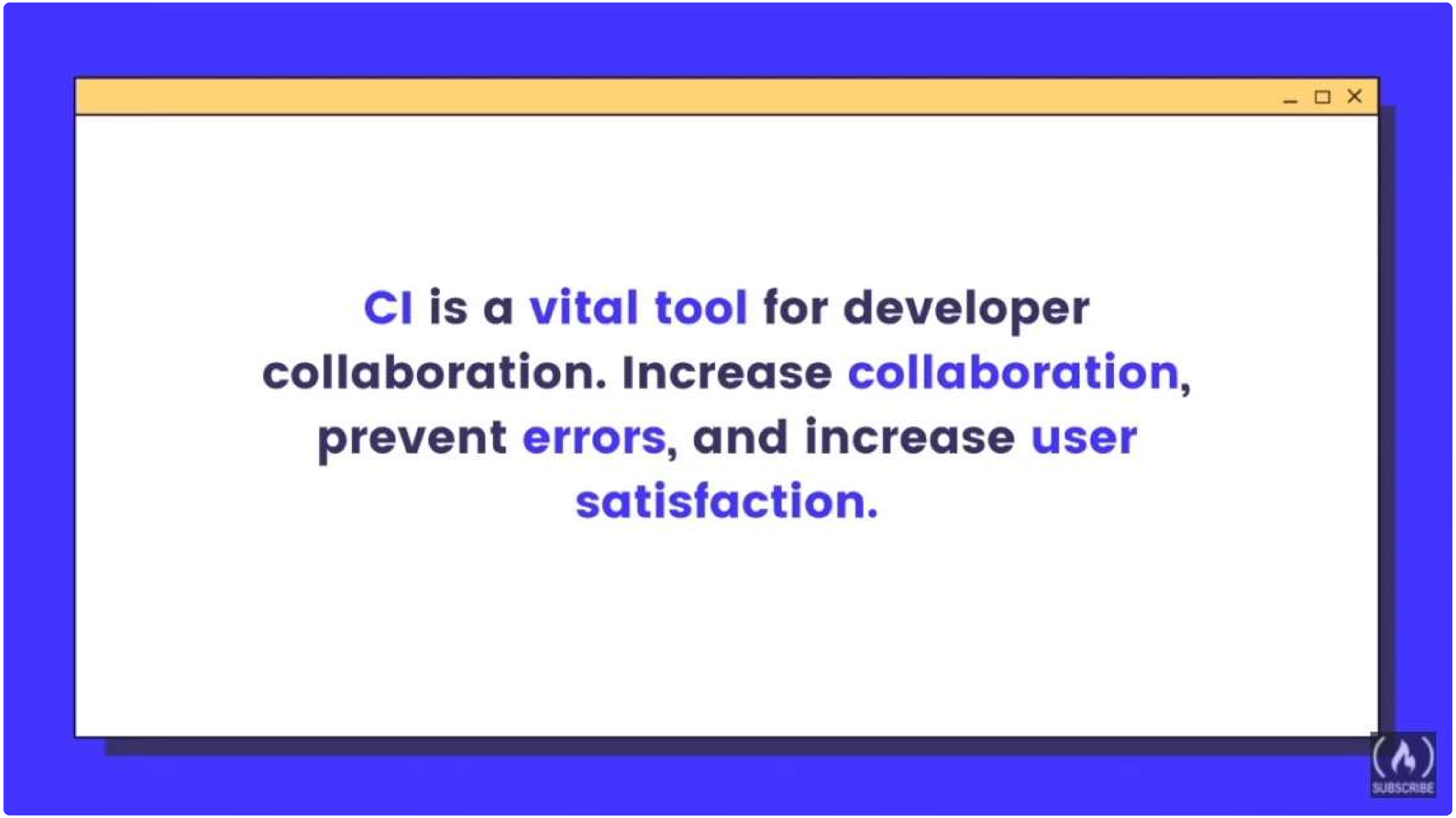
Continuous Integration Definition

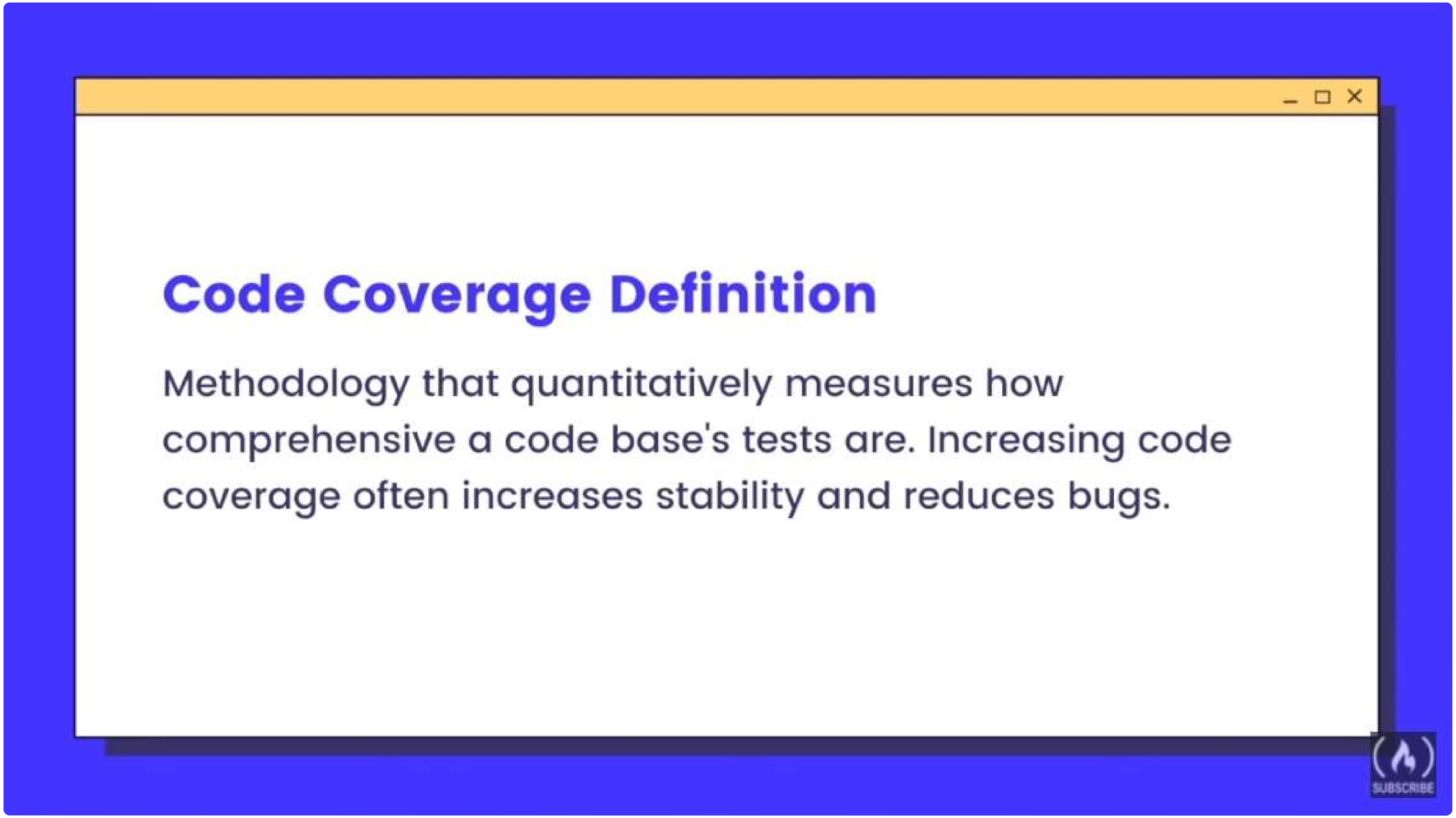
Developers pushing many small changes to a central git repository per day. These changes are verified by an automatic software that runs comprehensive tests to ensure no major issues are seen by customers.



Top 3 Benefits of CI:

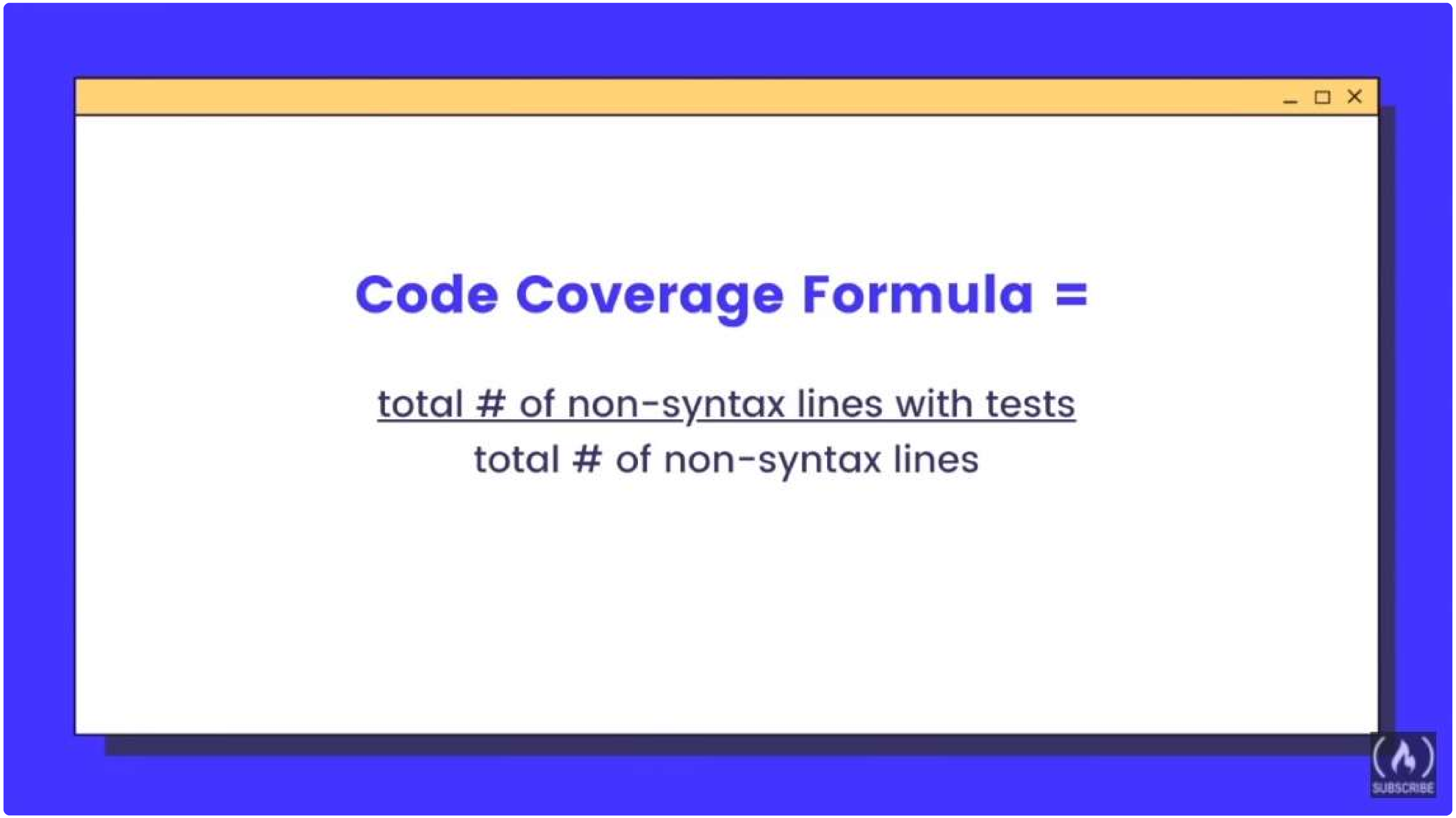
- CI is the first step to DevOps automation and helps with code collaboration
- CI helps improve developer speed without breaking existing code
- CI helps reduce customer churn and user satisfaction by preventing broken code from publishing


CI is a vital tool for developer collaboration. Increase collaboration, prevent errors, and increase user satisfaction.



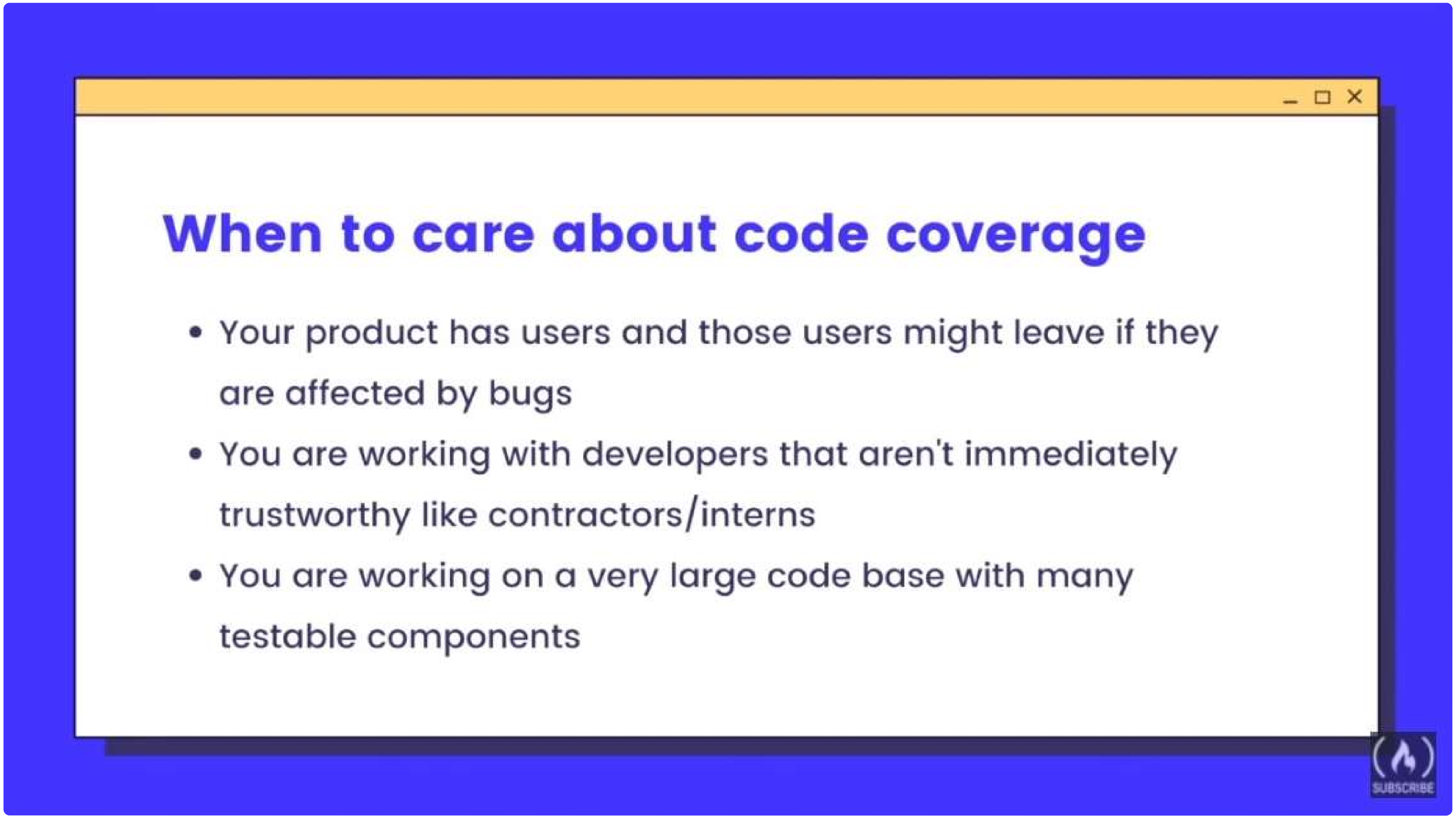
Code Coverage Definition

Methodology that quantitatively measures how comprehensive a code base's tests are. Increasing code coverage often increases stability and reduces bugs.



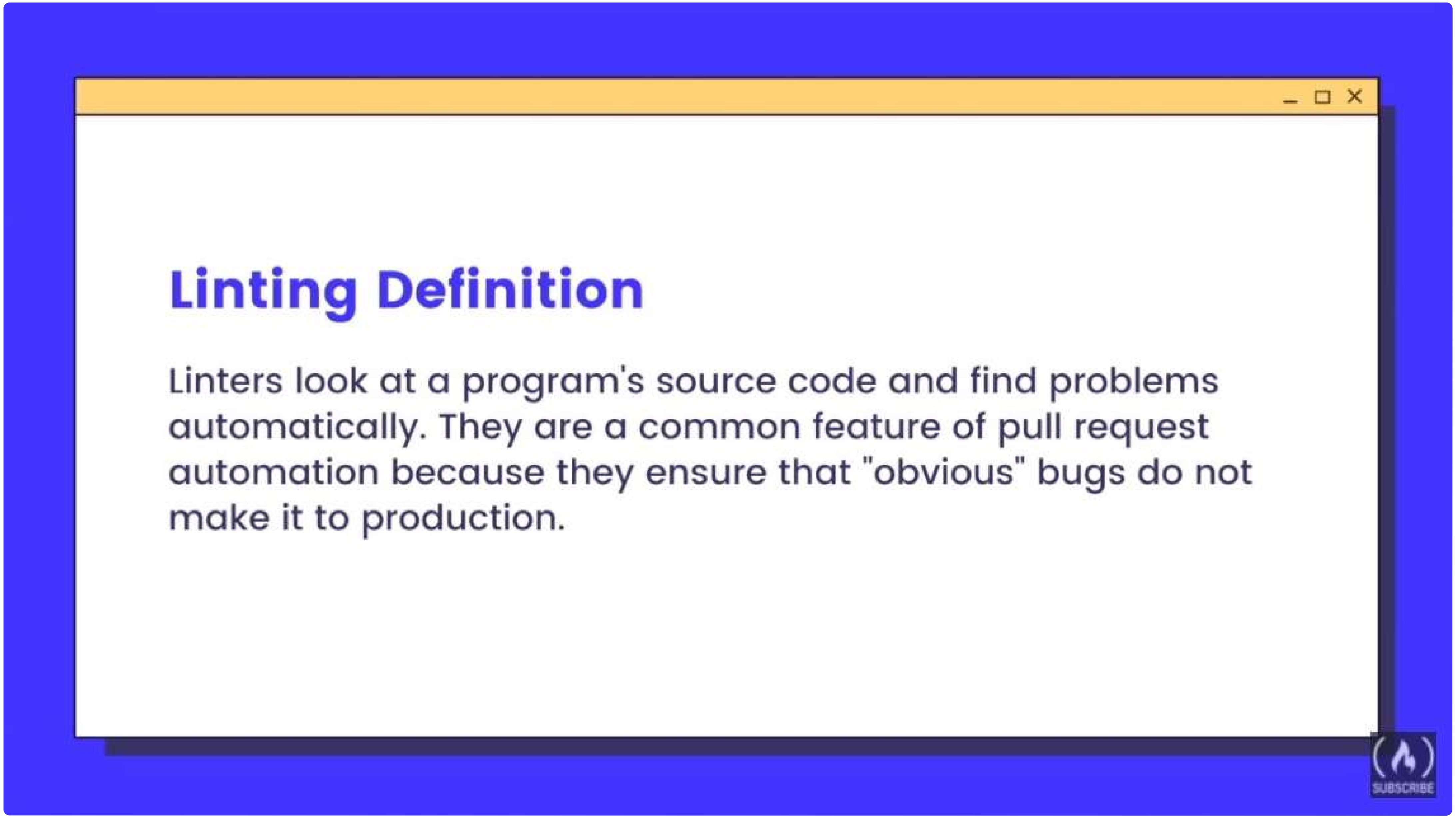
Code Coverage Formula =

$$\frac{\text{total \# of non-syntax lines with tests}}{\text{total \# of non-syntax lines}}$$



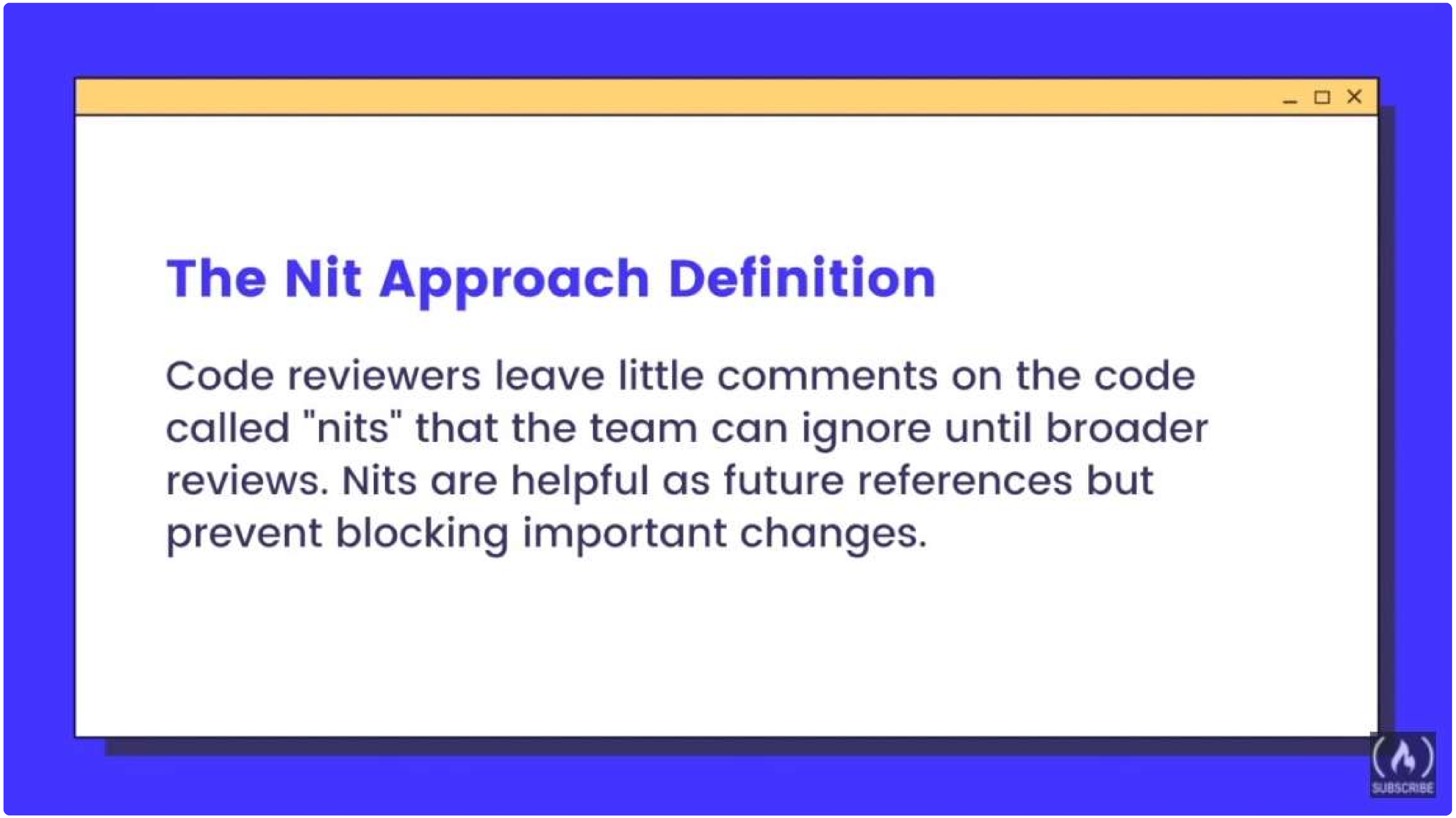
When to care about code coverage

- Your product has users and those users might leave if they are affected by bugs
- You are working with developers that aren't immediately trustworthy like contractors/interns
- You are working on a very large code base with many testable components



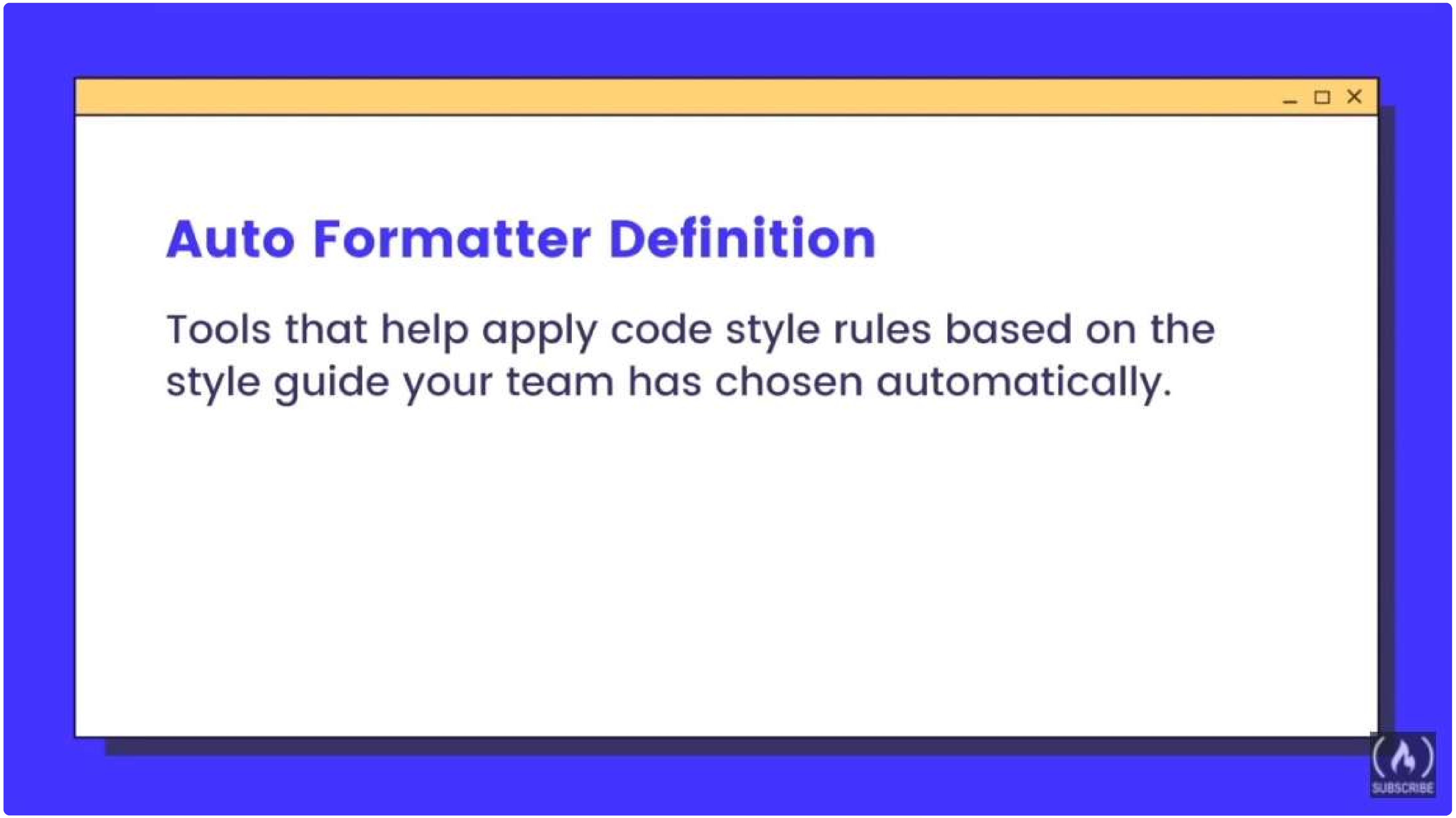
Linting Definition

Linters look at a program's source code and find problems automatically. They are a common feature of pull request automation because they ensure that "obvious" bugs do not make it to production.



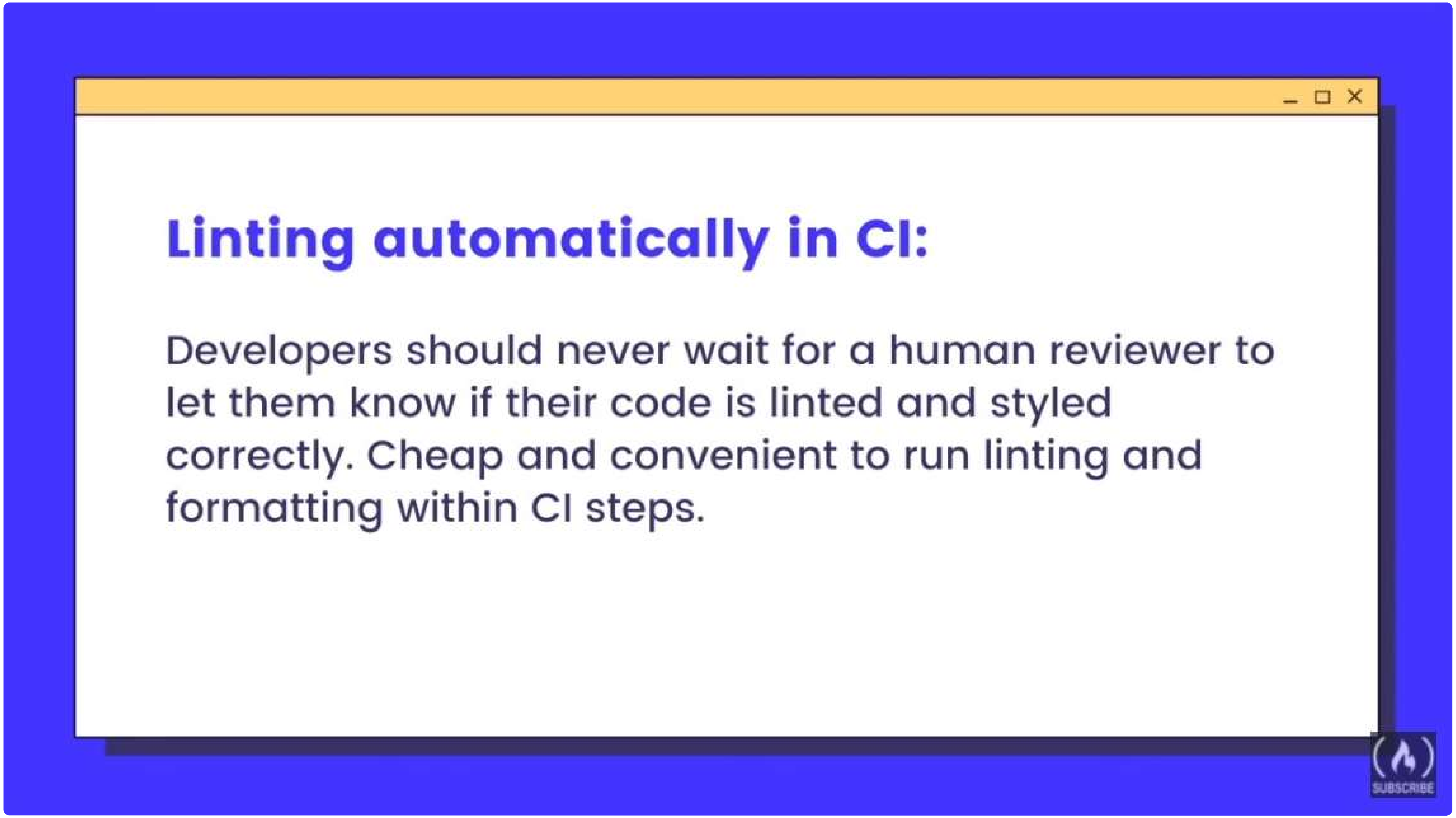
The Nit Approach Definition

Code reviewers leave little comments on the code called "nits" that the team can ignore until broader reviews. Nits are helpful as future references but prevent blocking important changes.



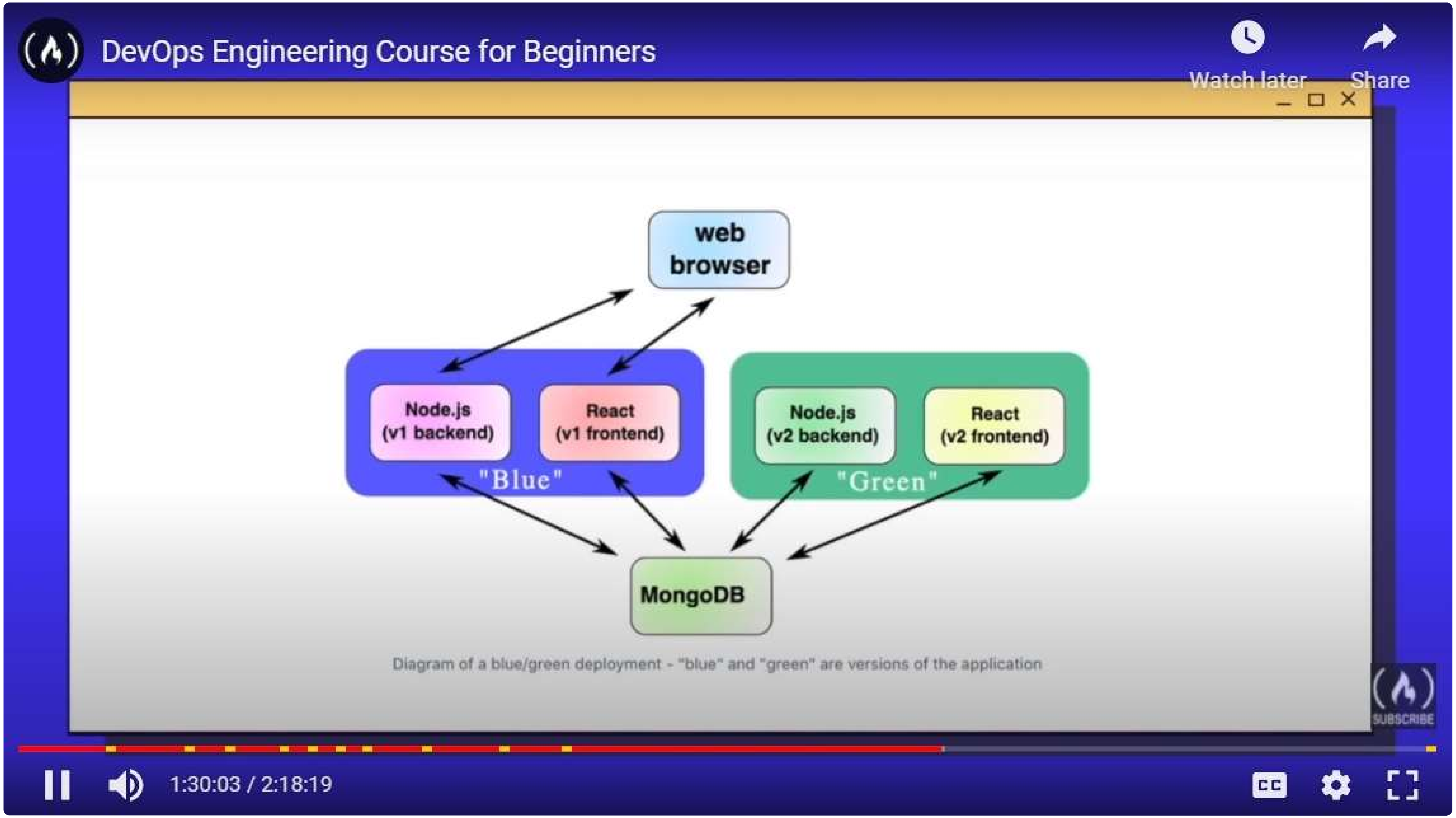
Auto Formatter Definition

Tools that help apply code style rules based on the style guide your team has chosen automatically.



Linting automatically in CI:

Developers should never wait for a human reviewer to let them know if their code is linted and styled correctly. Cheap and convenient to run linting and formatting within CI steps.



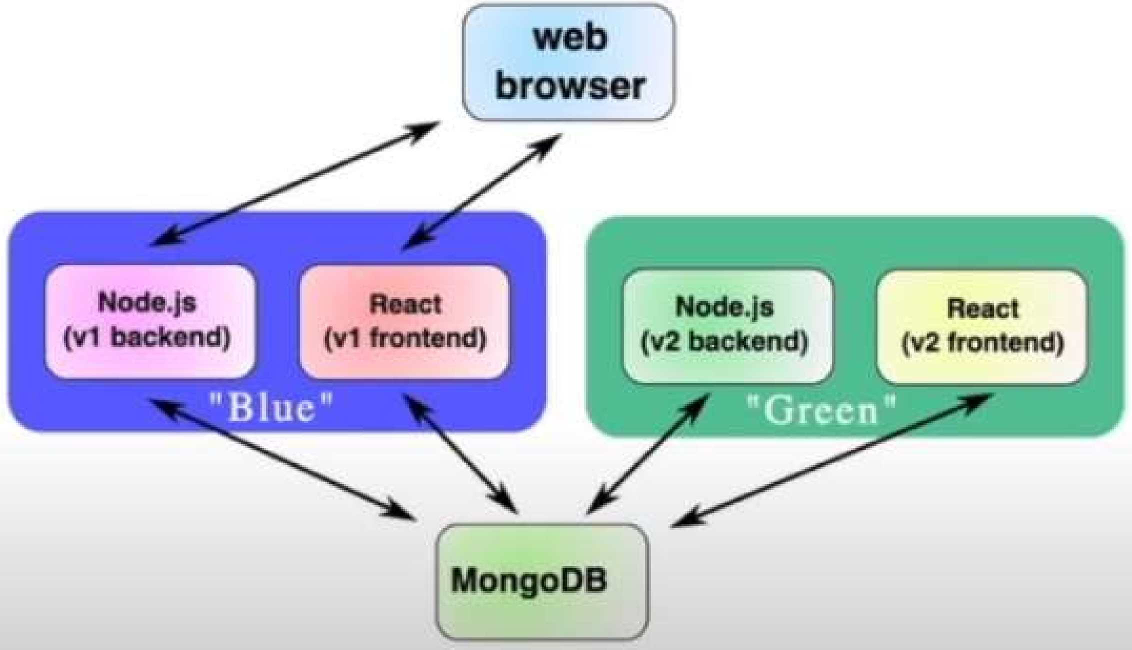
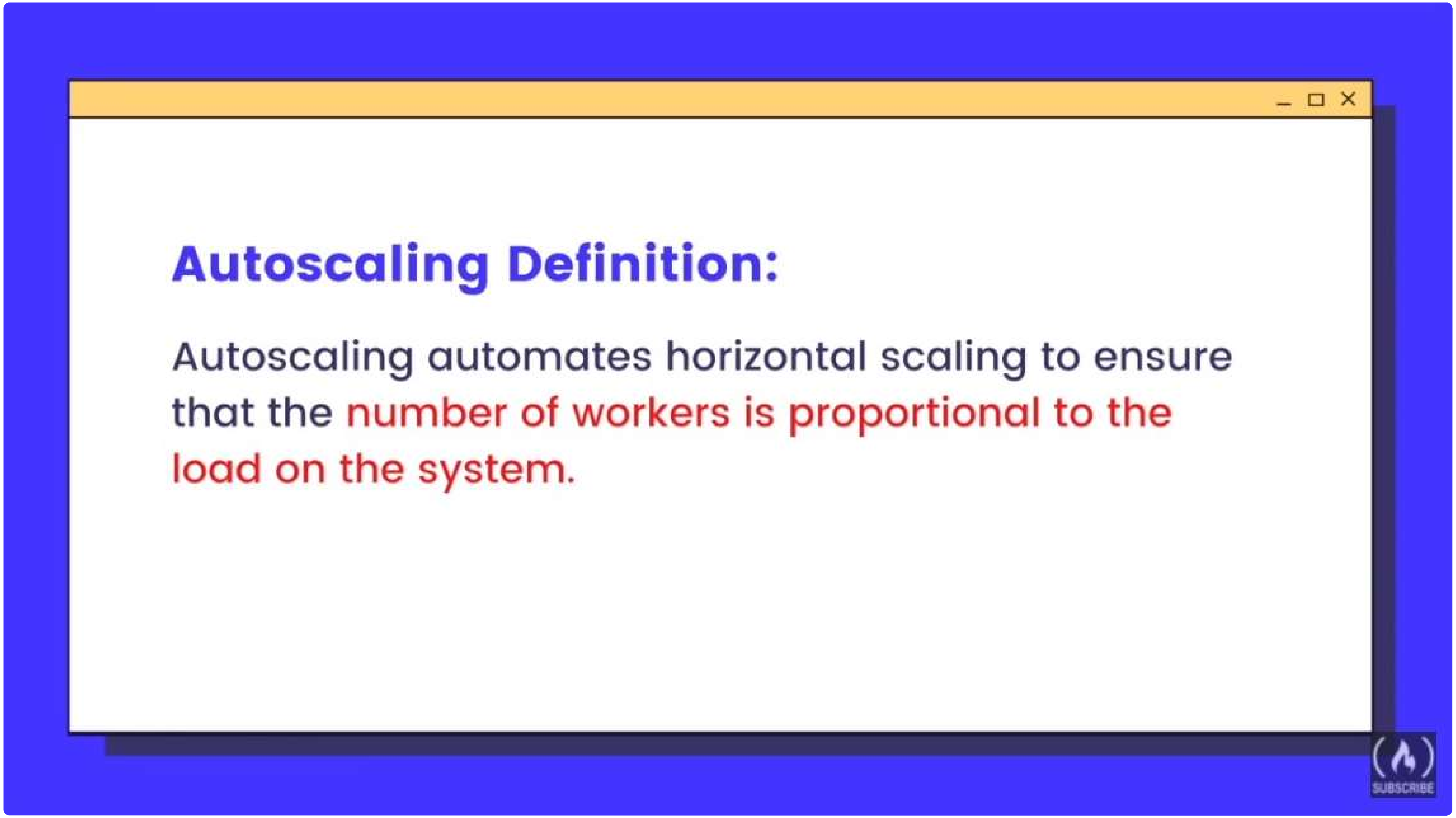
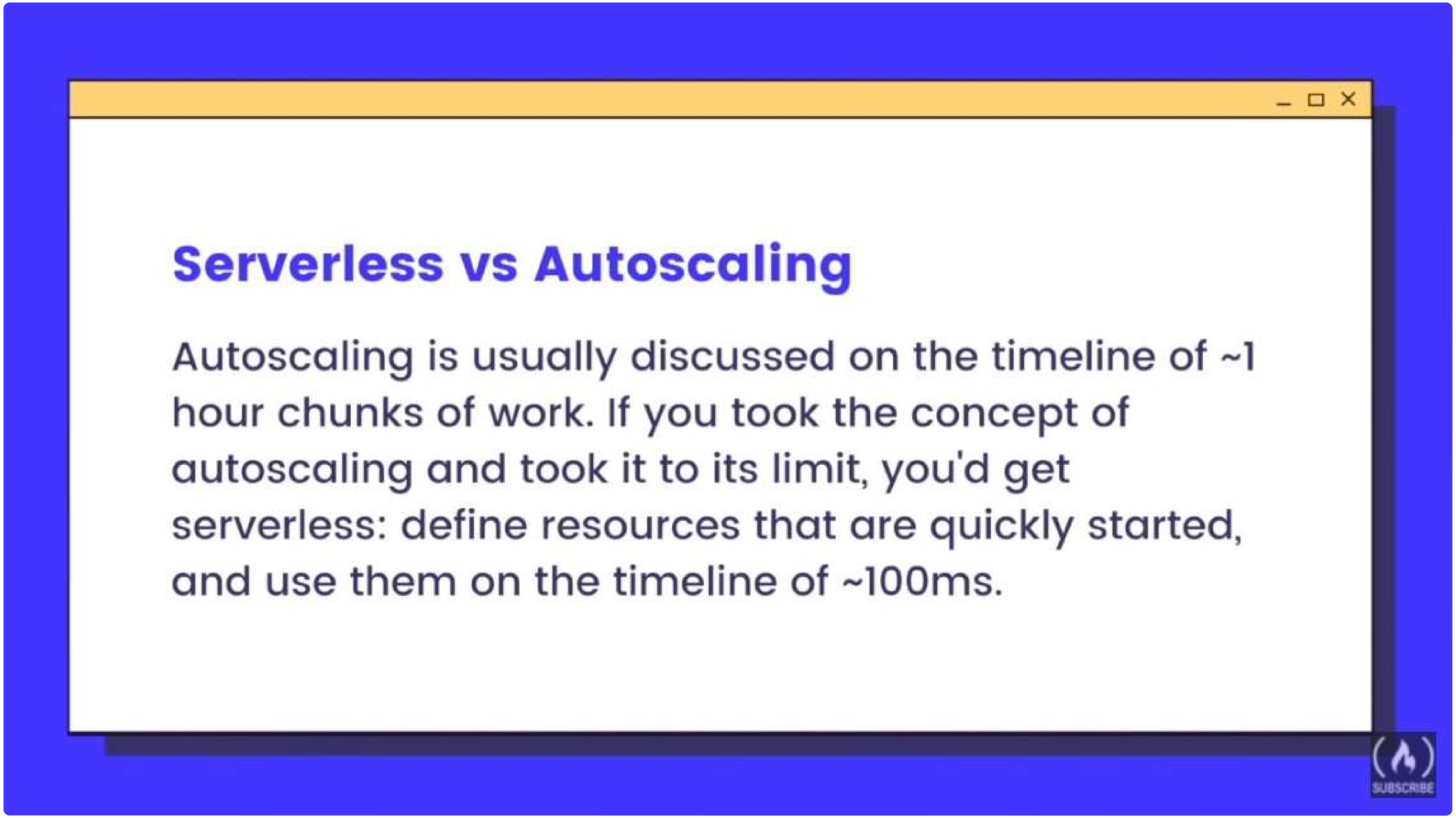


Diagram of a blue/green deployment - "blue" and "green" are versions of the application



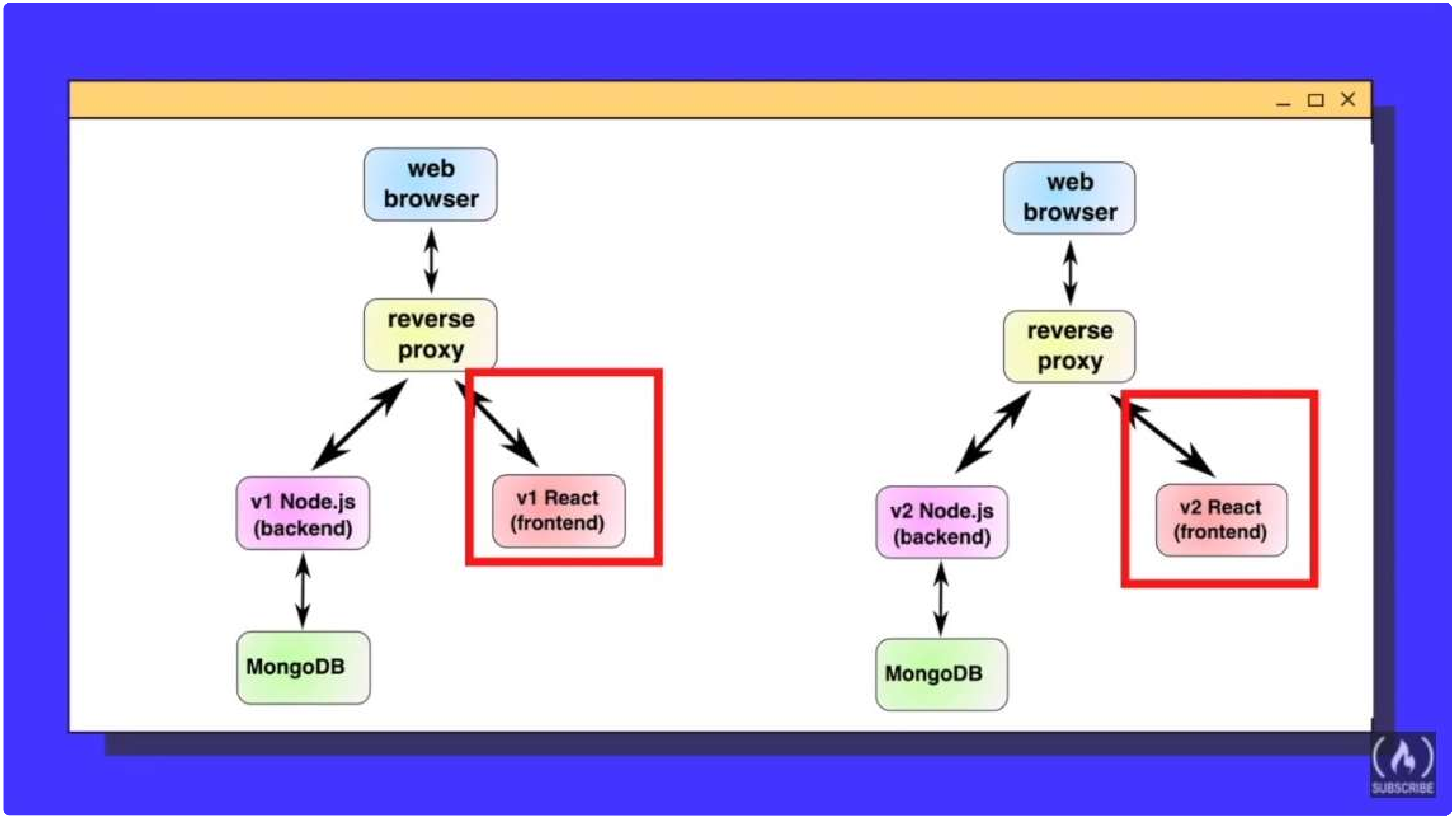
Autoscaling Definition:

Autoscaling automates horizontal scaling to ensure that the number of workers is proportional to the load on the system.

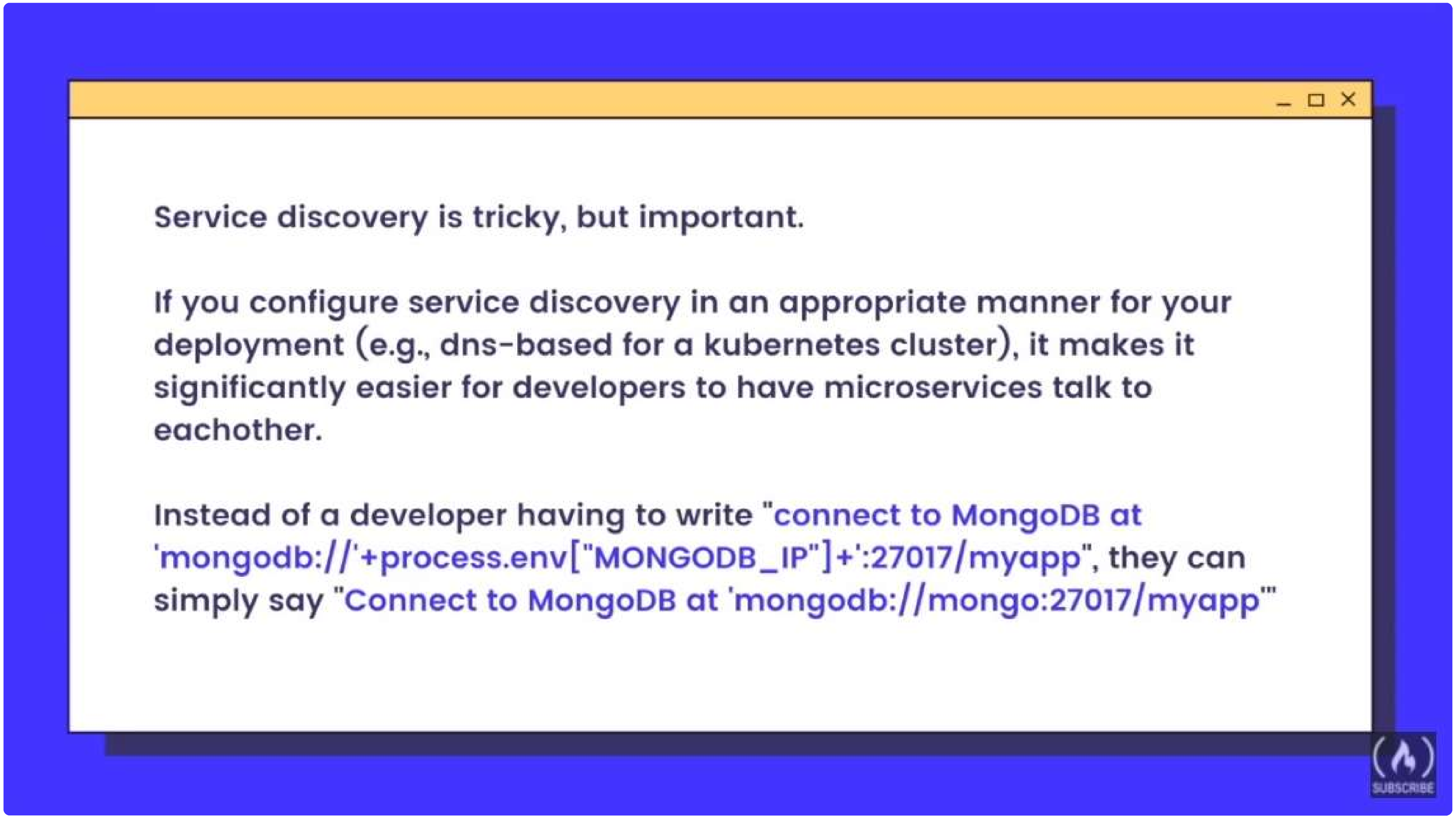


Serverless vs Autoscaling

Autoscaling is usually discussed on the timeline of ~1 hour chunks of work. If you took the concept of autoscaling and took it to its limit, you'd get serverless: define resources that are quickly started, and use them on the timeline of ~100ms.



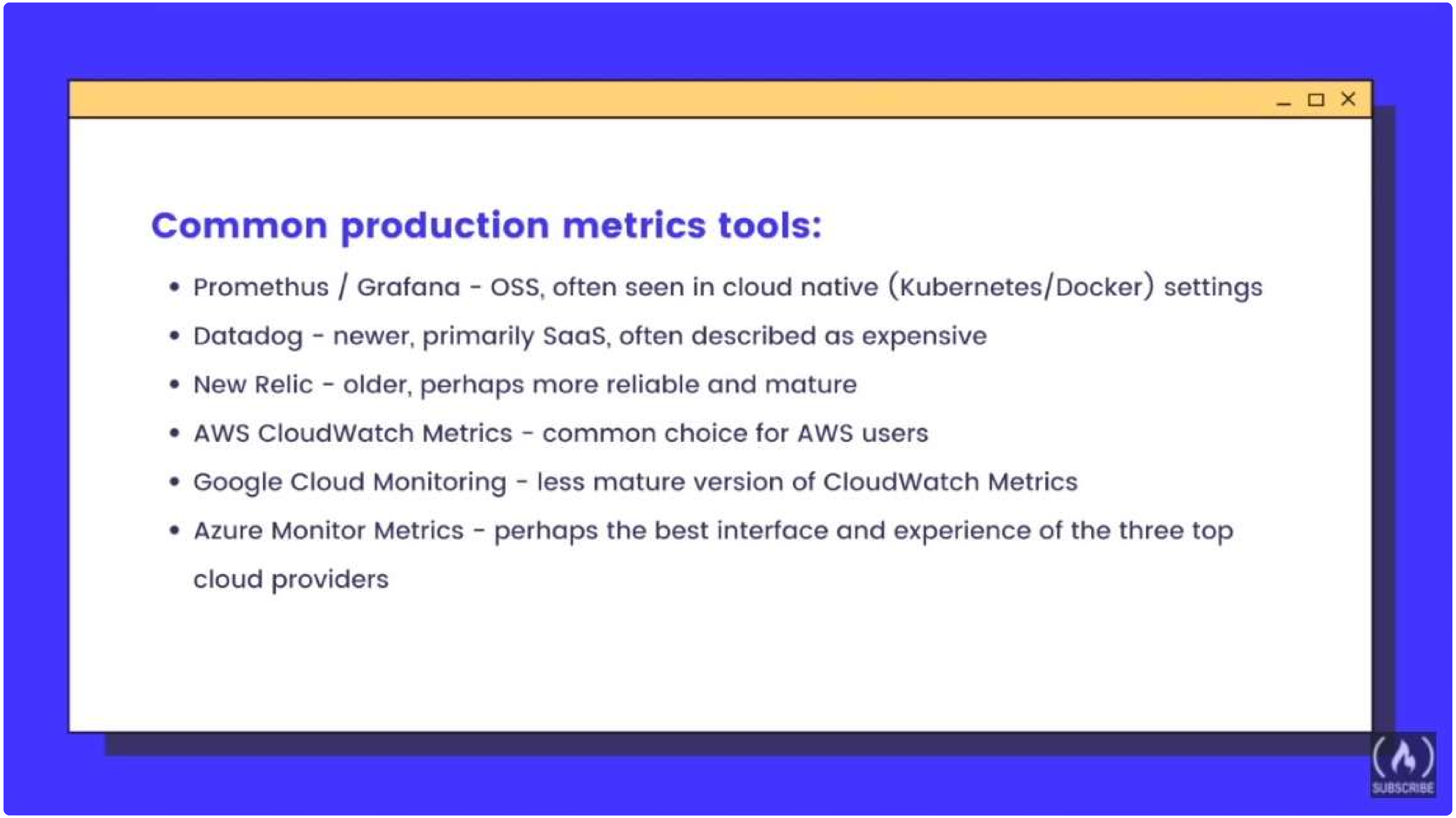
Reverse Proxy



Service discovery is tricky, but important.

If you configure service discovery in an appropriate manner for your deployment (e.g., dns-based for a kubernetes cluster), it makes it significantly easier for developers to have microservices talk to eachother.

Instead of a developer having to write "connect to MongoDB at 'mongodb://'+process.env["MONGODB_IP"]+':27017/myapp", they can simply say "Connect to MongoDB at 'mongodb://mongo:27017/myapp'"



Common production metrics tools:

- Promethus / Grafana - OSS, often seen in cloud native (Kubernetes/Docker) settings
- Datadog - newer, primarily SaaS, often described as expensive
- New Relic - older, perhaps more reliable and mature
- AWS CloudWatch Metrics - common choice for AWS users
- Google Cloud Monitoring - less mature version of CloudWatch Metrics
- Azure Monitor Metrics - perhaps the best interface and experience of the three top cloud providers