SOFTWARE ENGINEERING

CO3001

CONTINUOUS INTEGRATION AND DELIVERY (CI/CD)

Anh Nguyen-Duc
Tho Quan-Thanh

WEEK 9

Adapted from https://iansommerville.com/software-engineering-book/slides/

OUTLINE

- ✓ Challenges of modern code development
- ✓ Code integration
- ✓ Continuous integration
- ✓ Continuous delivery
- ✓ DevOps

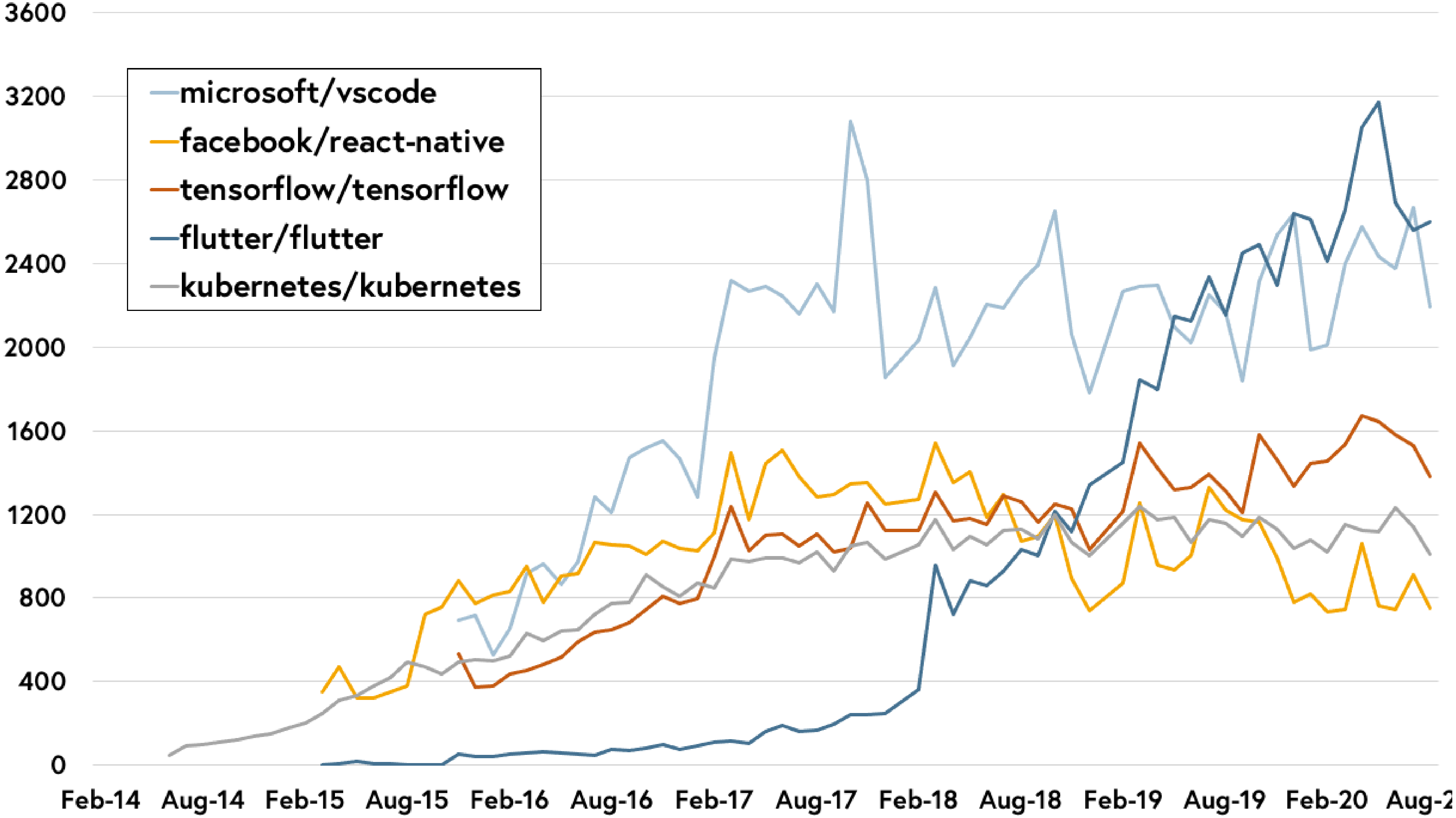
THE CHALLENGE: COMPLEXITY AND SIZE

- ✓ As the project grows, complexity grows:
  - Physical code size
  - Dependencies
  - Number of developers
  - Package versions
- ✓ Examples of well-known open source projects

THE CHALLENGE: COMPLEXITY AND SIZE

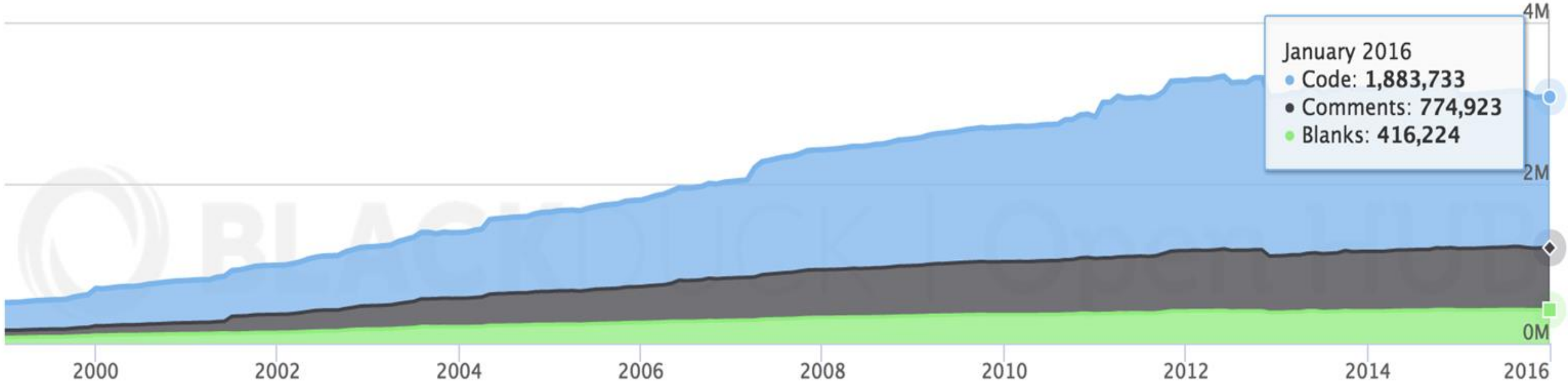
- ✓ Some notably large codebases include:
  - Google: monolithic, 1 billion files, 9 million source code files, 2 billion lines of source code, 35 million commits in total, 86 TB total size (January 2015)
  - Facebook: monolithic, 8 GB (repo 54 GB including history, 2014),[6] hundreds of thousands of files (2014)
  - Linux kernel: distributed, over 15 million lines of code (as of 2013 and kernel version 3.10)

Unique Monthly Contributors
Top 5 Projects (by Cumulative Contributions since 2011)



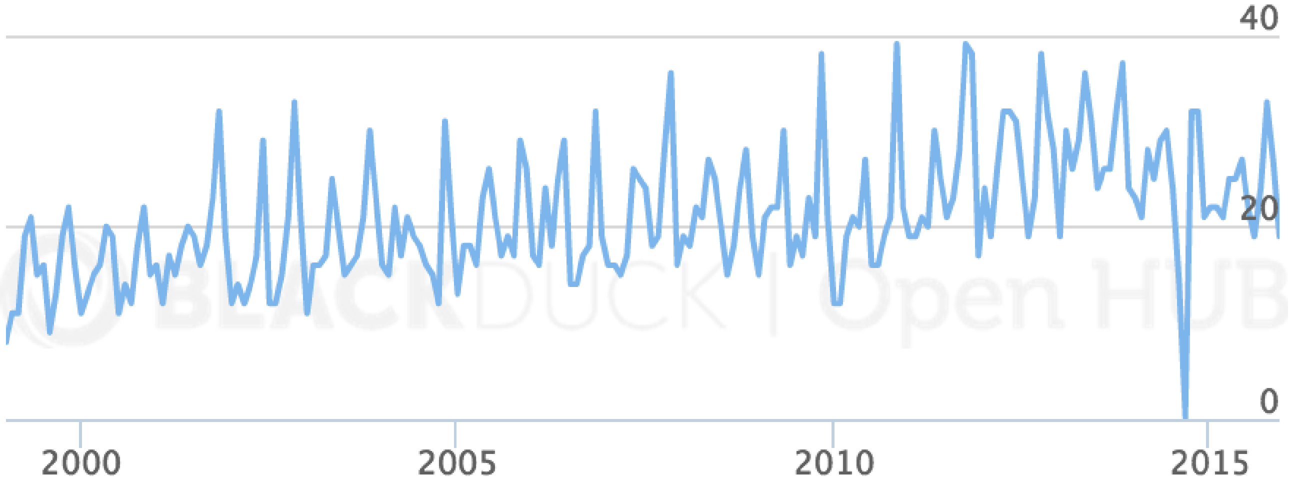
Example - Geant4

- A framework for the simulation of the passage of particles through matter.
  - Used in HEP, medical and space physics
- Just under 2 million lines of code
  - Mostly C++



LOC

Example - Geant4



# of contributors

- 537 person-years
    - Estimated cost: ~ €29 million
- 58,683 commits from 160 developers

THE CHALLENGE

- How do we handle increasing code-base sizes?
- How do we handle an increasing number of developers?
  - How can developers interact with each other?
- How do we build across multiple platforms?
- How do we build multiple versions?
- How can we make sure we don't break things!

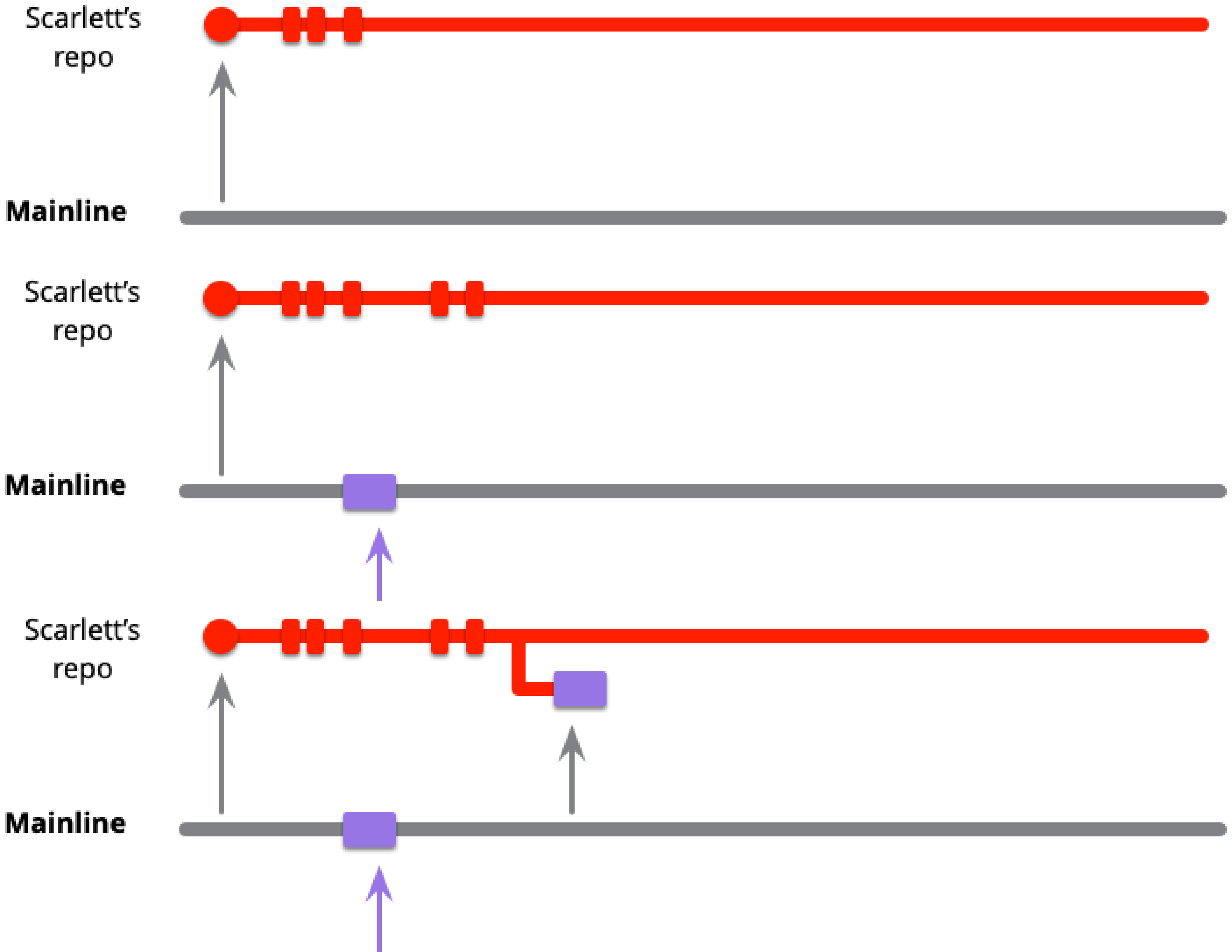
WHAT IS INTEGRATION?

- 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 needs to integrate their work into the main codebase.

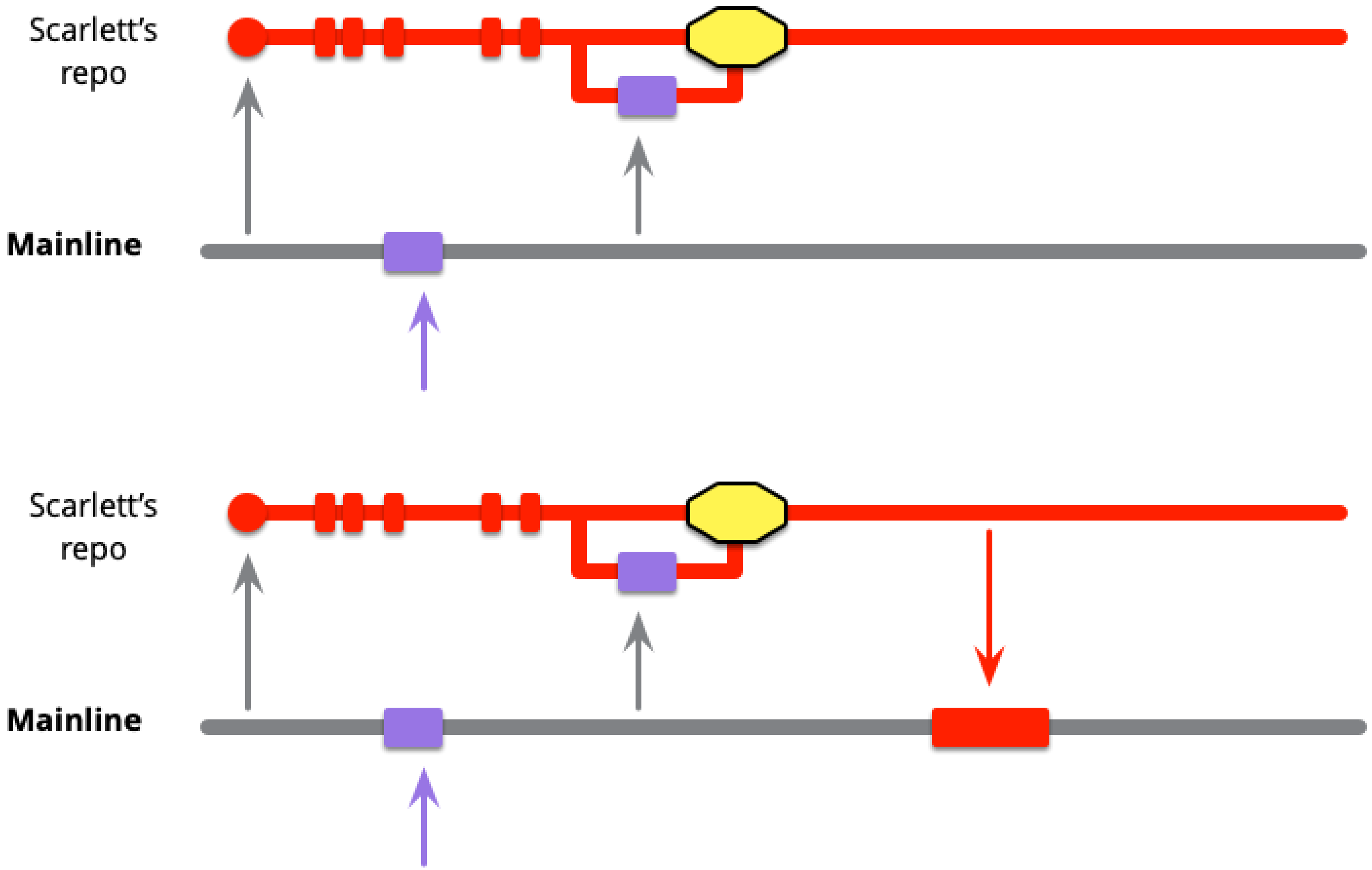
“I can't compile the program if you're in the middle of typing a variable name”

https://martinfowler.com/articles/branching-

- Mainline integration:Developers integrate their work by pulling from mainline, merging, and - if healthy - pushing back into mainline



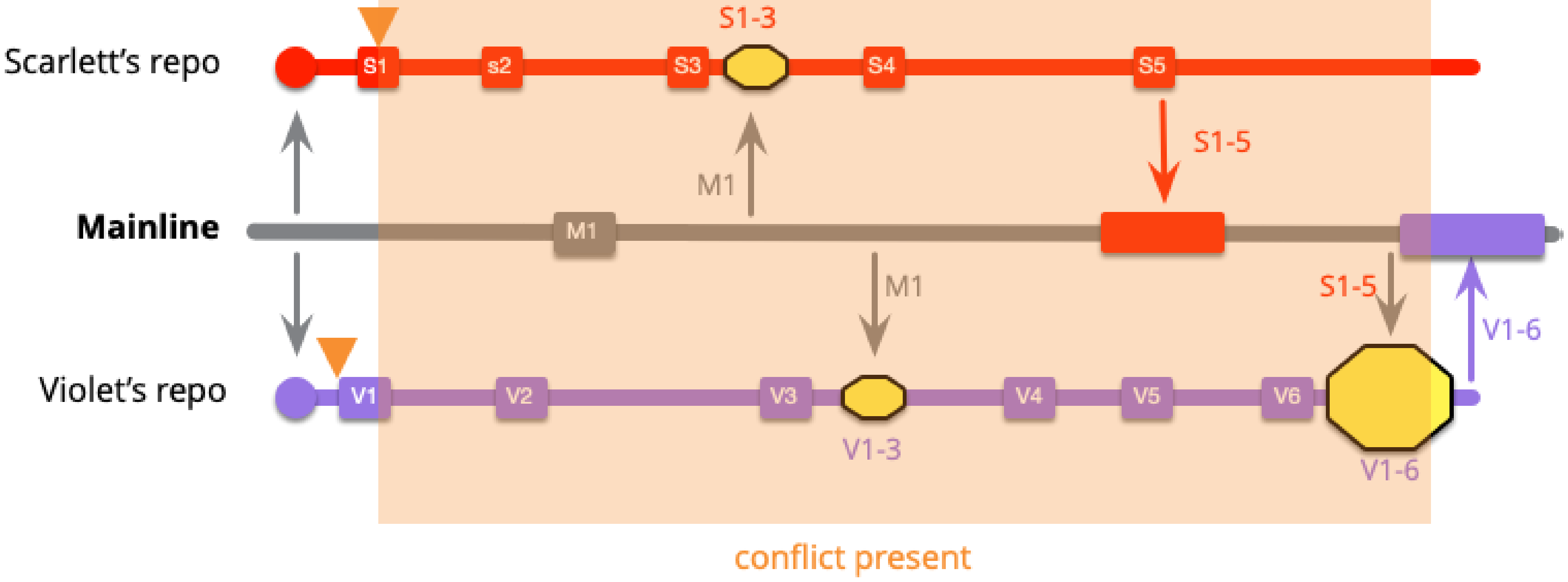
- Mainline integration:Developers integrate their work by pulling from mainline, merging, and - if healthy - pushing back into mainline



INTEGRATION FREQUENCY

✓ Elite development teams integrate notably more often than low performers

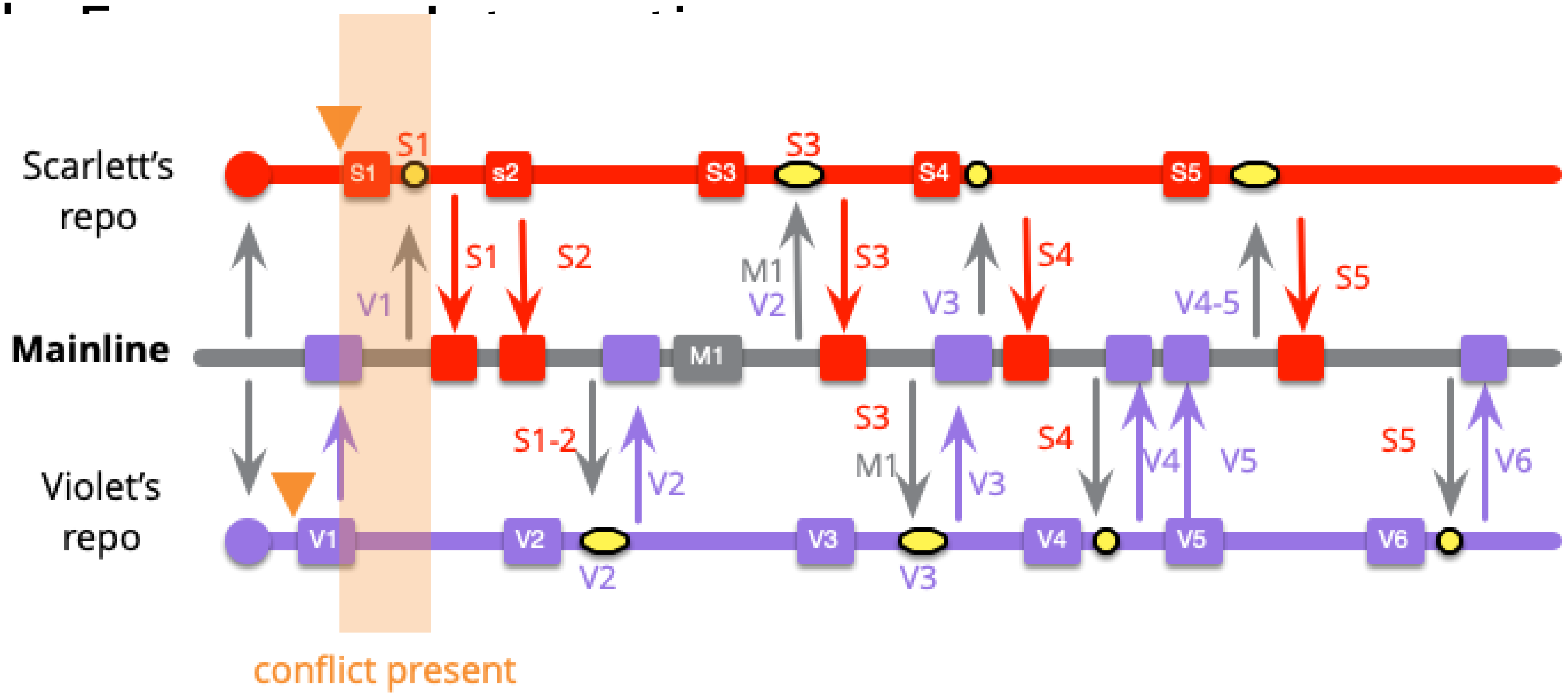
✓ Low-Frequency Integration



https://martinfowler.com/articles/branching-

INTEGRATION FREQUENCY

- ✓ Elite development teams integrate notably more often than low performers
- ✓ Hi



https://martinfowler.com/articles/branching-

What is continous integration?

"... a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily—leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible."

- Martin Fowler

What is continous integration?

- Continuous integration (CI) 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 anautomated build: compile the code andalso run automated tests?
- Question: Why are automated tests run?

How is continous integration?

- Use various existing tools to:
  - Combine changes often
  - Build often
  - Test often
  - Deploy often

Why is continous integration?

- Early/rapid feedback!
  - Do all components/projects compile?
  - Coding standards?
  - Are tests successful?
  - Performance requirements?
  - Problems archiving or deploying?
- Better project visibility
  - Possible to notice trends
  - What features are needed/being added

How is continous integration?

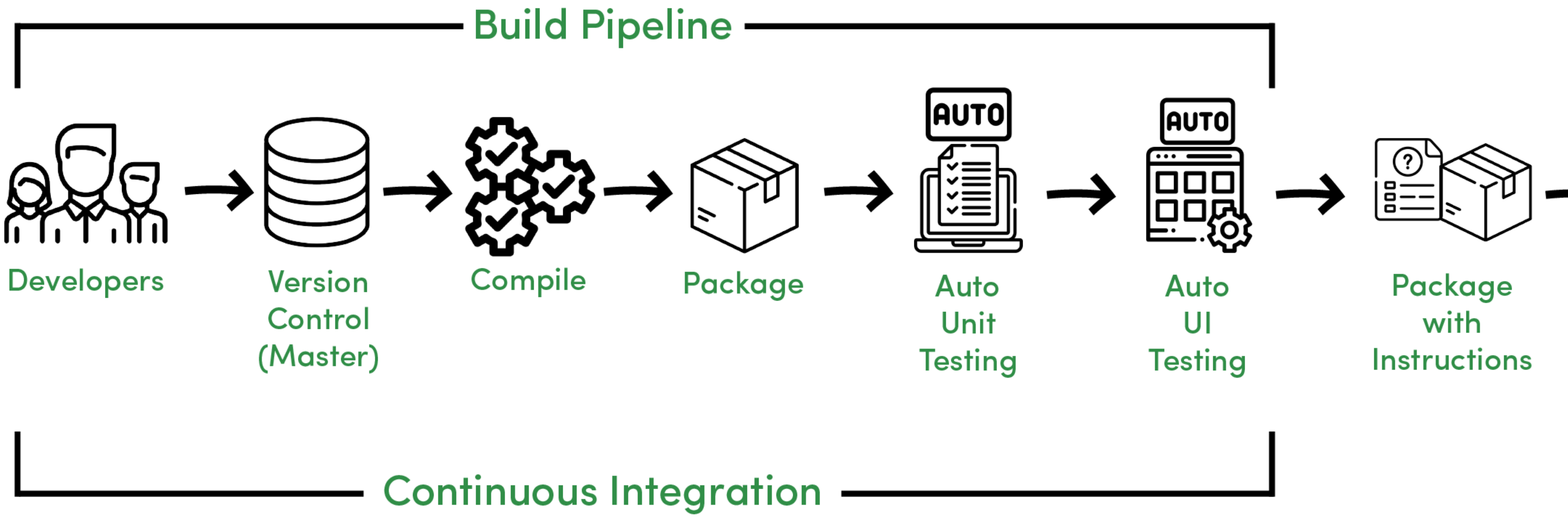
In order for CI to work, individual developers should:

- Commit frequently
  - Many small commits
- Run local build first (if possible)
  - Huge code repos may make this difficult
  - Only commit working code
- Fix broken builds immediately
- Write automated tests

Why is continous integration?

- Insures clean environments
- Manual tasks automated
- Speedup of working software turnover
- No large integration steps
- Much less likely to break something
- A full working/deployable version at ANY POINT IN TIME
- Complete documentation of who did what

CONTINUOUS INTEGRATION MODEL



- ✓ Version control software
- ✓ Dependency management
- ✓ Automated testing software
- ✓ Continuous integration framework
- ✓ Infrastructure management
- ✓ Build automation

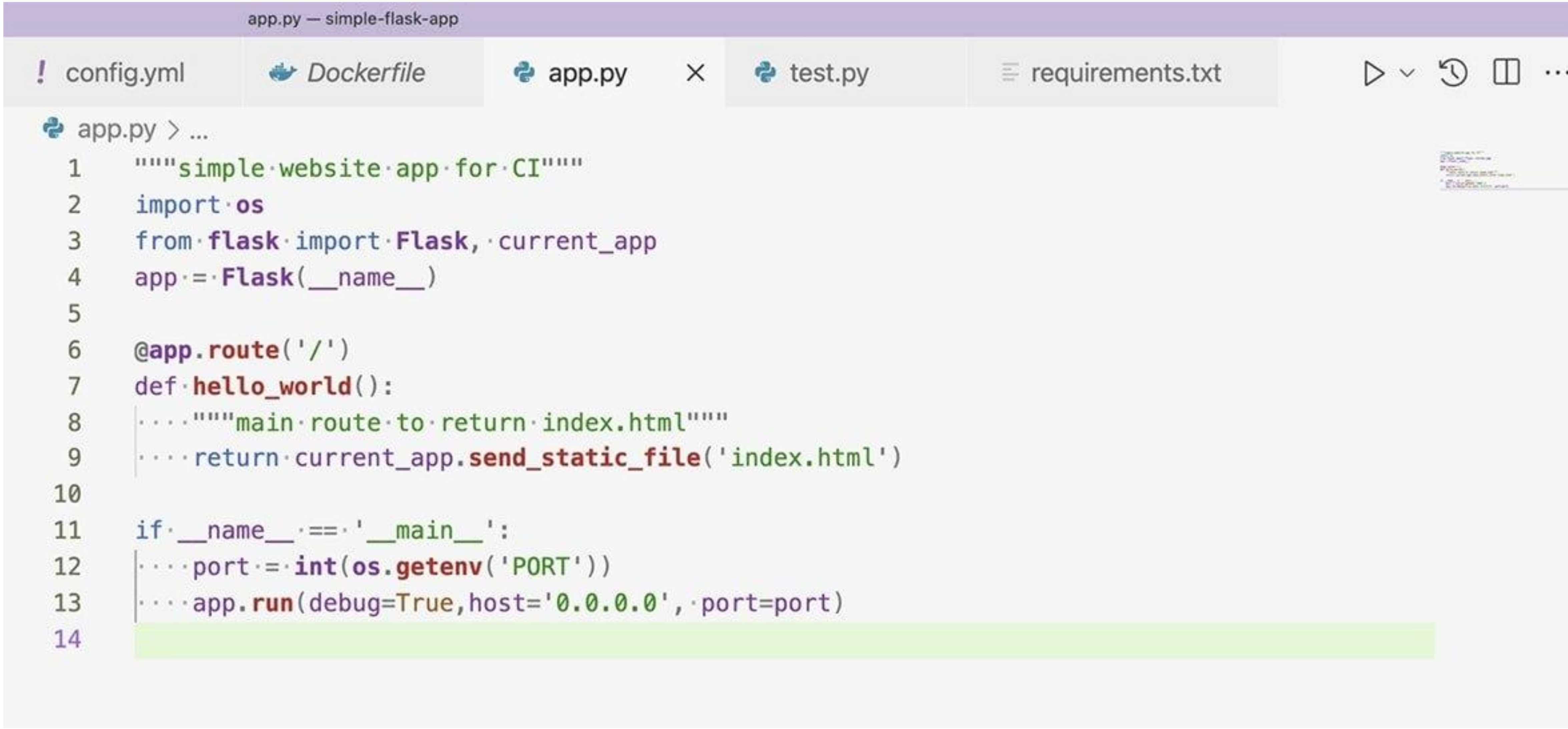
CONTINUOUS INTEGRATION TOOLS

- Code repositories
  - BitKeeper, Bzr, CVS, Darcs, Gerrit, Git, Mercurial, Monotone, P4, SVN ...
- Test frameworks
  - CppUnit, Valgrind, JUnit, Unittest, TestNg ...
- Continuous Integration
  - Bamboo, Buildbot, CruiseControl, Jenkins, Gitlab CI …

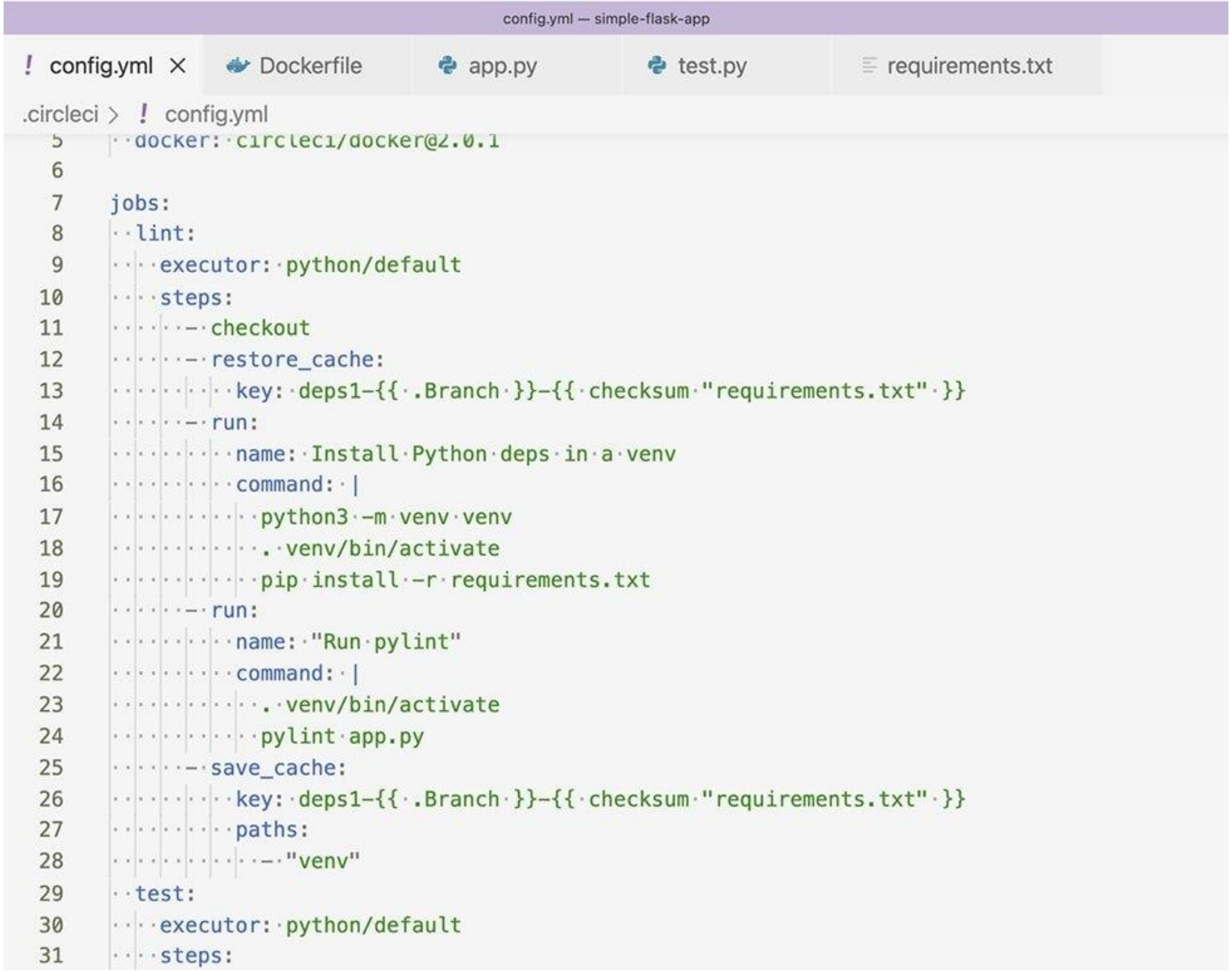
Wikipedia has a great comparison table:

SETTING UP A CI PIPELINE

A simple example of a Flask web appliation

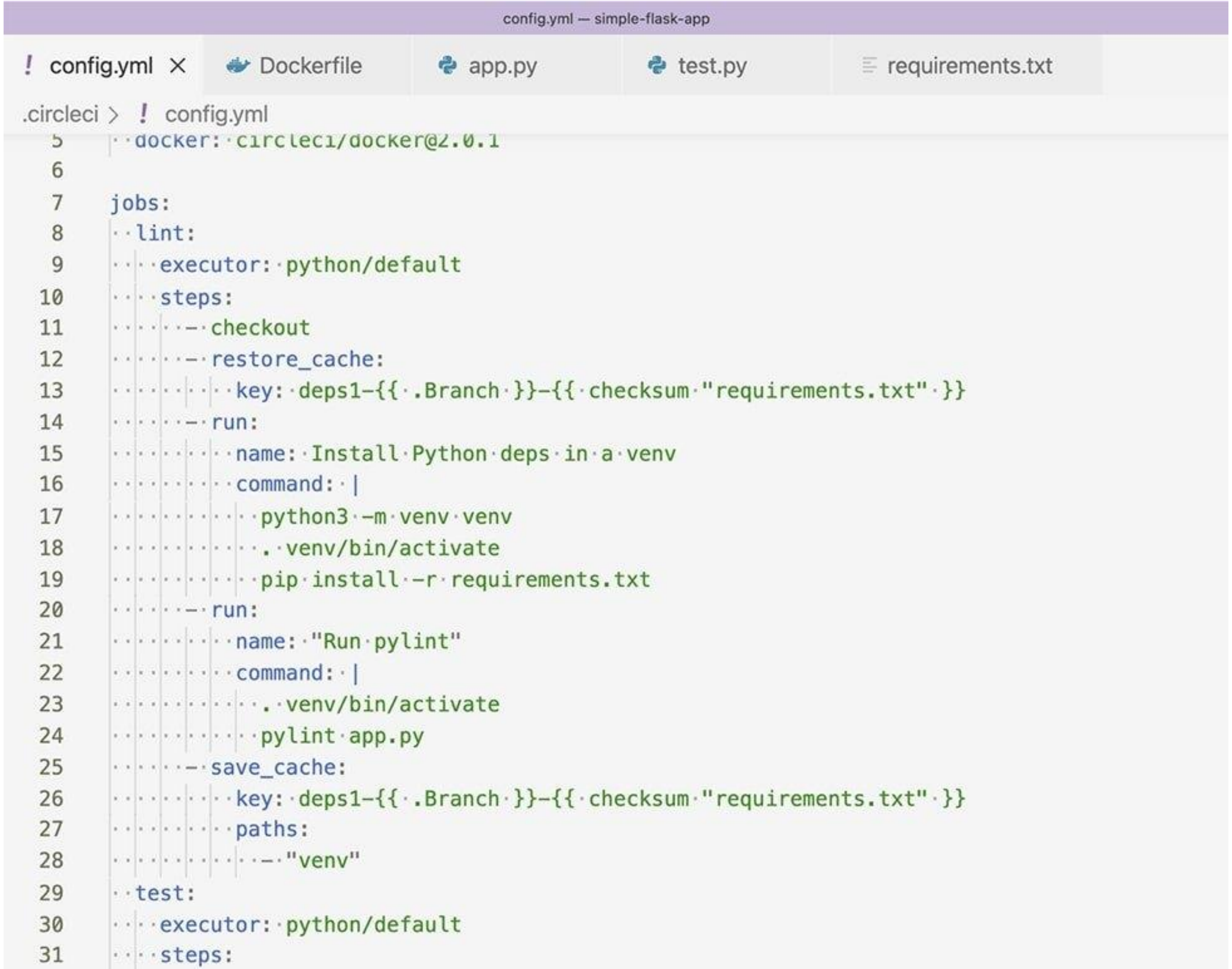


```python
"""simple website app for CI"""
import os
from flask import Flask, current_app
app = Flask(__name__)

@app.route('/')
def hello_world():
    """main route to return index.html"""
    return current_app.send_static_file('index.html')

if __name__ == '__main__':
    port = int(os.getenv('PORT'))
    app.run(debug=True,host='0.0.0.0', port=port)

```



```yaml
  docker: circleci/docker@2.0.1

jobs:
  lint:
    executor: python/default
    steps:
      - checkout
      - restore_cache:
          key: deps1-{{ .Branch }}-{{ checksum "requirements.txt" }}
      - run:
          name: Install Python deps in a venv
          command: |
            python3 -m venv venv
            . venv/bin/activate
            pip install -r requirements.txt
      - run:
          name: "Run pylint"
          command: |
            . venv/bin/activate
            pylint app.py
      - save_cache:
          key: deps1-{{ .Branch }}-{{ checksum "requirements.txt" }}
          paths:
            - "venv"
  test:
    executor: python/default
    steps:
```

- ✓ Our YAML file defines four different processes to run: lint, test, build and deploy.



```yaml
  docker: circleci/docker@2.0.1

jobs:
  lint:
    executor: python/default
    steps:
      - checkout
      - restore_cache:
          key: deps1-{{ .Branch }}-{{ checksum "requirements.txt" }}
      - run:
          name: Install Python deps in a venv
          command: |
            python3 -m venv venv
            . venv/bin/activate
            pip install -r requirements.txt
      - run:
          name: "Run pylint"
          command: |
            . venv/bin/activate
            pylint app.py
      - save_cache:
          key: deps1-{{ .Branch }}-{{ checksum "requirements.txt" }}
          paths:
            - "venv"
  test:
    executor: python/default
    steps:
```

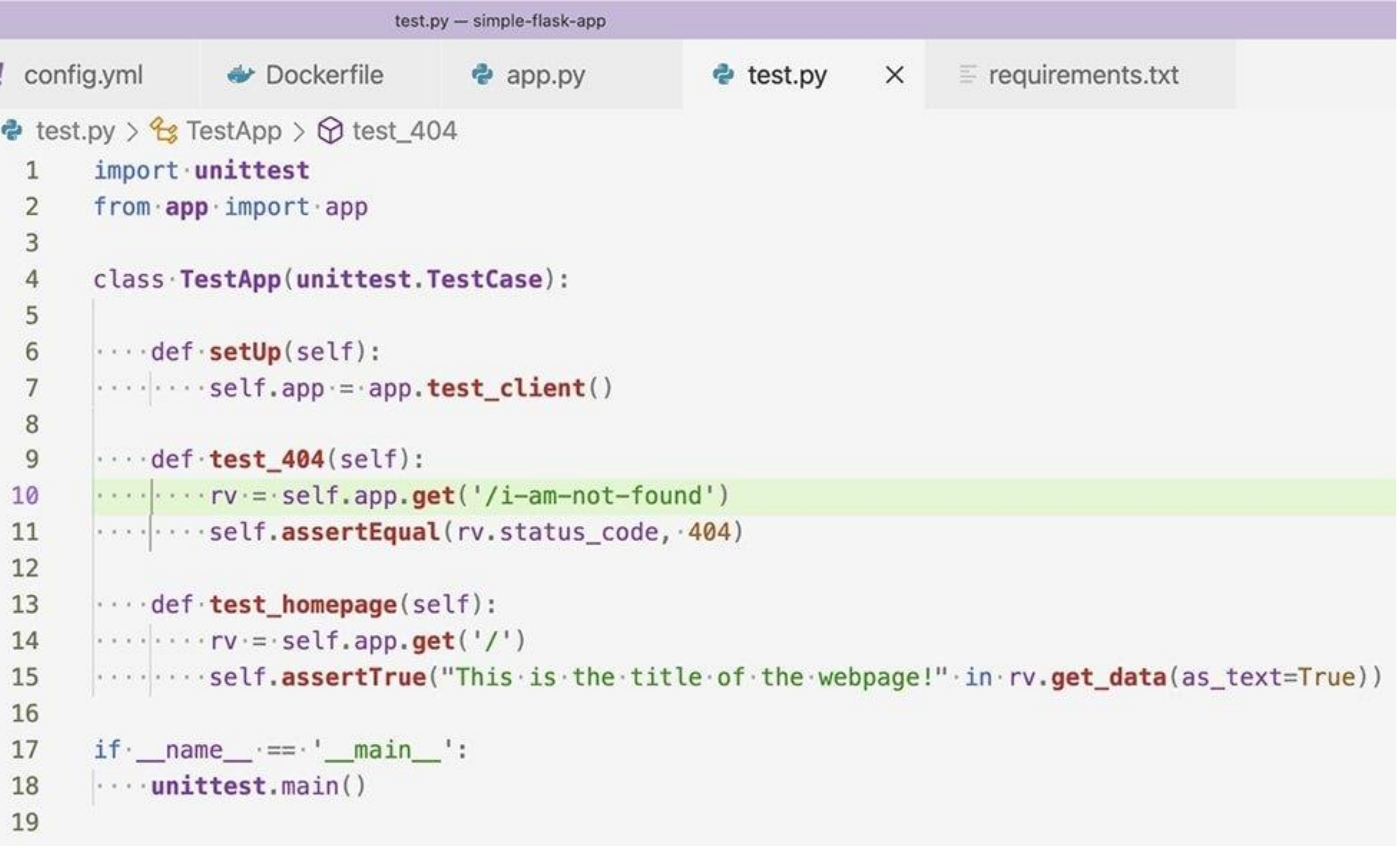
Pylint

Star your Python code!

- ✓ The lint stage checks for possible errors and formatting issues without running the code. The linting program used in this case is a popular tool called Pylint.

Pylint

Star your Python code!



```python
import unittest
from app import app

class TestApp(unittest.TestCase):

    def setUp(self):
        self.app = app.test_client()

    def test_404(self):
        rv = self.app.get('/i-am-not-found')
        self.assertEqual(rv.status_code, 404)

    def test_homepage(self):
        rv = self.app.get('/')
        self.assertTrue("This is the title of the webpage!" in rv.get_data(as_text=True))

if __name__ == '__main__':
    unittest.main()

```

- ✓ The next step in our CI/CD pipeline tutorial is testing. Our tests in this project are run with the unit test framework

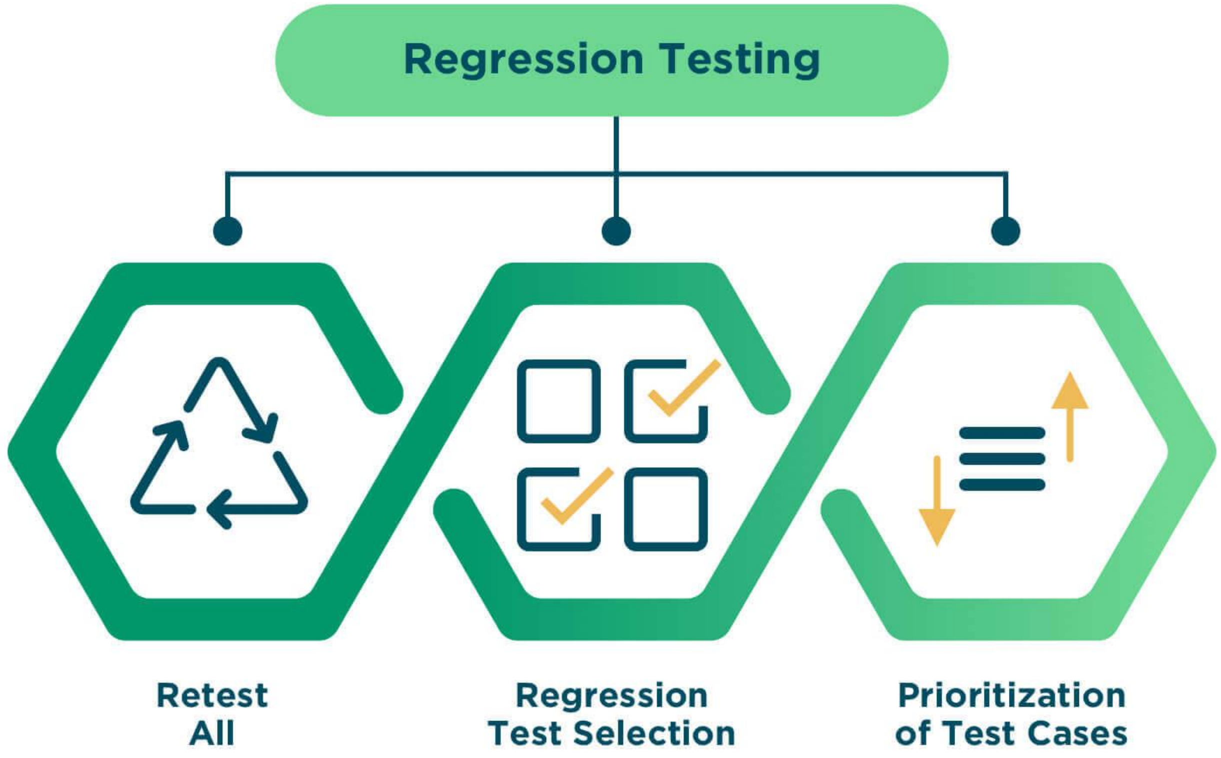
AUTOMATED TESTING

- ✓ Automated testing is the application of software tools to automate a human-driven manual process of reviewing and validating a software product
- ✓ Different levels:
  - Unit test
  - Integration test: mocking these 3rd party dependencies and asserting the code interfacing with them behaves as expected
  - Performance test: i.e. speed and responsiveness

REGRESSION TESTING

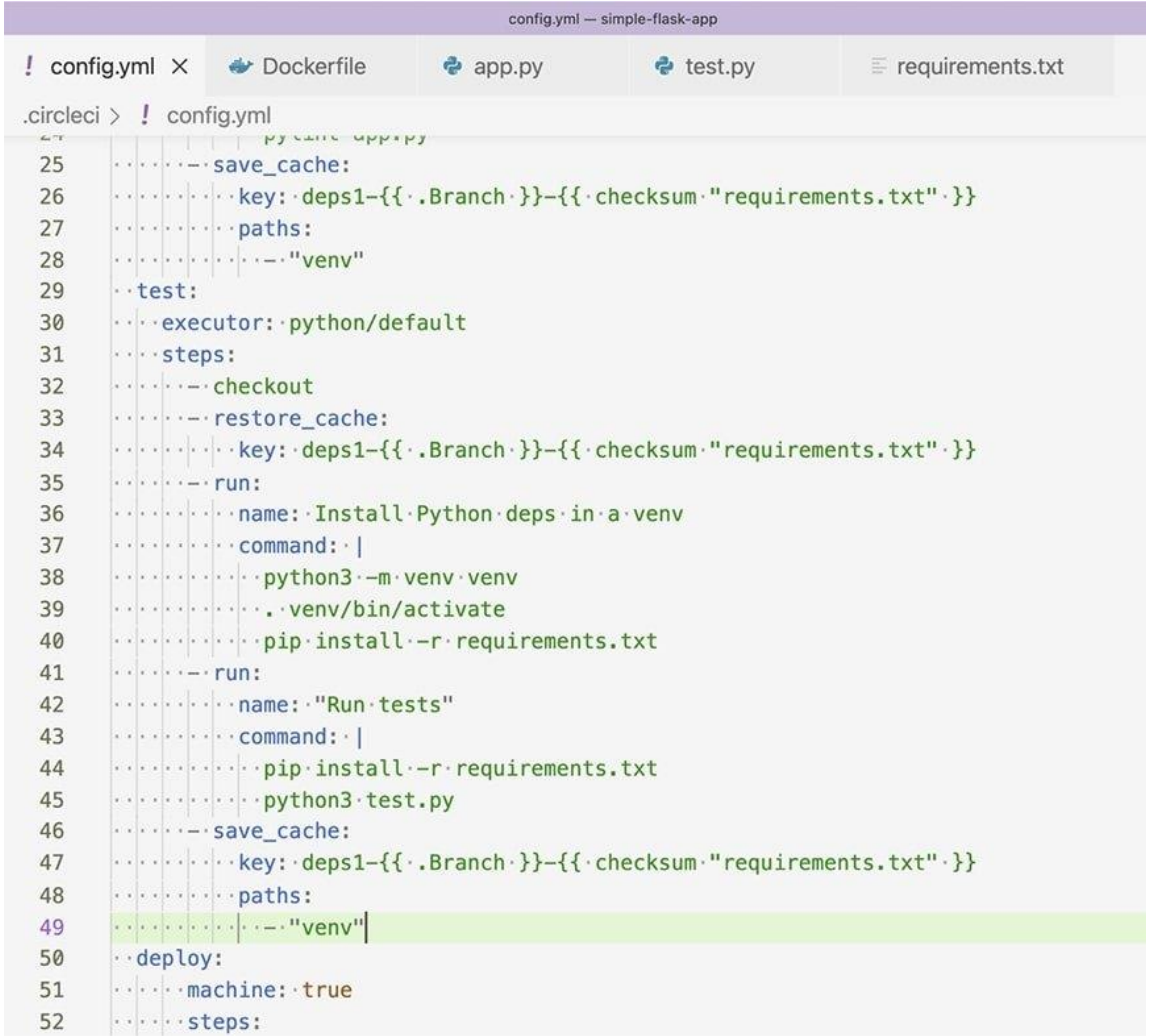
✓ re-running functional and non-functional tests to ensure that previously developed and tested software still performs after a change

✓ 3 types



Pylint

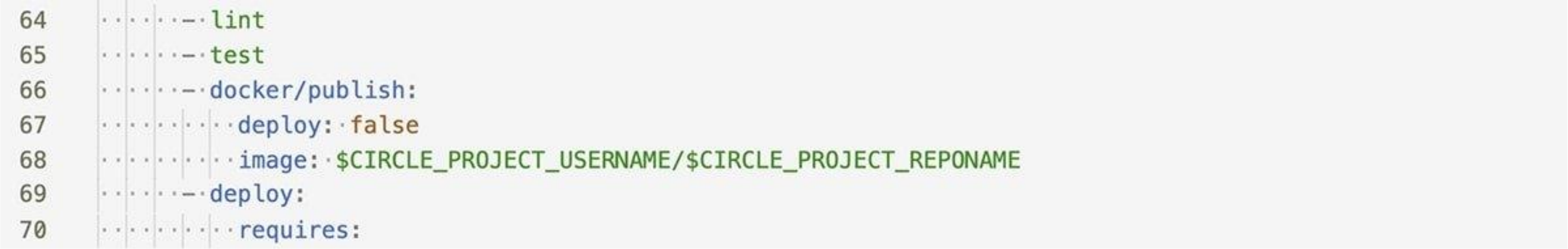
Star your Python code!



```yaml
            pylint app.py
      - save_cache:
          key: deps1-{{ .Branch }}-{{ checksum "requirements.txt" }}
          paths:
            - "venv"
  test:
    executor: python/default
    steps:
      - checkout
      - restore_cache:
          key: deps1-{{ .Branch }}-{{ checksum "requirements.txt" }}
      - run:
          name: Install Python deps in a venv
          command: |
            python3 -m venv venv
            . venv/bin/activate
            pip install -r requirements.txt
      - run:
          name: "Run tests"
          command: |
            pip install -r requirements.txt
            python3 test.py
      - save_cache:
          key: deps1-{{ .Branch }}-{{ checksum "requirements.txt" }}
          paths:
            - "venv"
  deploy:
      machine: true
      steps:
```

- ✓ The next step in our CI/CD pipeline tutorial is testing. Our tests in this project are run with the unit test framework
- ✓ Running tests on every commit is crucial to a project's success

BUILD STEP:

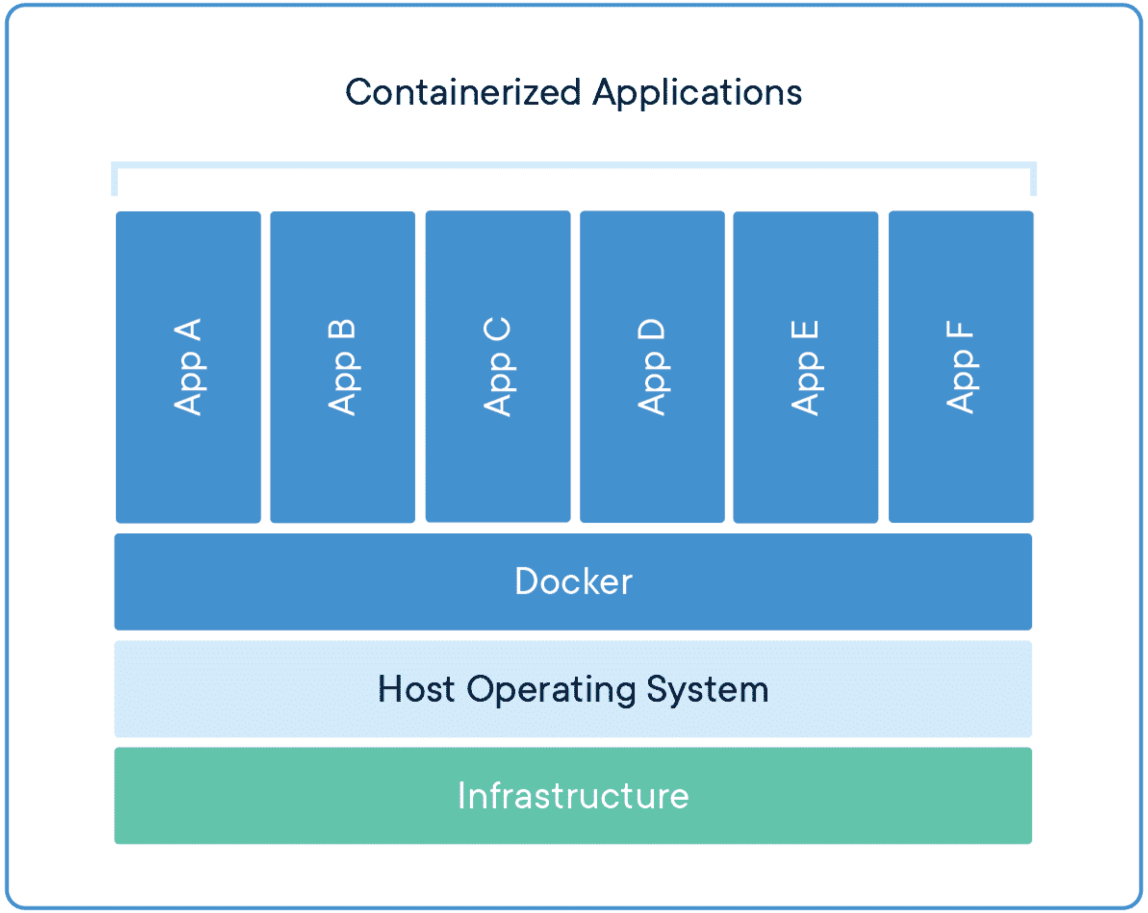


```
64        - lint
65        - test
66        - docker/publish:
67            deploy: false
68            image: $CIRCLE_PROJECT_USERNAME/$CIRCLE_PROJECT_REPONAME
69        - deploy:
70            requires:
```

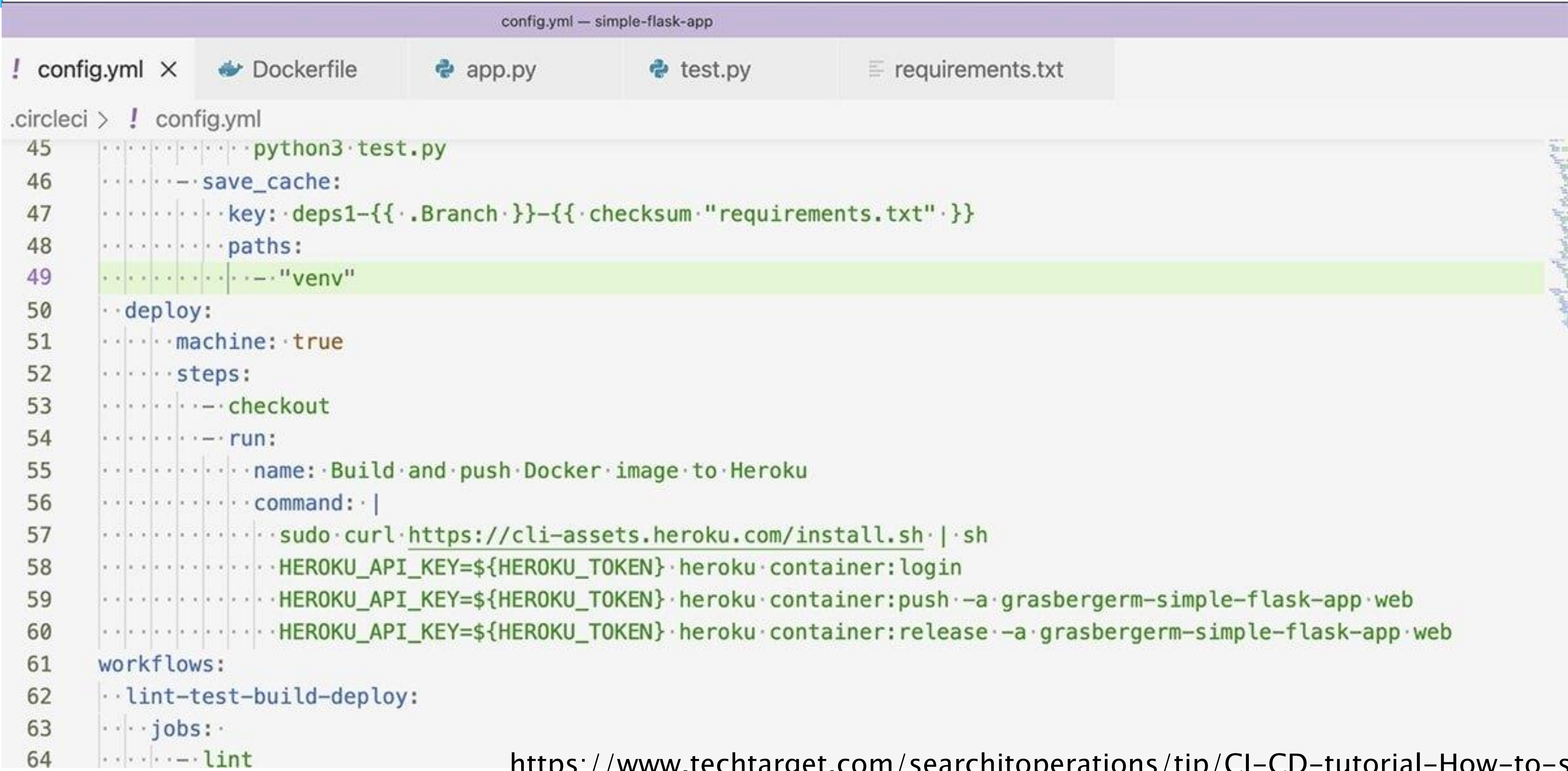
- ✓ Lines 66-68 reference the Docker orb and define how the Docker job will run. Set the `deploy` attribute to `false` to instruct the Docker/publish job to build the image without pushing it to a repository. By default, the Docker/publish job finds the Dockerfile by name and builds it. It will also fail the job if the Docker build fails.
- ✓

DOCKER

- an open platform for developing, shipping, and running applications
- separate your applications from your infrastructure
- significantly reduce the delay between writing code and running it in production
- container



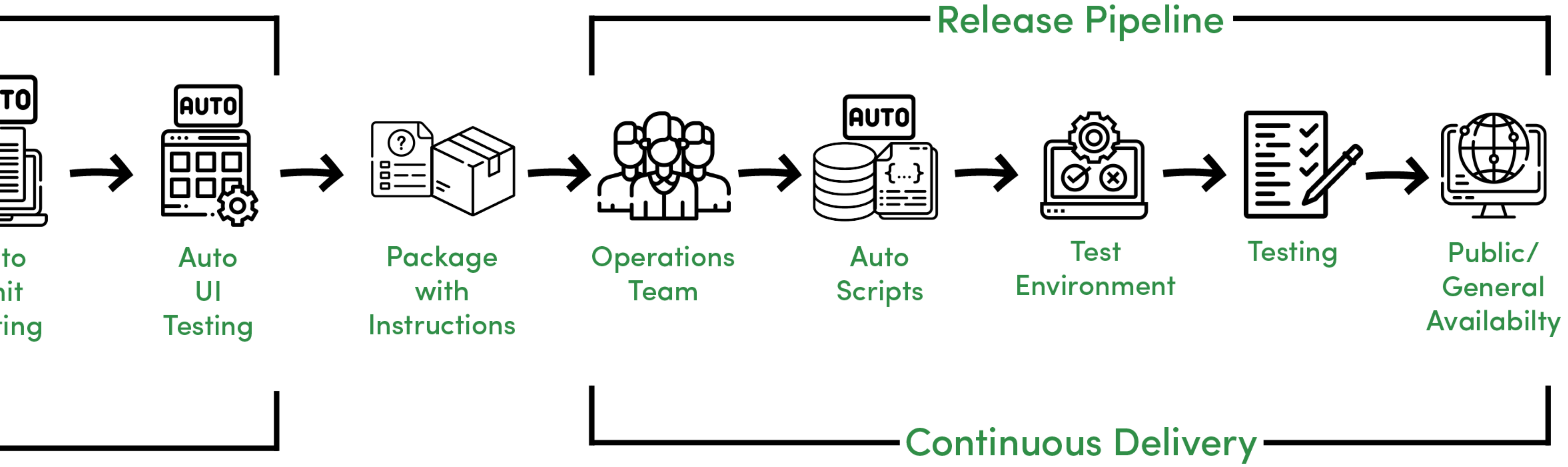
DEPLOY STEP:



```yaml
            python3 test.py
      - save_cache:
          key: deps1-{{ .Branch }}-{{ checksum "requirements.txt" }}
          paths:
            - "venv"
  deploy:
      machine: true
      steps:
        - checkout
        - run:
            name: Build and push Docker image to Heroku
            command: |
              sudo curl https://cli-assets.heroku.com/install.sh | sh
              HEROKU_API_KEY=${HEROKU_TOKEN} heroku container:login
              HEROKU_API_KEY=${HEROKU_TOKEN} heroku container:push -a grasbergerm-simple-flask-app web
              HEROKU_API_KEY=${HEROKU_TOKEN} heroku container:release -a grasbergerm-simple-flask-app web
workflows:
  lint-test-build-deploy:
    jobs:
      - lint
```

https://www.techtarget.com/searchitoperations/tip/CI-CD-tutorial-How-to-s

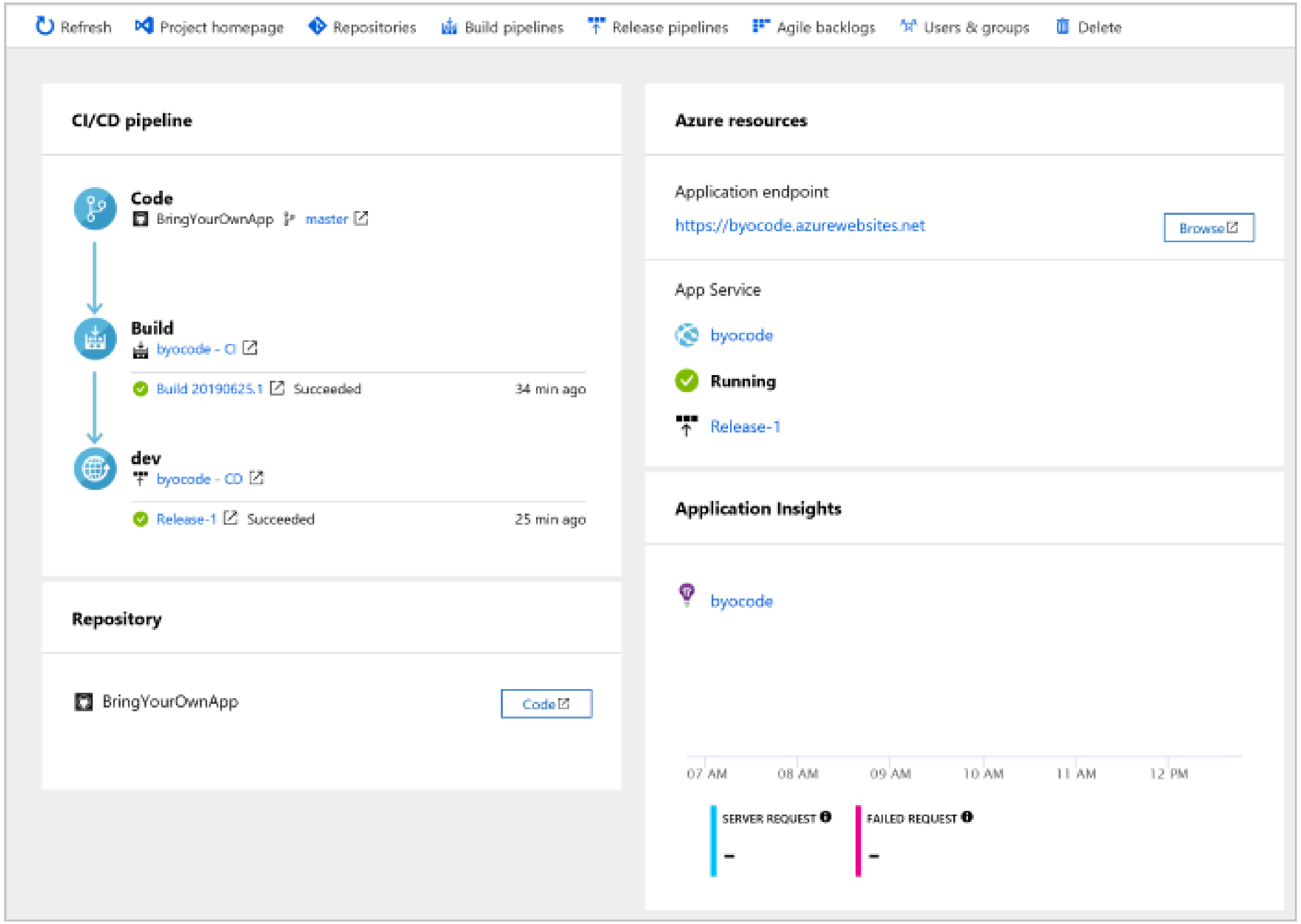
CONTINUOUS DELIVERY



CONTINUOUS DELIVERY

- ✓ teams produce software in short cycles, ensuring that the software can be reliably released at any time and, when releasing the software, without doing so manually.
- ✓ Continuous delivery is an extension of continuous integration since it automatically deploys all code changes to a testing and/or production environment after the build stage.

OTHER SOLUTIONS FOR CI/CD ....



CI/CD pipeline

Code
BringYourOwnApp master

Build
byocode - CI
Build 20190625.1 Succeeded 34 min ago

dev
byocode - CD
Release-1 Succeeded 25 min ago

Repository

BringYourOwnApp Code

Azure resources

Application endpoint
https://byocode.azurewebsites.net Browse

App Service
byocode
Running
Release-1

Application Insights

byocode

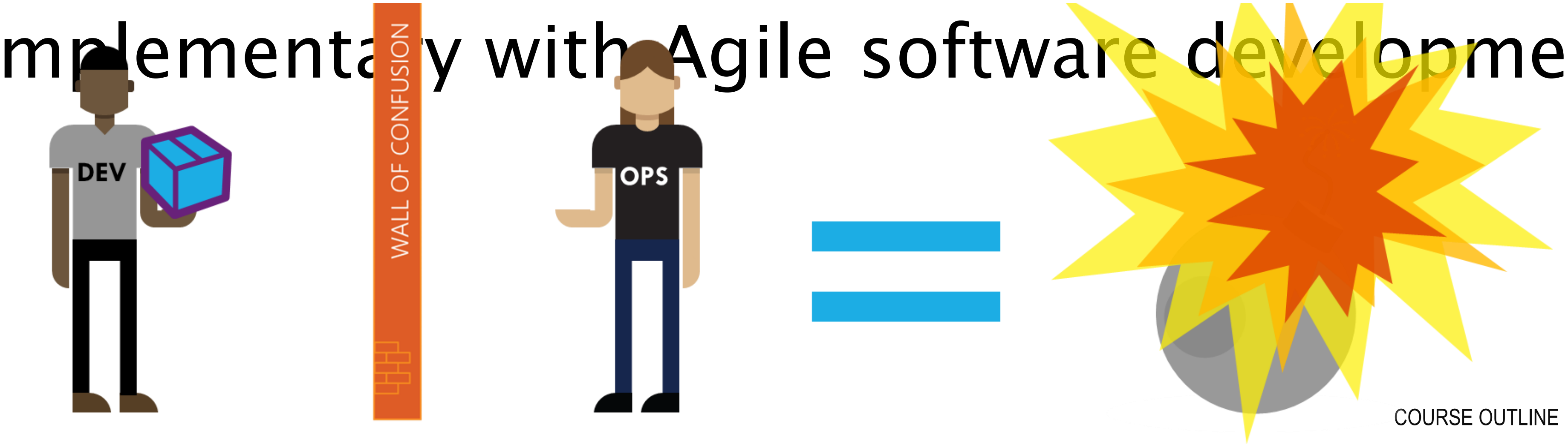
COMMON PITFALL OF CI/CD

- The repetition frequency of the process.
- The dependencies involved in the process & delay produced by them.
- Length of the process.
- The urgency in process automation.
- If the process is prone to errors if not automated.

✓

DEVOPS

- ✓ a set of practices that combines software development (Dev) and IT operations (Ops)
- ✓ Breaking the Silos: Dev and Ops
- ✓ aims to shorten the systems development life cycle and provide continuous delivery with high software quality
- ✓ complementary with Agile software development

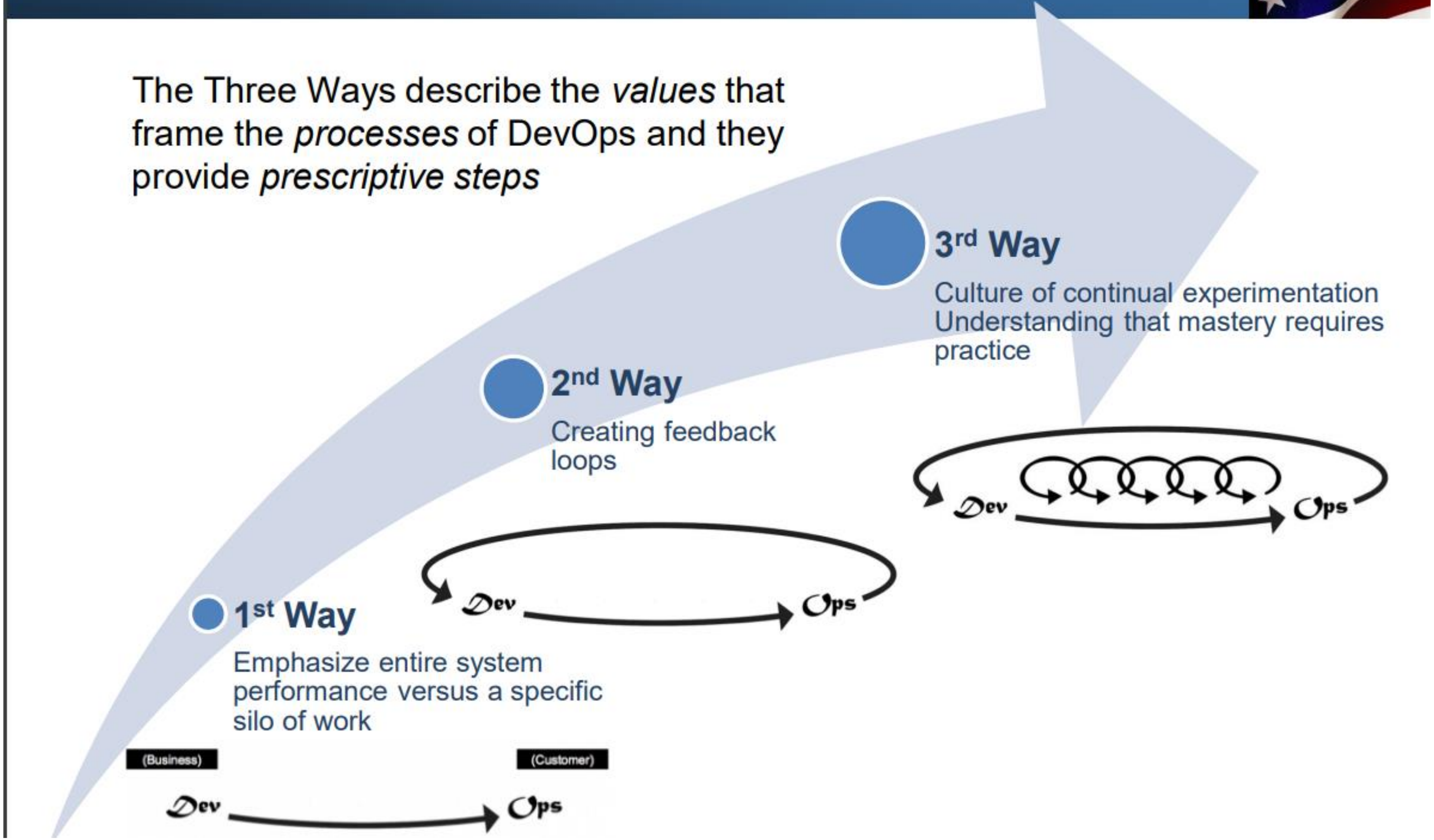


DEVS AND OPS WORKING TOGETHER

- ✓ Create feedback loops between inventors and mechanics
- ✓ Expose real-time metrics from ops enabling dev to learn from the system running under real world conditions
- ✓ Expose real-time metrics from dev enabling ops to anticipate production needs and provide early input
- ✓ Cross-functional teams collaborate to deliver whole working systems including all infrastructure, software code, and configurations

The Three Ways of DevOps by Gene Kim

The Three Ways describe the *values* that frame the *processes* of DevOps and they provide *prescriptive steps*



1st Way

Emphasize entire system performance versus a specific silo of work

2nd Way

Creating feedback loops

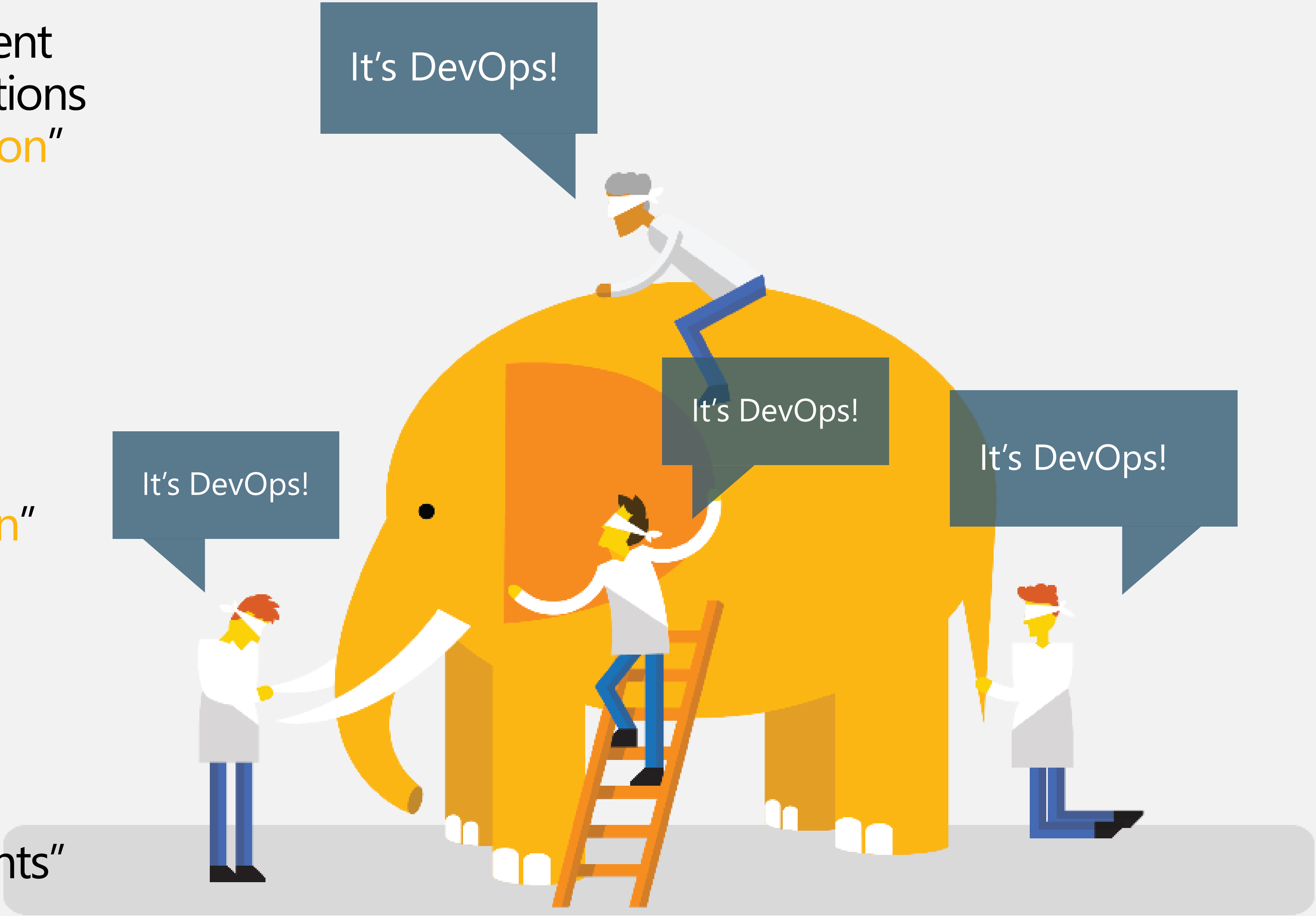
3rd Way

Culture of continual experimentation Understanding that mastery requires practice

“**DevOps** is development and operations collaboration”

“**DevOps** is using automation”

“**DevOps** is small deployments”

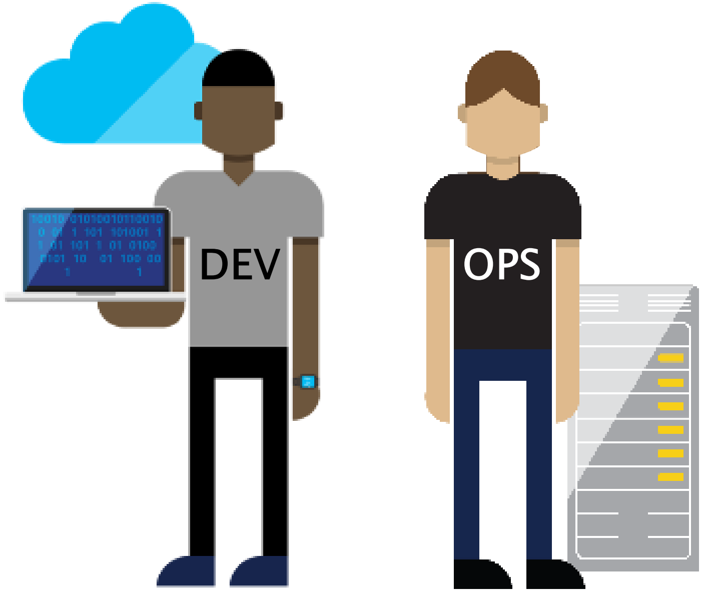


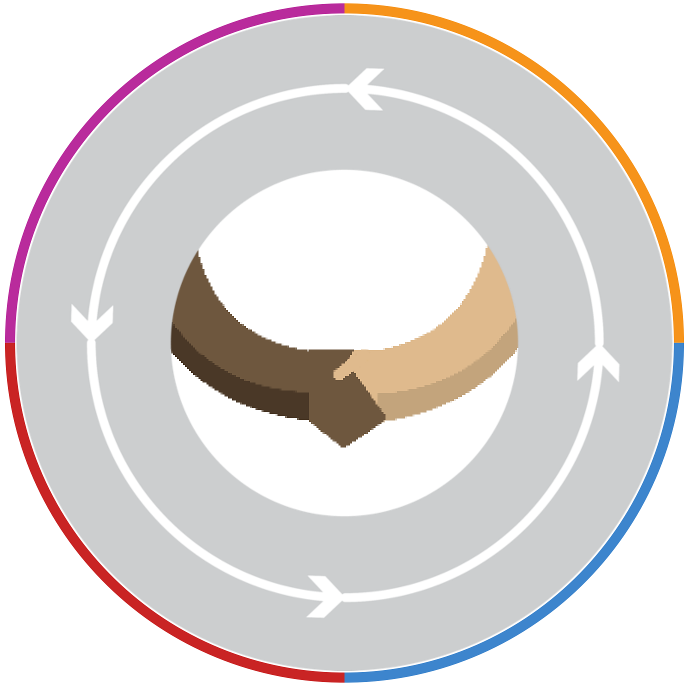
“**DevOps** is treating your infrastructure as code”

“**DevOps** is feature switches”

“Kanban for Ops?”

DEVOPS: THE THREE STAGE CONVERSATION





1 People

2 Process

3 Products

LIST OF DEVOPS PRACTICES

- Infrastructure as Code (IaC)
- Continuous Integration
- Automated Testing
- Continuous Deployment
- Release Management
- App Performance Monitoring
- Load Testing & Auto-Scale
- Availability Monitoring
- Change/Configuration Management
- Feature Flags
- Automated Environment De-Provisioning
- Self Service Environments
- Automated Recovery (Rollback & Roll-Forward)
- Hypothesis Driven Development
    - Testing in Production
    - Fault Injection
    - Usage Monitoring/User Telemetry

http://www.itproguy.com/devops-practices/

GitHub JetBRAINS Develop with pleasure! LeanKit AppVeyor slack Octopus Deploy

azuqua myget SOASTA Jenkins HU-BOT Atlassian HipChat

MODERN Requirements By eDevTECH Tasktop Aha! Trello dynatrace zendesk

Visual Studio Partners and Extensions

65
Visual Studio Code Extensions

5,910
Visual Studio Gallery Extensions

90
Visual Studio Sim-Ship Partners

48
VS Team Services Extensions

IncrediBuild BEYOND ACCELERATION CloudBees zapier uservoice flowdock bamboo solutions

macincloud

VS Anywhere Xamarin TFS TIMETRACKER jamo solutions perfecto mobile

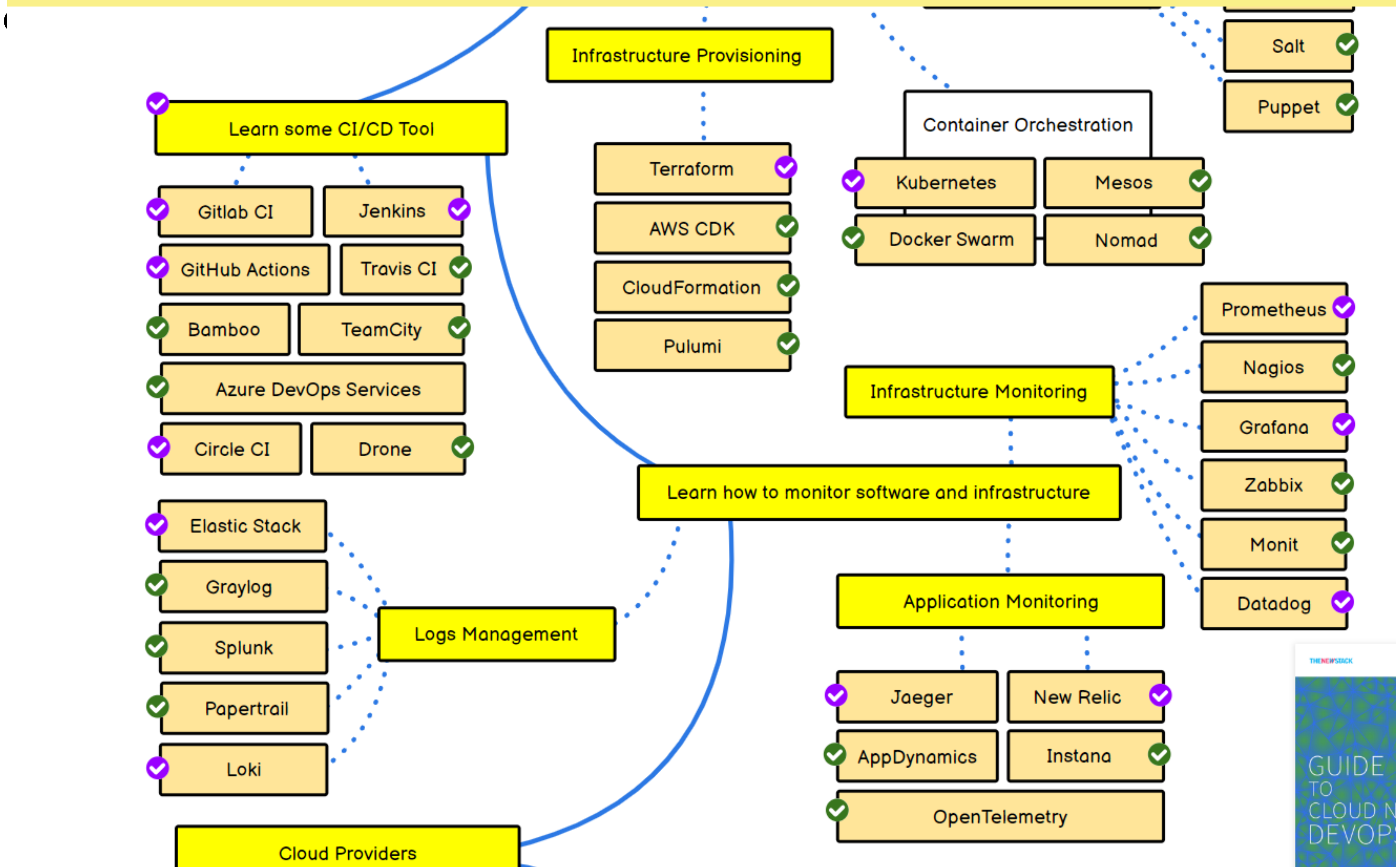
redgate ingeniously simple PreEmptive Solutions CHEF submain EKOBIT OpsHub Enabling Agile Integration

Campfire sauce LABS

BK TP.HCM

A BETTER VIEW

Step by step guide for DevOps, SRE or any other Operations Rol



Infrastructure Provisioning

Terraform

AWS CDK

CloudFormation

Pulumi

Container Orchestration

Kubernetes

Mesos

Docker Swarm

Nomad

Salt

Puppet

Learn some CI/CD Tool

Gitlab CI

Jenkins

GitHub Actions

Travis CI

Bamboo

TeamCity

Azure DevOps Services

Circle CI

Drone

Infrastructure Monitoring

Prometheus

Nagios

Grafana

Zabbix

Monit

Datadog

Learn how to monitor software and infrastructure

Elastic Stack

Graylog

Splunk

Papertrail

Loki

Logs Management

Application Monitoring

Jaeger

New Relic

AppDynamics

Instana

OpenTelemetry

Cloud Providers

Q&A