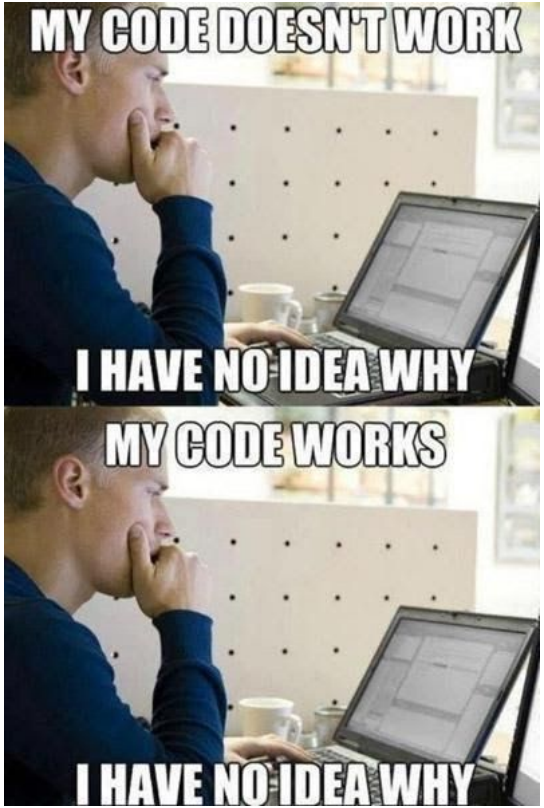
Testing

...

Testing prevents this:



Types of Testing

- Unit Test
- Integration Tests
- Load Tests
- Canarying
- Security Tests
- A/B
- SxS
- ...Others!

Unit tests

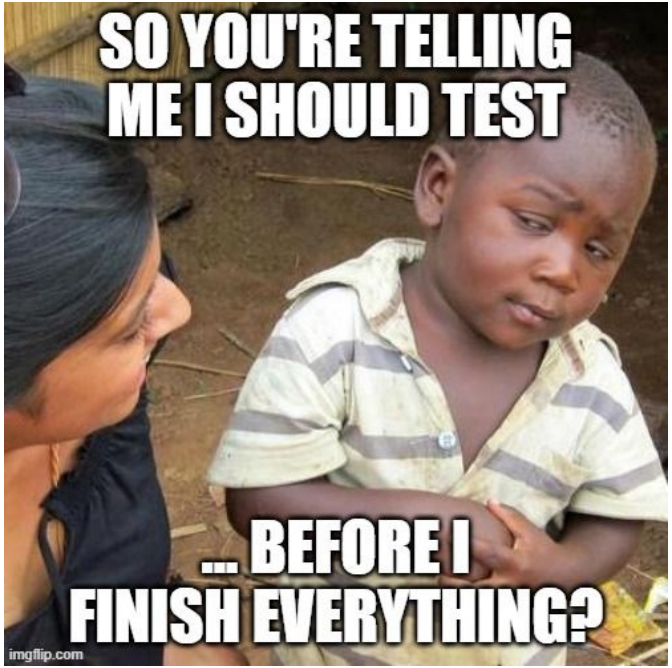
- A unit test is a way of testing a unit - the smallest piece of code that can be logically isolated in a system.
- Function, a subroutine, a method or property.
- tests are not unit tests when they rely on external systems
- “If it talks to the database, it talks across the network, it touches the file system, it requires system configuration, or it can't be run at the same time as any other test."

Unit tests

- I write a class Scraper which lives in scraper.py
- I should then, immediately, in the same PR:
  - Write a class called test_scraper.py
  - Run those tests **while developing** to ensure my scraper works as expected
  - Create a small amount of dummy data locally if needed!
    - example.html page that you leave in test dir

Unit tests

- I write a class Scraper which lives in scraper.py
- I should then, immediately, in the same PR:
  - Write a class called test_scraper.py
  - Run those tests **while developing** to ensure my scraper works as expected
  - Create a small amount of dummy data locally if needed!
    - example.html page that you leave in test dir



Unit tests

```
From my_module import bad_adder


def test_sum():
    assert bad_adder([1, 2, 3]) == 7
```

Running Unit Tests

- There are many test runners available for Python.
- The one built into the Python standard library is called unittest.
- The most popular test runners are:
    - unittest
    - nose or nose2
    - pytest

You can also make your unittests run on every PR before they can be merged in github! This is good practice; I'll show you how later in this lecture.

Where to put tests

```
project/
│
├── src/
│   └── __init__.py
│   └── bad_adder.py
│
├── test/
  └── test_bad_adder.py
```

But I wrote my class in a way that makes it hard to test?!

- That probably means that you're not separating out the chunks of logic into reasonable functions...can you refactor?

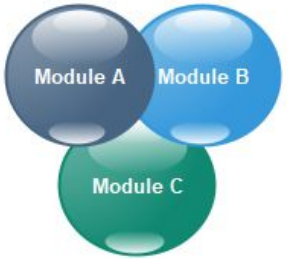
Mocking in Python

- As a developer, you care more that your library **successfully called the system function for changing volume** as opposed to experiencing volume going up every time a test is run.
- Mocks allow you to create “fakes” of classes
- Code Example: Facebook Graph API
- Really useful for testing a class which depends upon other things/external APIs
- Example of Mocking
-

Integration test

Type of testing where software modules are integrated logically and tested as a group.

Tested in Unit Testing



Under Integration Testing

Integration test

Purpose: Expose faults in the interaction between modules or units

Integration test

- Usually not run for every PR
- Run before pushing dev-> stage
- (can depend upon setup – sometimes good to run manually in code development if you suspect you're breaking things)

Load Testing

- How much load can my system handle before it becomes overwhelmed?
- “Load” - concurrent requests
  - Writing to your DB
  - Users accessing your site

Load Testing

- Load testing determines the maximum operating capacity of an application
- Determine whether the current infrastructure is sufficient to run the application
- Sustainability of application with respect to peak user load
- Number of concurrent users that an application can support, and scalability to allow more users to access it.
- Load Testing helps identify the bottlenecks in the system under heavy user stress scenarios before they happen in a production environment.

Load Testing

- Manual Load Testing:
  - Easy to just write a script to do this
  - Does not produce repeatable results
- Open source load testing tools:
  - Jmeter, locust, k6, many others
  - Free, less sophisticated, automatable
- Enterprise-class load testing tools:
  - They usually come with capture/playback facility.
  - They support a large number of protocols.
  - They can simulate an exceptionally large number of users.
  - AWS, GCP

Scalability Testing

- “How to” depends upon design of system
- During the scalability testing, different system parameters are taken into account, such as:
  - CPU usage
  - Network bandwidth consumption
  - Throughput
  - Number of requests processed within a stipulated time
  - Latency
  - Memory usage of the program
  - End-user experience when the system is under heavy load and so on.
- In this testing phase, simulated traffic is routed to the system to study how the system behaves and scales under a heavy load.
- Predictable “spikes” (i.e., shopping on Cyber Monday)

Security Testing

- Security Testing is a form of software testing performed to evaluate the security of a system or application.
- Security testing ensures the system's safety from hackers, viruses, or cyber threats.
- Often done manually – expert tries to break into system from outside
- If you don't do this, the world will do it for you upon launch ;)

Chaos Engineering

- Chaos Engineering is an approach to identifying failures before they become outages.
- Proactively causing a system to fail and seeing how it responds to failure conditions,
- Identify and fix failures before they become public facing outages.
- Netflix Chaos Monkey: Randomly terminate instances.
    - https://netflix.github.io/chaosmonkey/

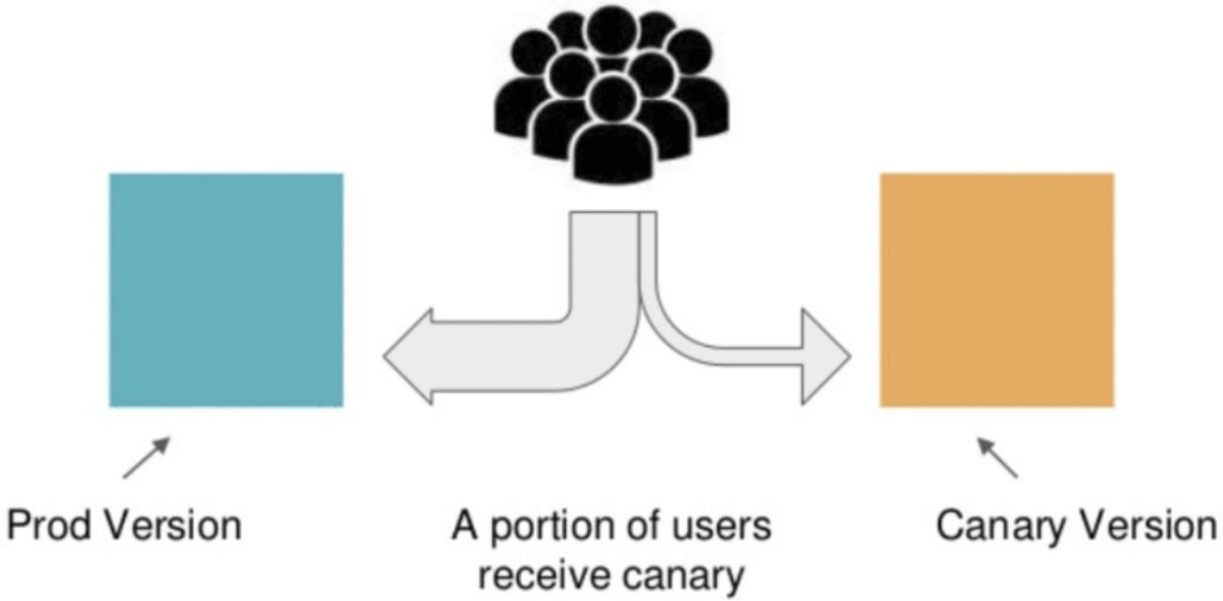
Canary rollouts

- Canary birds are sensitive to gas and show visible distress when detecting it.
- Coal miners in the past used canary birds as an early warning system for harmful gasses like carbon monoxide (CO) and methane (CH_4).
- Canary birds alerted the miners of danger before they recognized it.

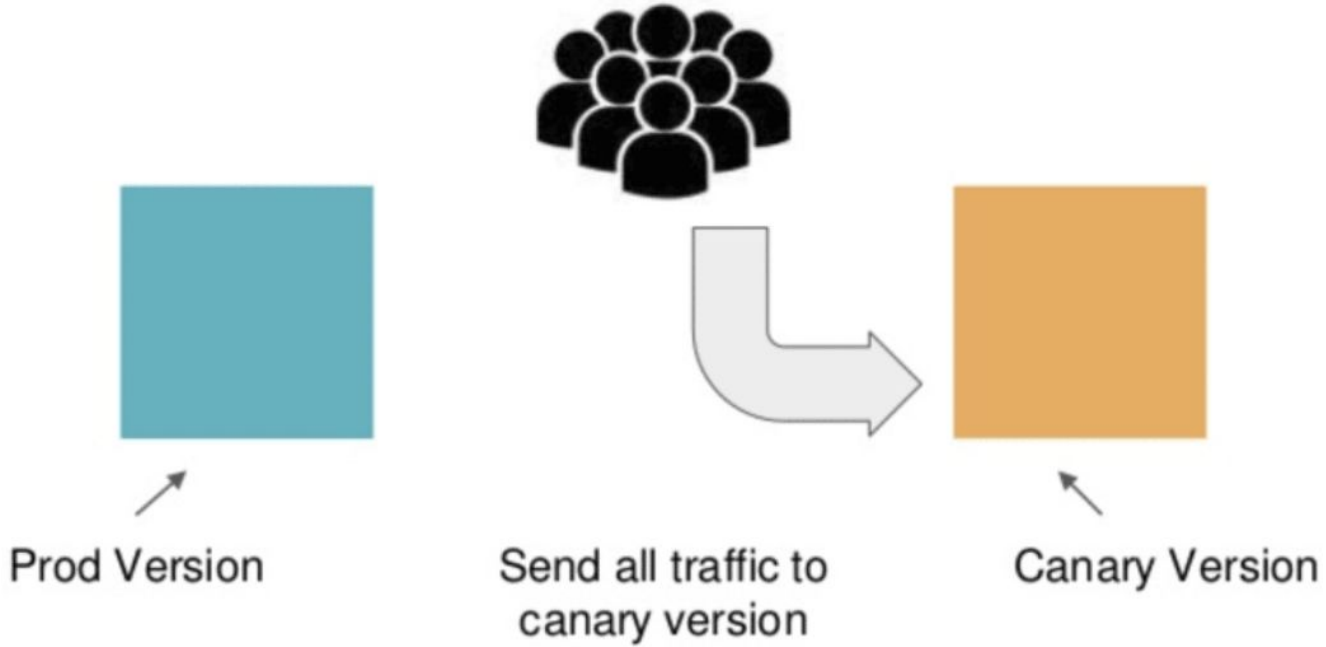
Canary rollouts

- A canary deployment is a method that exposes a new feature to an early sub-segment of users.
- The goal is to test new functionality on a subset of customers before releasing it to the entire user base.
- You can choose randomly or a specific group of users and roll back if anything breaks.
- If everything works as intended, you can gradually add more users while monitoring logs, errors, and software health.

Canary rollouts: Process



Canary rollouts: Process



Canary rollouts: Process

- In the beginning, the current version receives 100% of user traffic
- A new deployment, the “canary” is performed with brand new pods and only a small amount of traffic, e.g. 5% while maintaining 95% of users on the older version.
- A decision regarding the current canary/subset of traffic takes place in an automated manner.
    - If the new version works as expected, a larger portion of live traffic is sent to the new version and the process repeats again for different percentages of canary traffic (e.g. 5%, 25%, 50%, 75%, 100%).
    - If the new version has issues, the service is switched back to the original version. This has minimal impact on most users. The canary version is destroyed and everything is back to the original state.
- In the end, 100% of traffic goes to the new version and the old version can be discarded.
-

Pre-Commit Hooks

- Pre-Commit Hooks – code that runs before you commit, or before you merge your commit to the shared branch.
- Before every piece of code is merged, it has to pass some basic tests!
  - Linting
  - Unused imports
  - Unit tests
- Pre-commit library

Github Actions

- You can use Github to run your pre-commit checks on every PR, and only allow merging if the tests pass
- Here is how to set up with Github

Browser Testing

- Check whether things are appearing on the page/ in the browser as expected
- Many open-source tools that automate web browsers
  - selenium
  - cucumber
  -

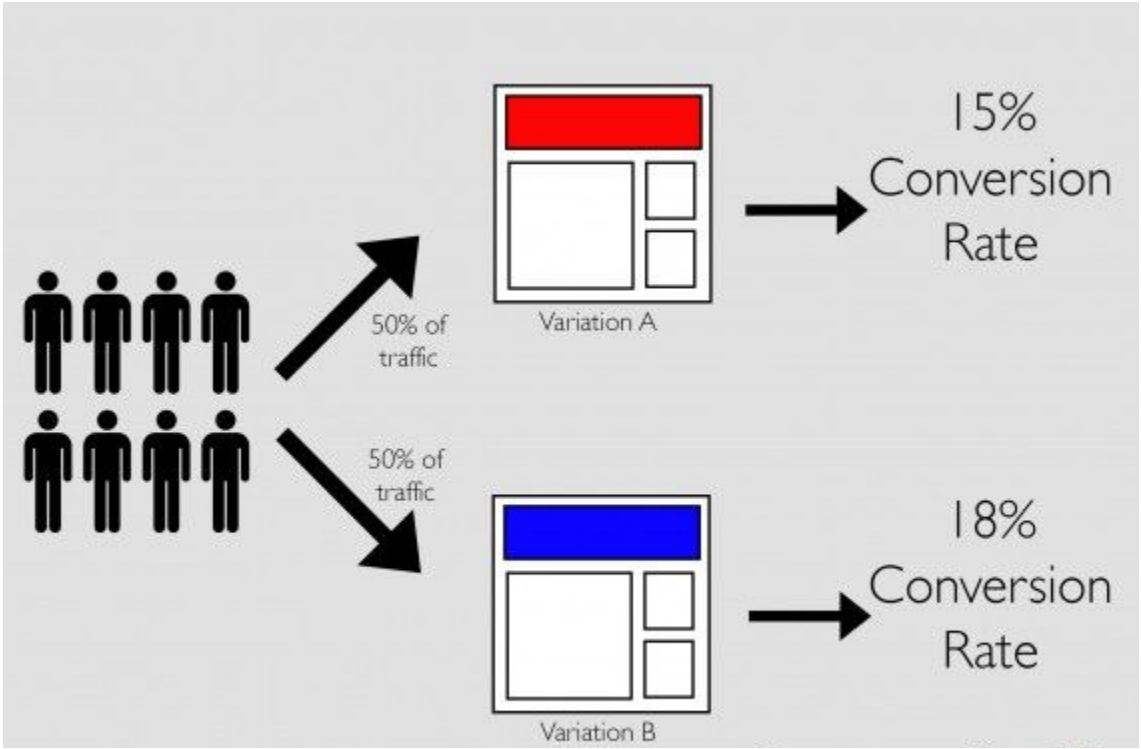
Acceptance Testing

- Acceptance testing is a quality assurance (QA) process that determines to what degree an application meets end users' approval.
- Depending on the organization, acceptance testing might take the form of:
  - beta testing
  - application testing
  - field testing
  - end-user testing.
- Last phase of process
- Often manual QA with humans

SxS (for ML models,)

- Take some number of example inputs
- Have humans rate `old_model(input)` vs `new_model(input)`
- Only deploy if better
  - Or the same plus more performant, bugfix, etc
  - Statistical significance

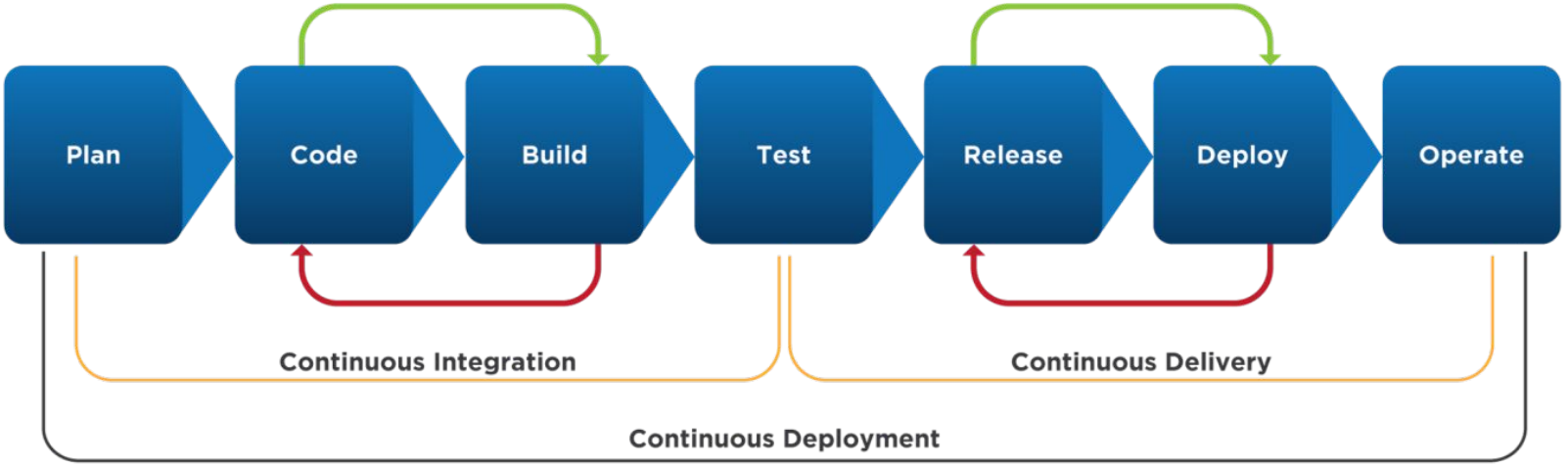
A/B testing


50% of traffic
50% of traffic
Variation A
Variation B
15% Conversion Rate
18% Conversion Rate

In Review: Code Deployment Timeline

1. Write Code
2. Run Unit tests
3. Push to github branch
4. Code review from other devs
5. Merge to Dev branch
6. Run Integration tests
7. Dev->stage
8. Run loadtests and performance tests
9. Q/A or acceptance test
10. Stage->prod
11. Monitoring and alerting

Code Deployment: A Too-Complete Diagram



NOTE FOR PROJECT: Django uses MTV

- The Model-View-Template (MVT) is slightly different from MVC.
- The main difference between the two patterns is that Django itself takes care of the Controller part (Code that controls the interactions between the Model and View)
- In Django-land, a “view” is a Python callback function for a particular URL, because that callback function describes which data is presented.
- In Django, a “**view**” describes which data is presented
    - a view normally delegates to a **template**, which describes **how** the data is presented.
- Where does the “controller” fit in, then?
    - In Django’s case, it’s probably the framework itself: the machinery that sends a request to the appropriate view, according to the Django URL configuration.
    - Also somewhat the view..it’s not a totally clean mapping tbh