

SYSTEM TESTING WITH PYTEST AND DOCKER-PY

By **Christie Wilson (@bobcatwilson)**
& **Michael Tom-Wing (@mtomwing)**
github.com/keeppythonweird/pytest-dockerpy



This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](https://creativecommons.org/licenses/by-nc-nd/4.0/).

HELLO!

- Christie Wilson
 - Senior Developer @ Demonware
 - Team Lead: Test Tools
- Michael Tom-Wing
 - Software Engineer in Test @ Demonware
 - Focus on automation and quality
- We're from Canada!



goo.gl/JkzmYJ



demonware

- Video Game Industry
- Online services for games
- Come see us in the vendor area!

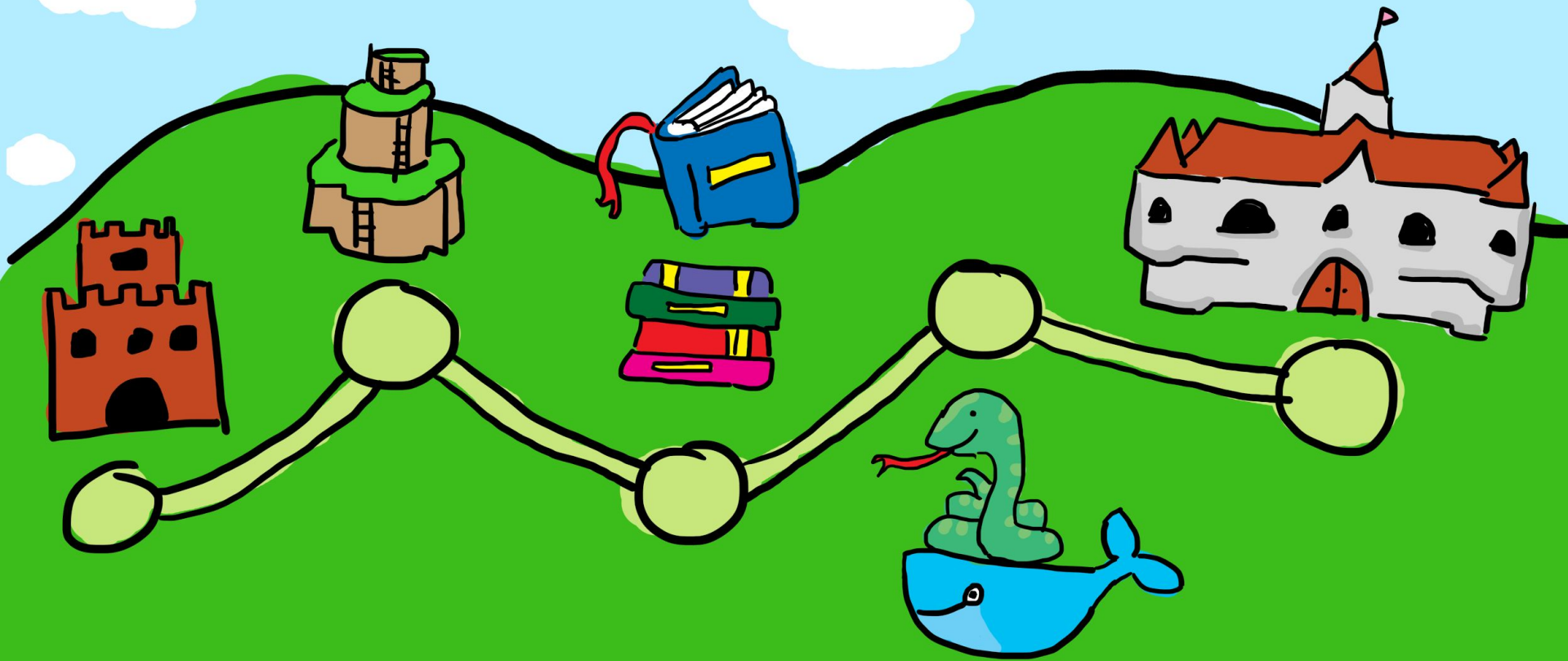


COMMANDER MCFLUFFLES AND THE QUEST FOR QUALITY

Once upon a system test



OVERVIEW

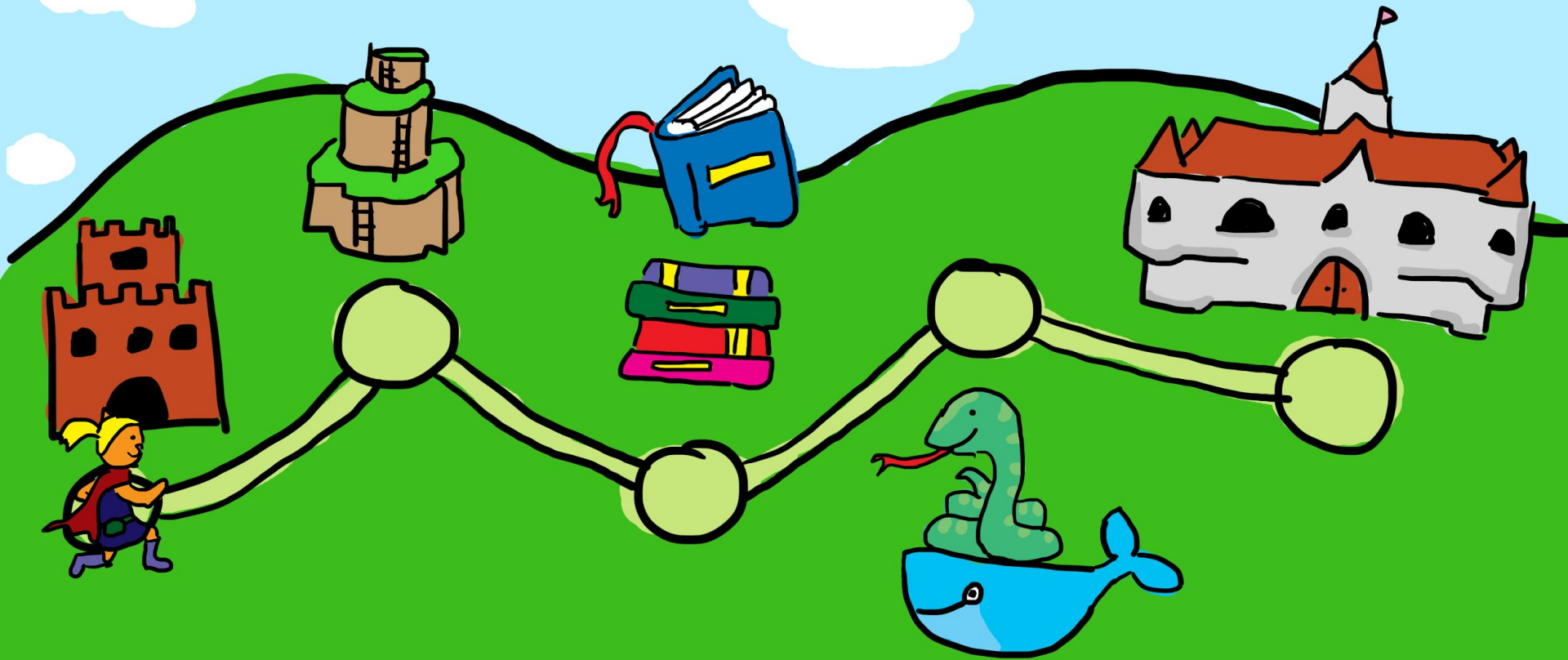


@bobcatwilson

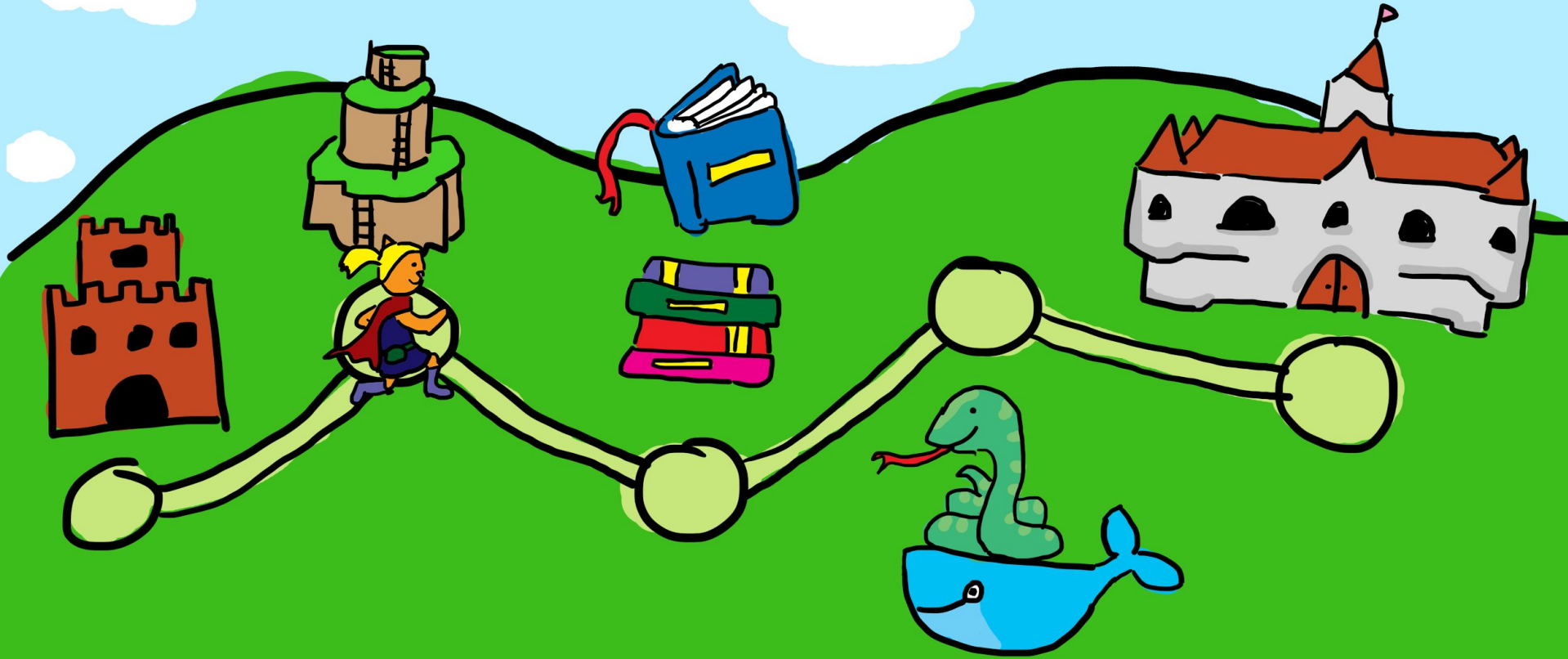
goo.gl/JkzmYJ

@mtomwing

TESTING AT DEMONWARE 2011-2016



WHAT IS TESTING?

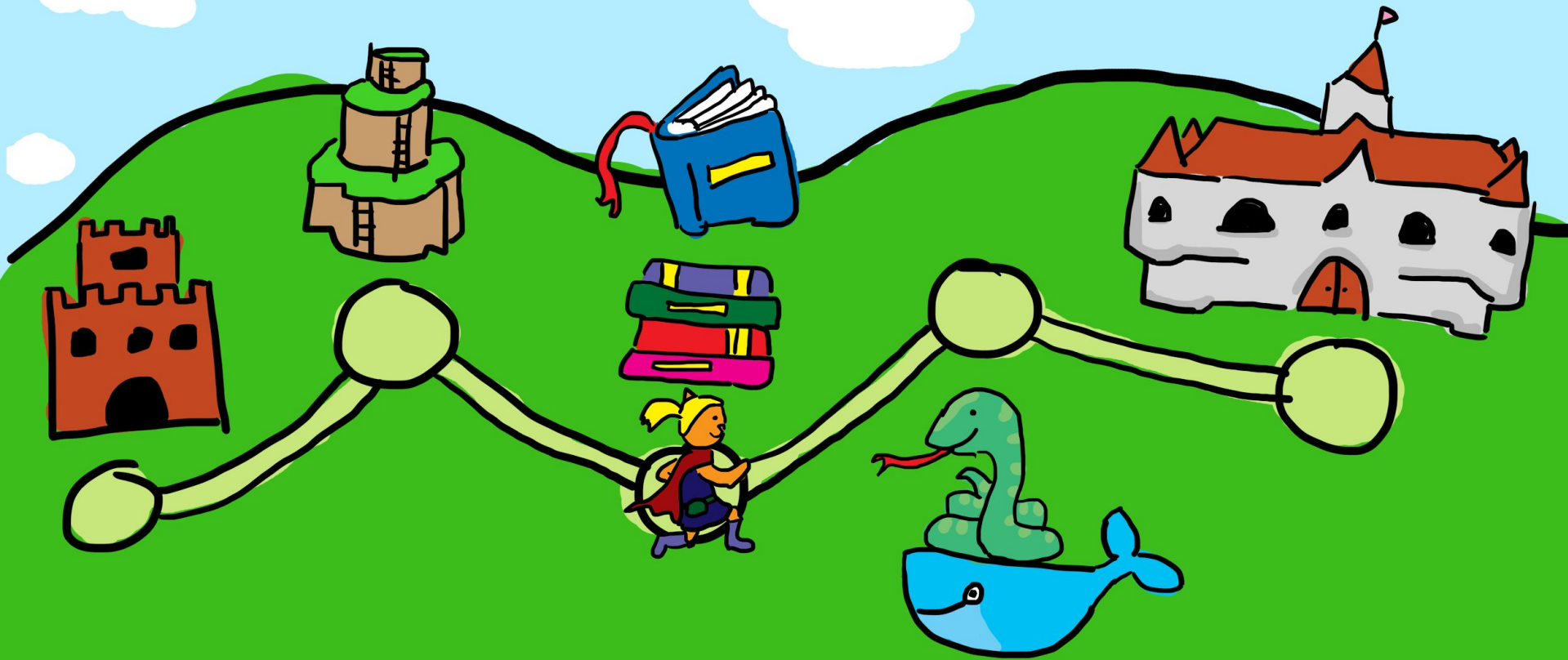


@bobcatwilson

goo.gl/JkzmYJ

@mtomwing

BEST PRACTICES FOR SYSTEM TESTING

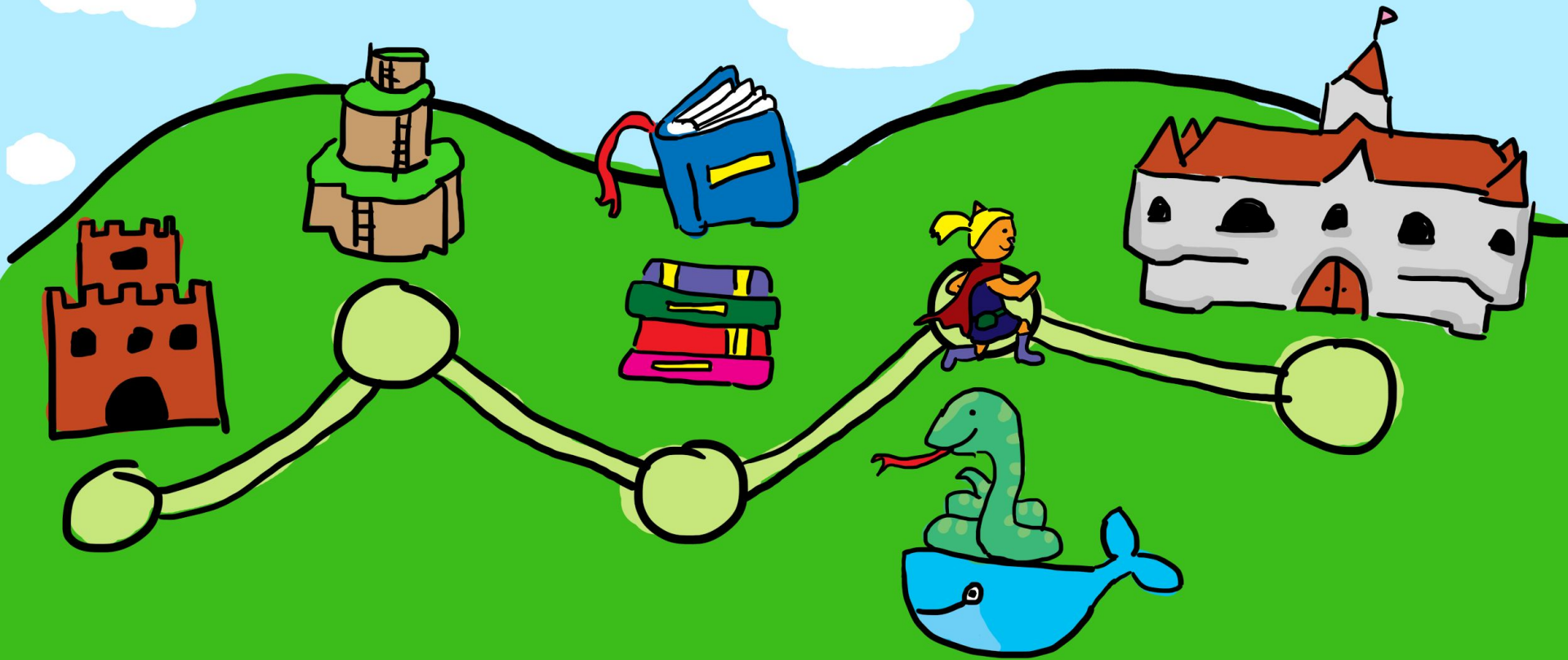


@bobcatnilson

goo.gl/JkzmYJ

@mtomwing

PYTEST FIXTURES AND DOCKER-PY

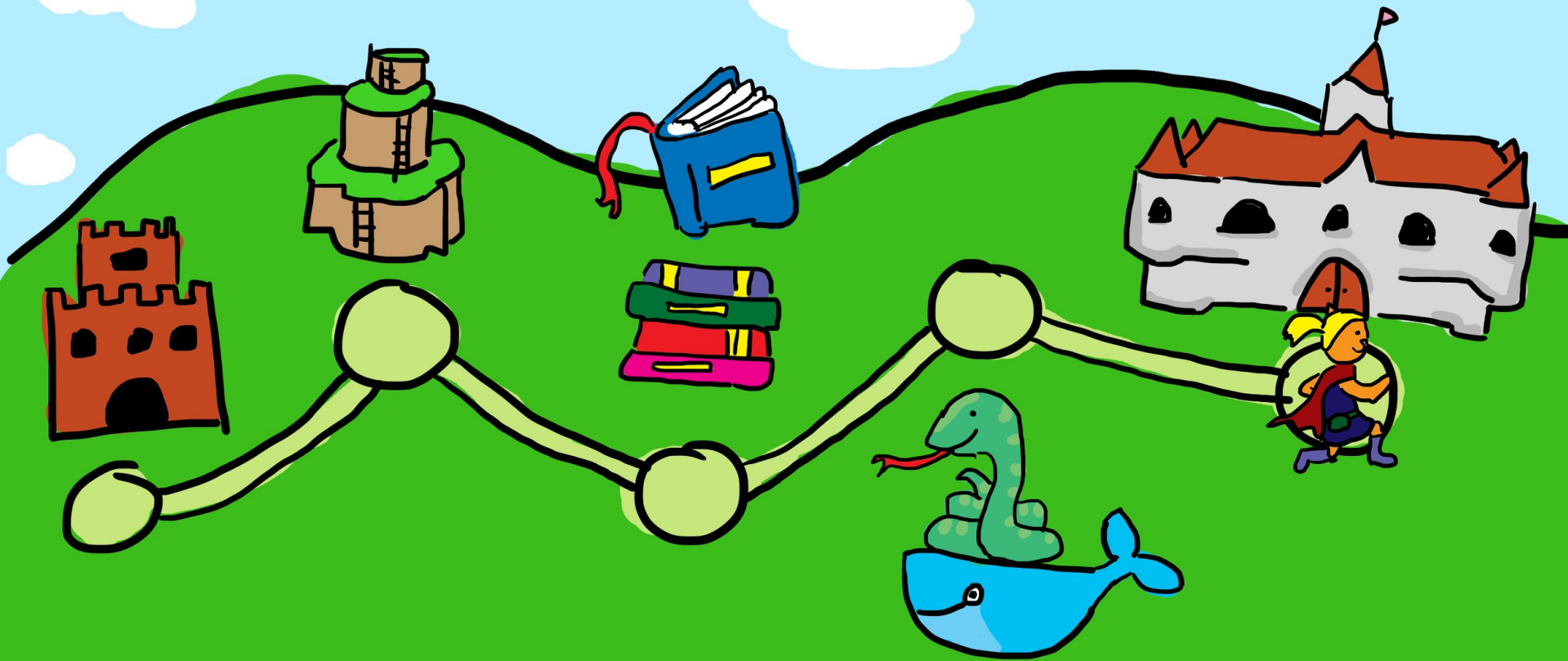


@bobcarlson

goo.gl/JkzmYJ

@mtomwing

TAKEAWAYS FOR DEV AND OPS



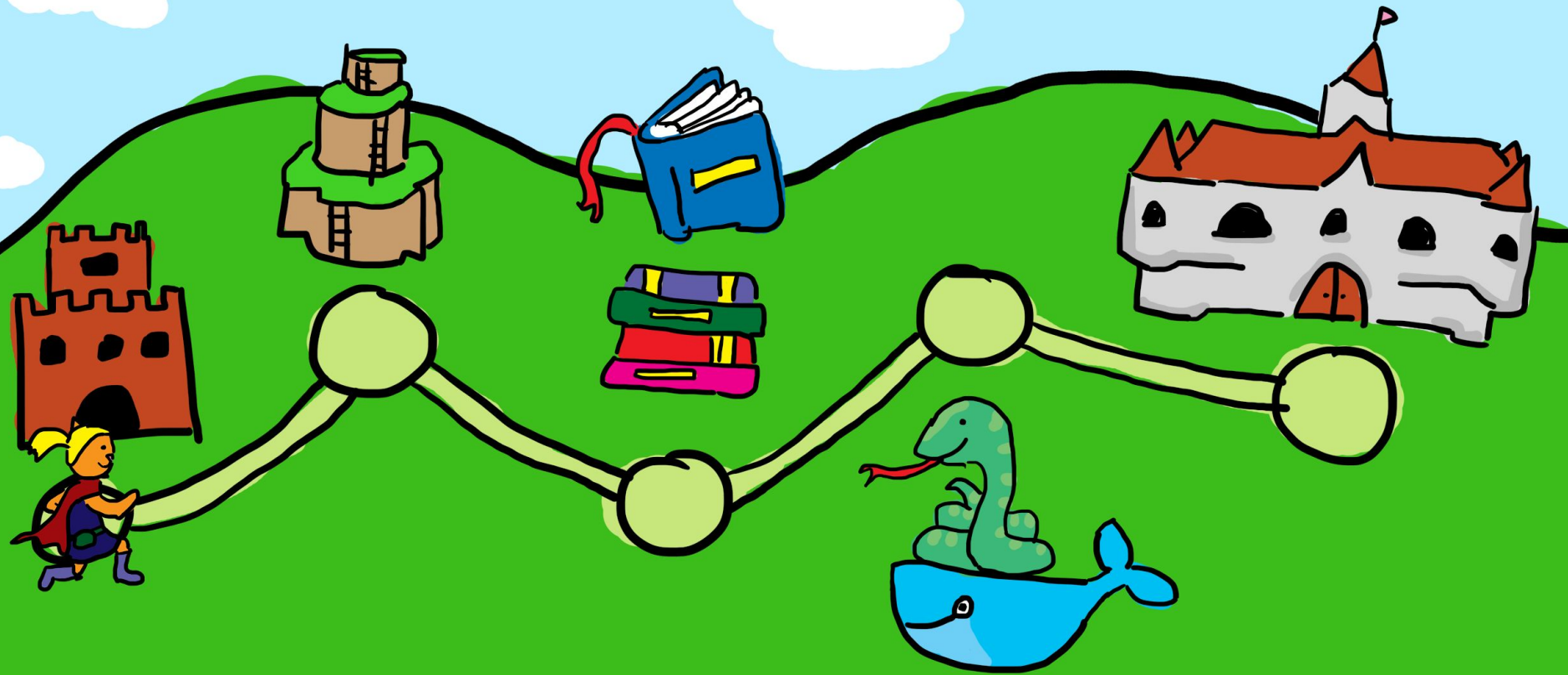
@bobcatwilson

goo.gl/JkzmYJ

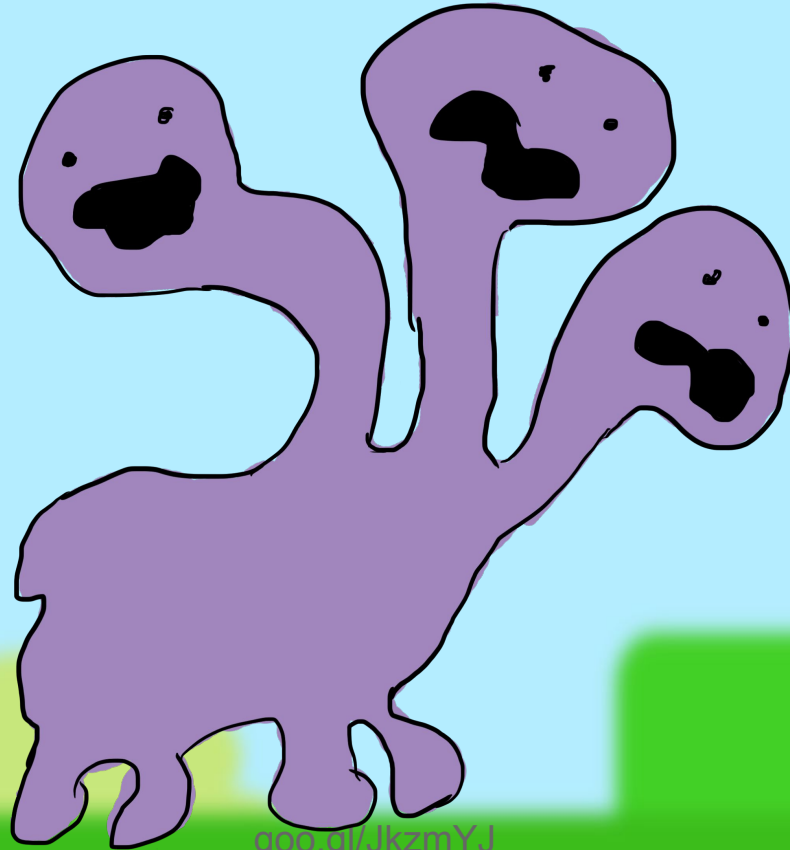
@mtomwing

ONCE UPON A
SYSTEM TEST

TESTING AT DEMONWARE 2011-2016



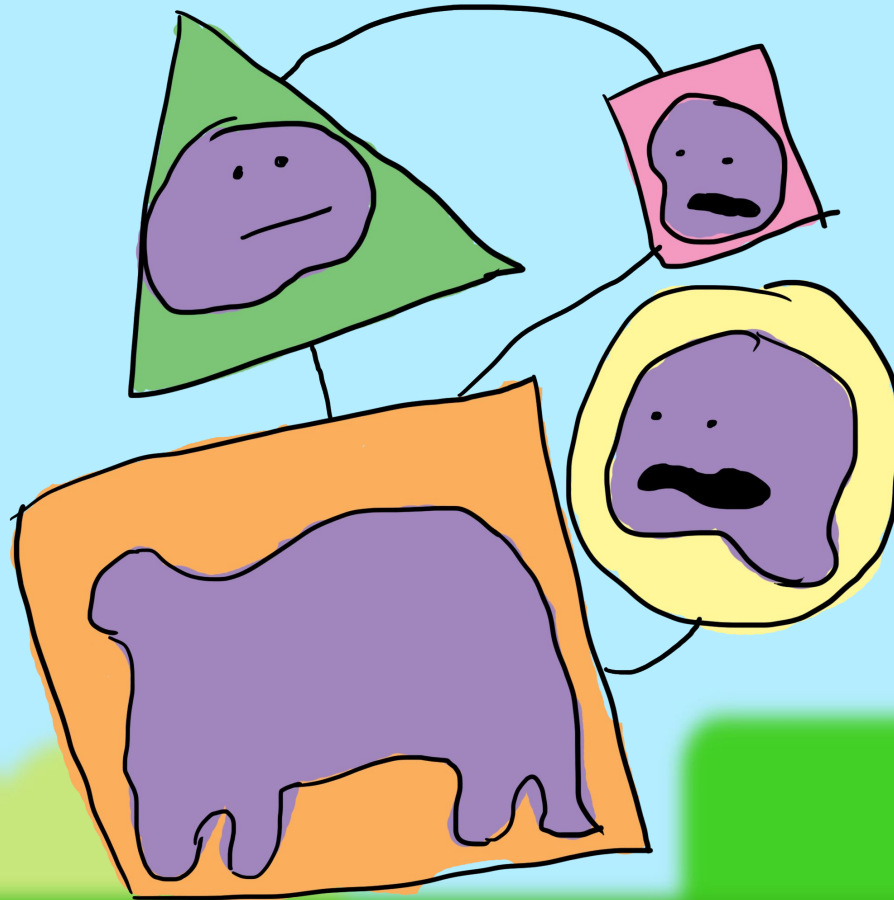
DEMONWARE - 2011



DEMONWARE - RIP OTHER TESTING METHODS



DEMONWARE - 2016



@brockmison

goo.gl/JkzmYJ

@mtomwing

DEMONWARE - 2016

- Test Tools team
- Variety of tests
 - Unit tests
 - Integration tests
 - System tests

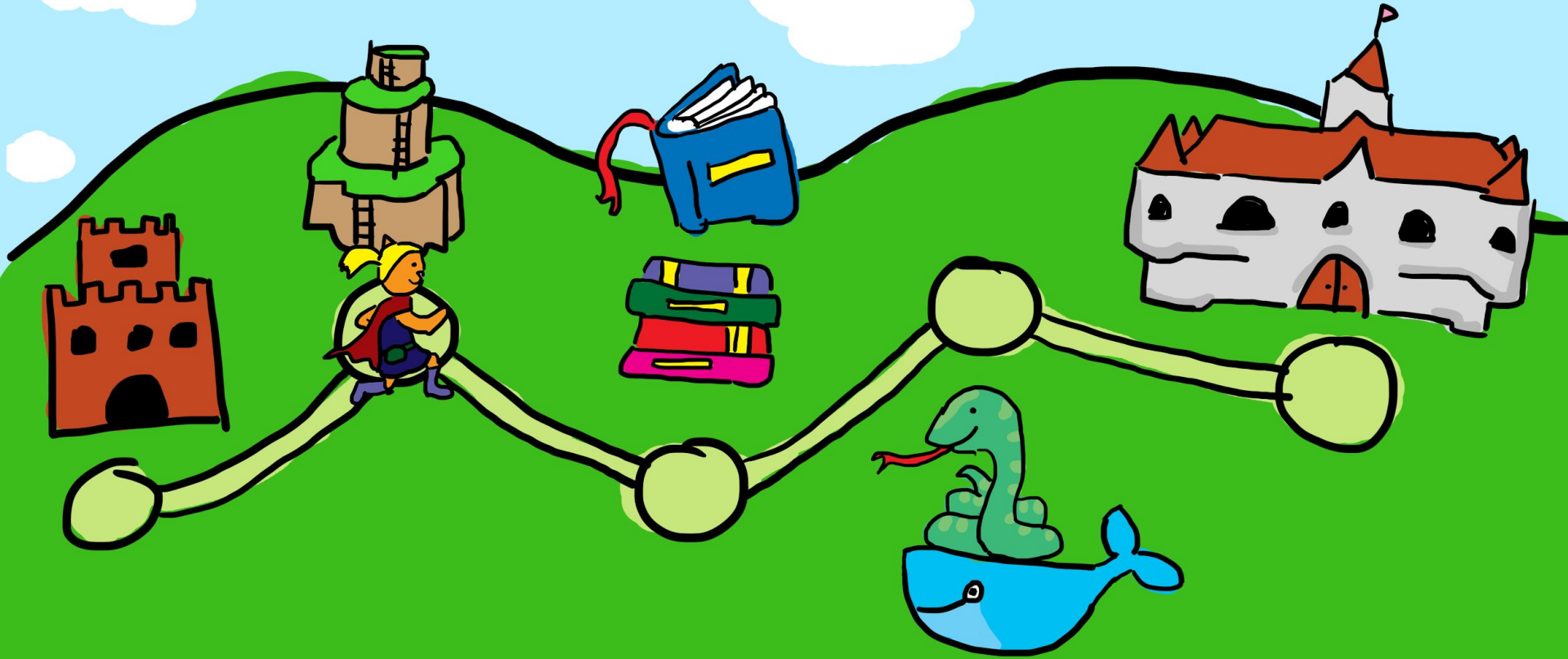


@brocknison

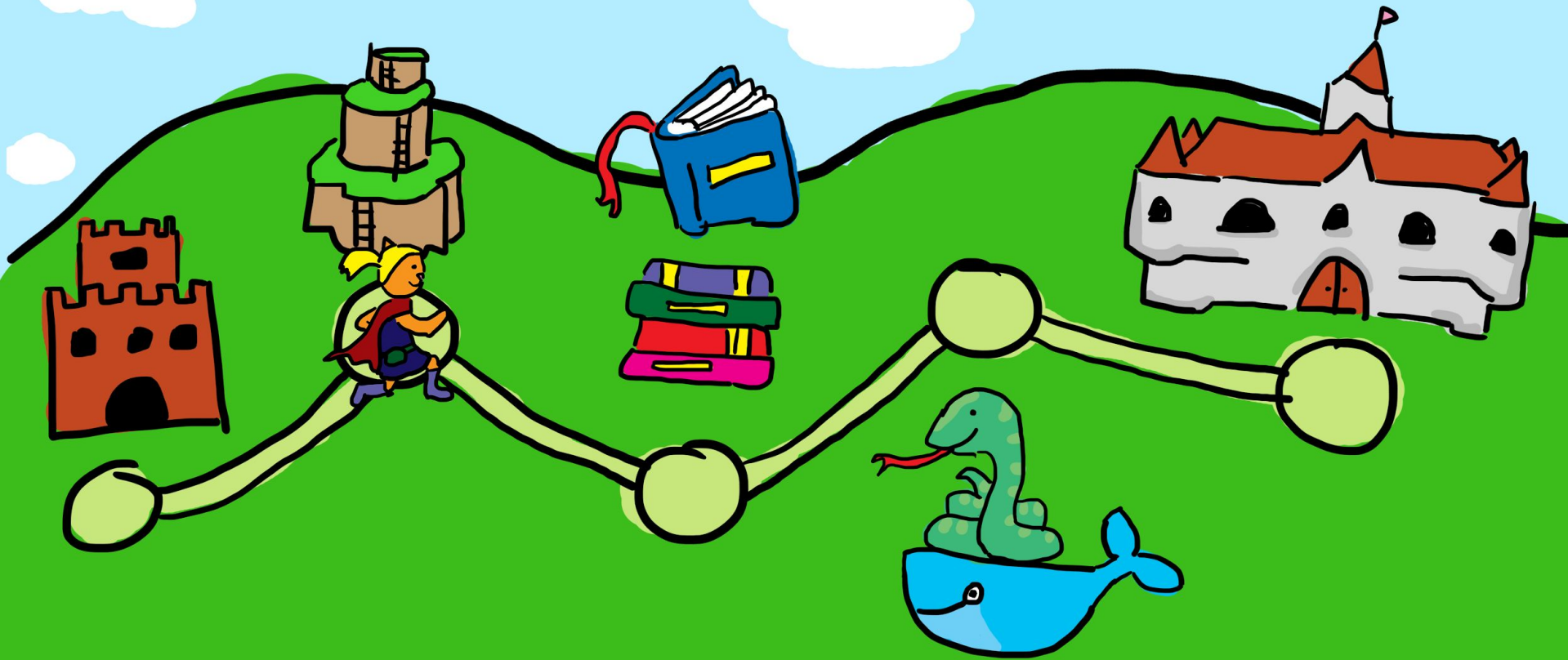
goo.gl/JkzmYJ

@mtomwing

WHAT IS TESTING?



~~WHAT IS TESTING?~~ WHY DO WE TEST?



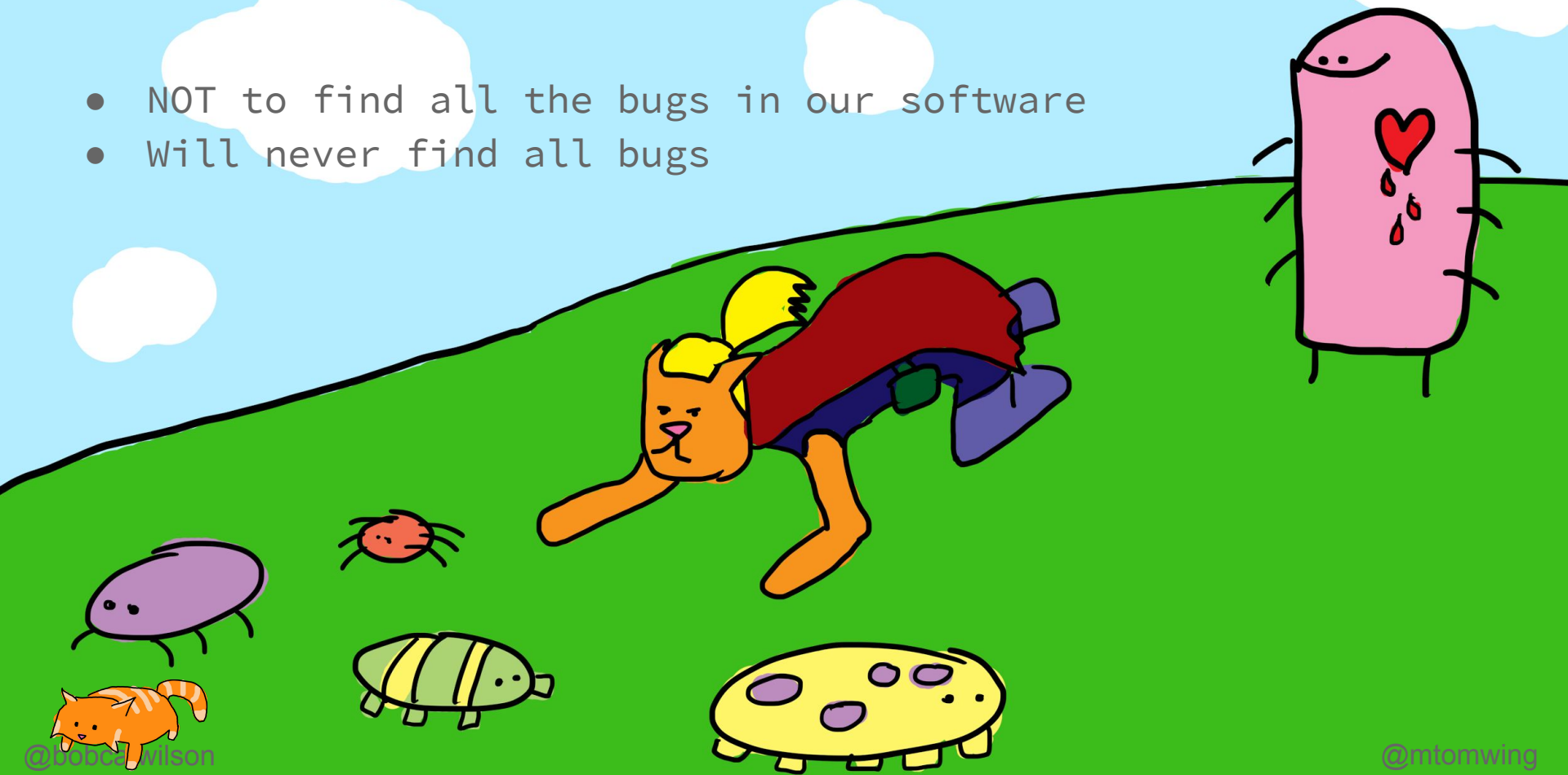
WHY DO WE TEST?

- To increase confidence in our software
- Avoid regressions
- Document behaviour



WHY DON'T WE TEST?

- NOT to find all the bugs in our software
- Will never find all bugs



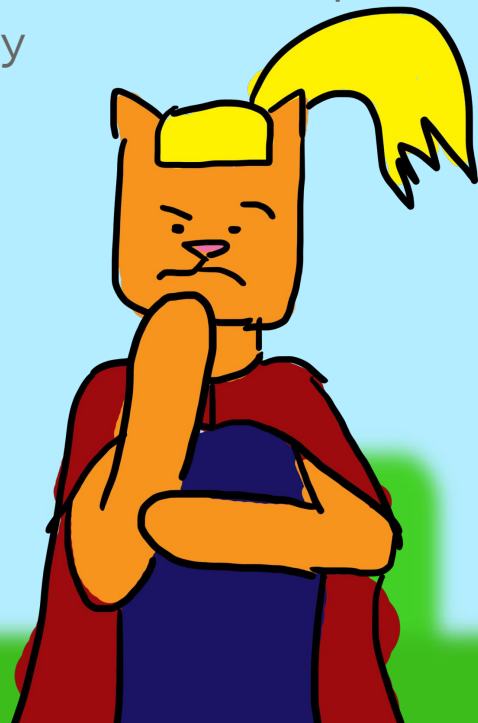
TESTING + SOFTWARE QUALITY

- Testing does not increase the quality of our software
- By the time our tests run our software is already buggy
 - Introduce quality through requirements + design!
- But...

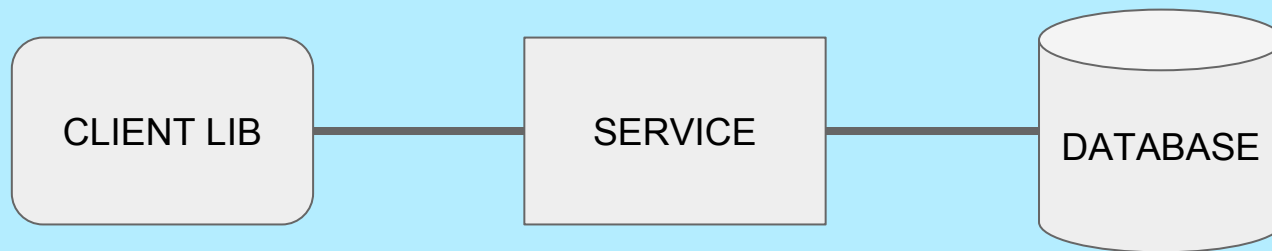


TESTING + SOFTWARE QUALITY

- Tests provide metrics to let us reason about quality
 - E.g. coverage, timing, # of logs
- Untested software is always VIEWED as lower quality
- Less information about the quality



EXAMPLE



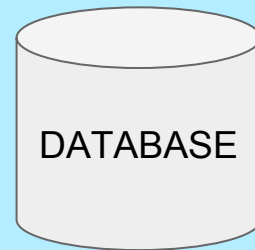
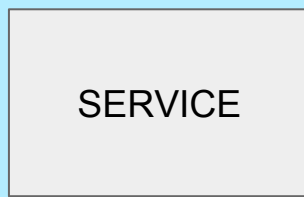
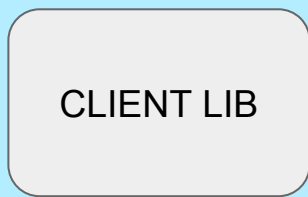
HOW TO TEST IT

- Unit tests
- Integration tests
- System tests



UNIT TESTS

- Unit tests: ~100% coverage
- Integration tests
- System tests



INTEGRATION TESTS

- Unit tests: ~100% coverage
- Integration tests: service \leftrightarrow DB
- System tests



SYSTEM TESTS

Test the entire system

PROS

- Most valuable
- Most likely to find bugs

CONS

- Slowest
- Hardest to maintain

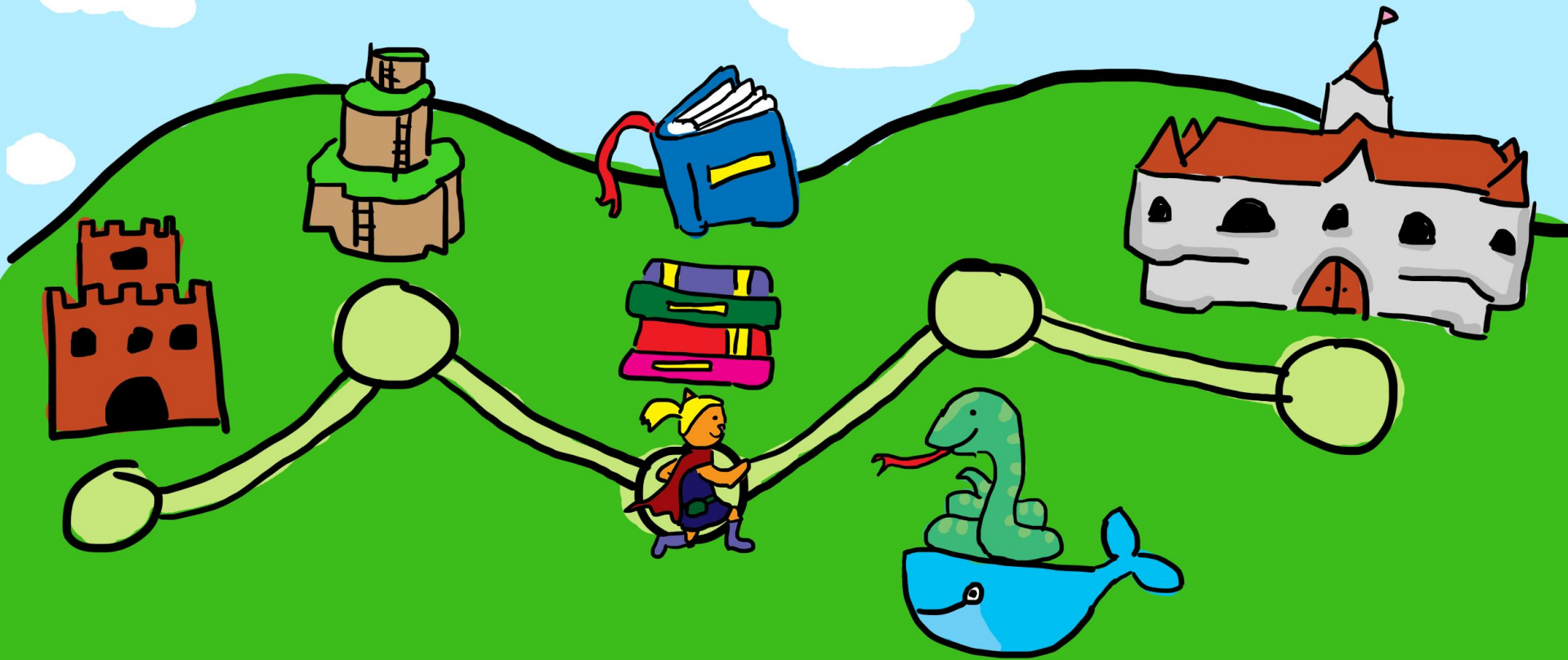


SYSTEM TESTS

- Unit tests: ~100% coverage
- Integration tests: service \leftrightarrow DB
- System tests: happy path, few simple failures



BEST PRACTICES FOR SYSTEM TESTING



BEST PRACTICES

- Use fresh state between tests
- Will help avoid dependencies between tests
 - e.g. test ordering shouldn't affect their outcomes
- Docker helps make this very easy!



BEST PRACTICES

- Ensure tests can run on build servers and locally
 - Ease the burden for writing and running tests
- Restrict the test environments you'll support
 - e.g. Linux, OSX, toaster



@doochwilson

goo.gl/JkzmYJ



@mtomwing

BEST PRACTICES

- Tests should clean up after themselves
- Fail fast
- Fail informatively

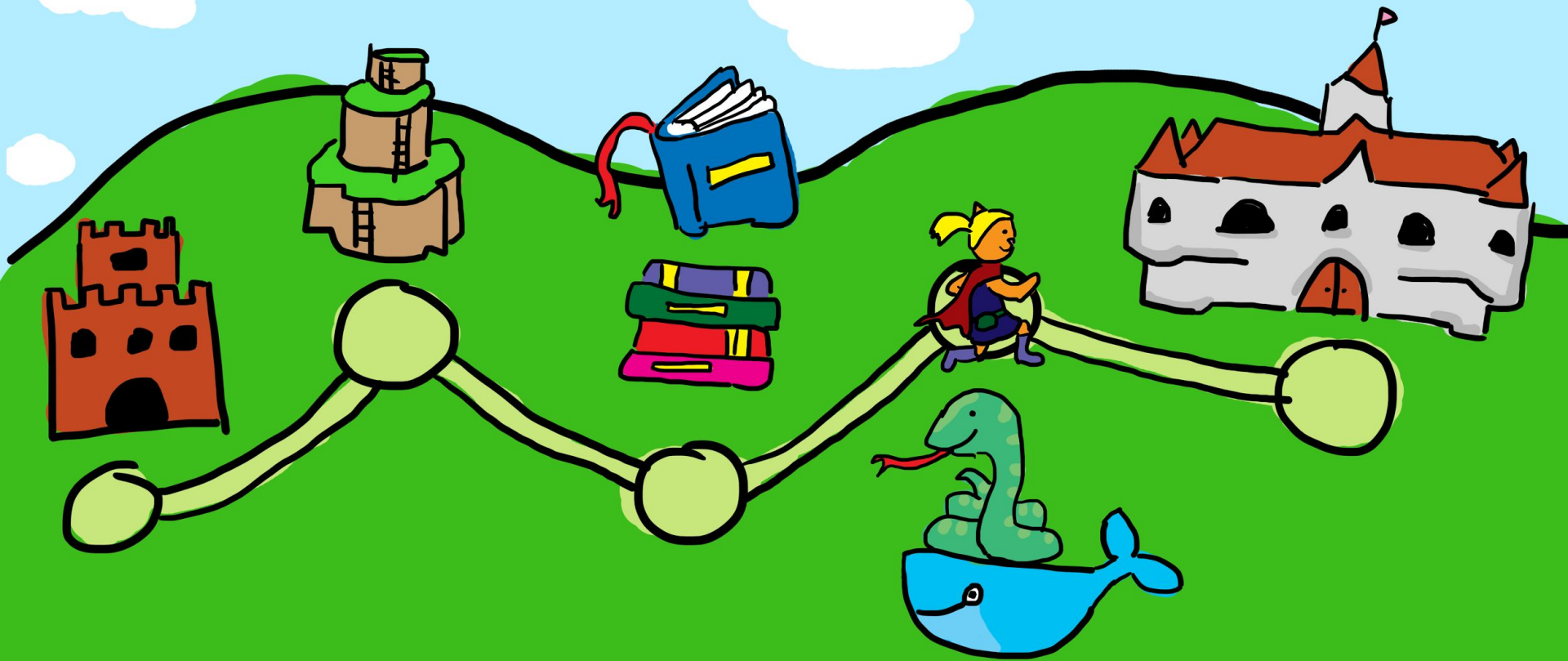


GLUE CODE

```
def glue_code():  
    with open('file') as f:  
        result = my_module.do_something(f.readlines())  
    other_result = other_module.do_things(result)  
    do_something_amazing(other_result)  
    return good_things(os.getcwd())
```



PYTEST FIXTURES AND DOCKER-PY



PYTEST

- Python testing library
- v.s. unittest = less boilerplate
- More batteries included
 - e.g. fixtures, plugins



PYTEST FIXTURES

- Provide setup and teardown for tests
- Pytest will ensure that the setup and teardown always happen
 - And in that order!
- System tests generally set up a lot of things!
- Very slick!



PYTEST FIXTURES

SETUP

TEARDOWN

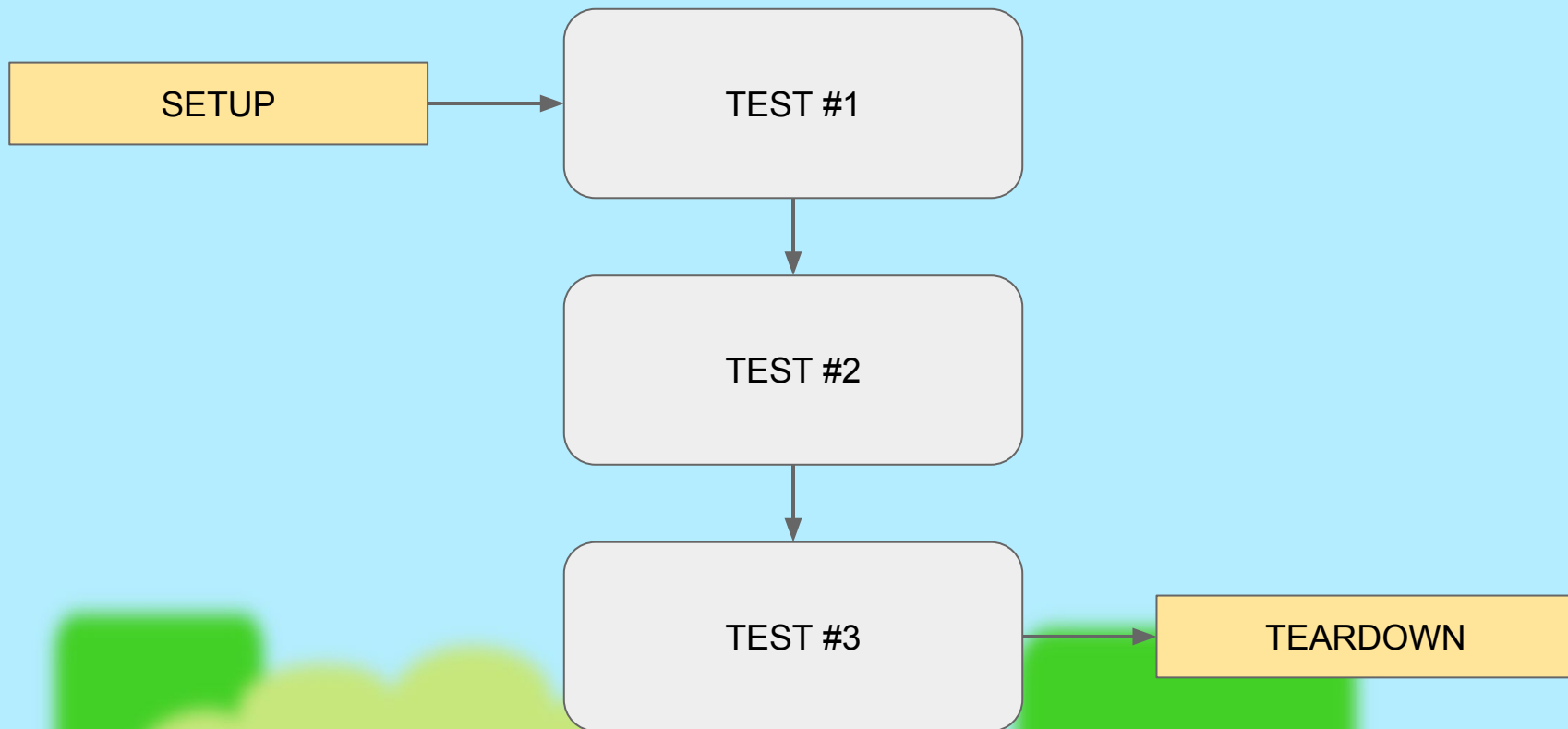
YOUR TEST



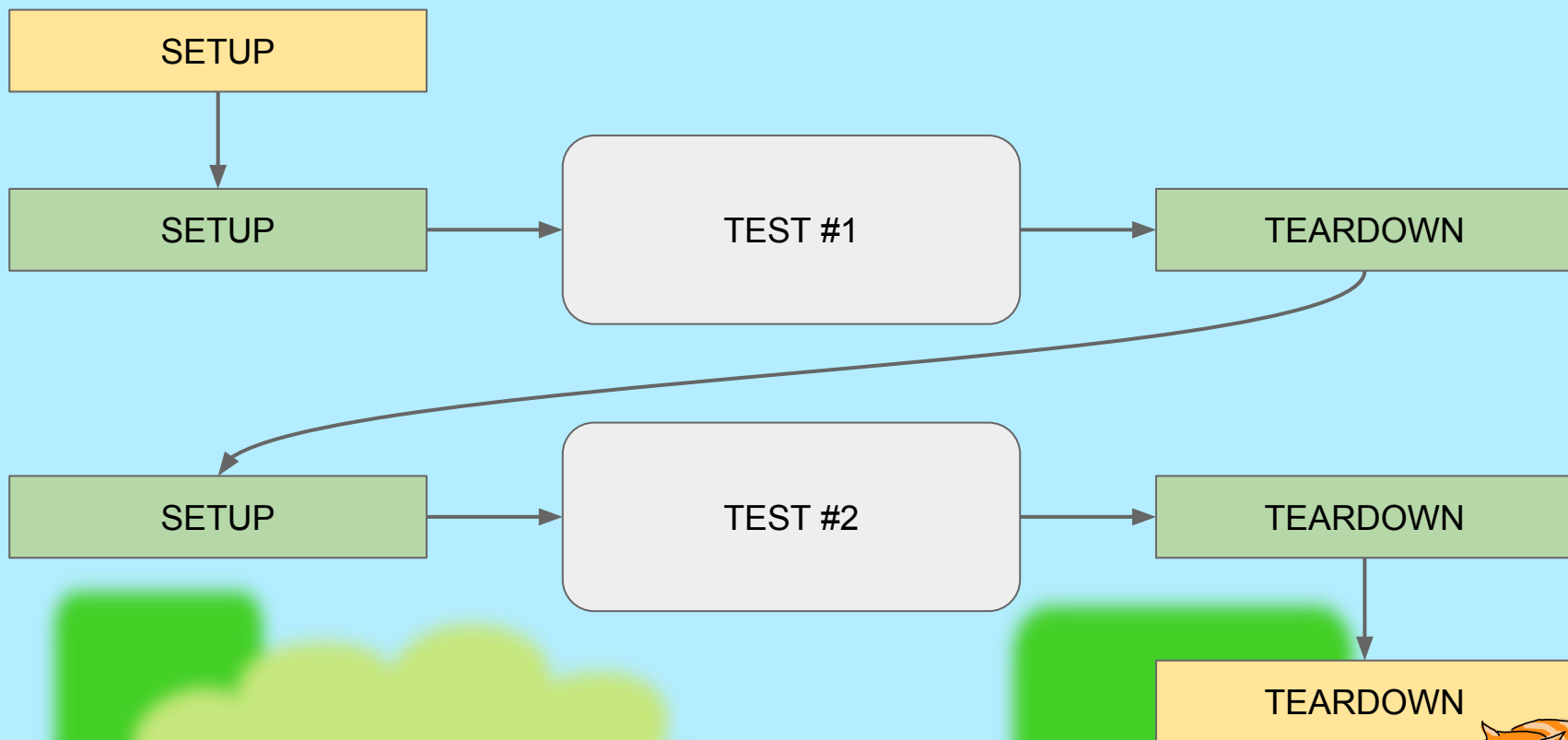
PYTEST FIXTURES



PYTEST FIXTURES - SCOPE



PYTEST FIXTURES - SCOPE



DOCKER

DOCKER DOCKER DOCKER DOCKER DOCKER DOCKER DOCKER DOCKER

DOCKER DOCKER DOCKER DOCKER DOCKER DOCKER DOCKER DOCKER

DOCKER DOCKER DOCKER DOCKER DOCKER DOCKER DOCKER DOCKER

DOCKER DOCKER DOCKER DOCKER DOCKER DOCKER DOCKER DOCKER

DOCKER DOCKER DOCKER DOCKER DOCKER DOCKER DOCKER DOCKER

DOCKER DOCKER DOCKER DOCKER DOCKER DOCKER DOCKER DOCKER

DOCKER DOCKER DOCKER DOCKER DOCKER DOCKER DOCKER DOCKER

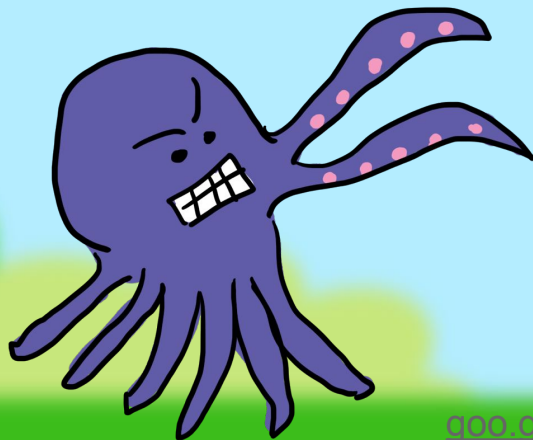


DOCKER

- Challenging service setup = docker
- Setup + teardown: bash scripts



@bobwilson



goo.gl/JkzmYJ



@mtomwing

DOCKER-PY

- Python library for using docker
- Interface is 1:1 with the REST interface
 - Can be a bit clunky



DOCKER-PY

1. Create client
2. Pull image
3. Create container
4. Start container
5. Remove container



CREATE CLIENT

```
import docker
docker_client = docker.Client(
    'unix:///var/run/docker.sock',
    version='auto')
```



@bobcatwilson

goo.gl/JkzmYJ



@mtomwing

PULL IMAGE

```
docker_client.pull('percona:5.6')
```



PULL IMAGE - ERRORS

```
response = self._docker_client.pull('busybox:latest')  
  
lines = [line for line in response.split('\n') if line]  
pull_result = json.loads(lines[-1])  
  
if 'error' in pull_result:  
    raise Exception(pull_result['error'])
```

Thanks Steven Erenst!



@bobcatwilson

goo.gl/JkzmYJ



@mtomwing

CREATE CONTAINER

```
container = docker_client.create_container(  
    image='busybox:latest',  
    labels=['docker-test-log'])
```



goo.gl/JkzmYJ



START CONTAINER

```
docker_client.start(container=container["Id"])
```



@bobcatwilson

goo.gl/JkzmYJ



@mtomwing

KILL AND REMOVE CONTAINER

```
docker_client.remove_container(  
    container=container["Id"],  
    force=True,  
)
```



@bobcatwilson

goo.gl/JkzmYJ

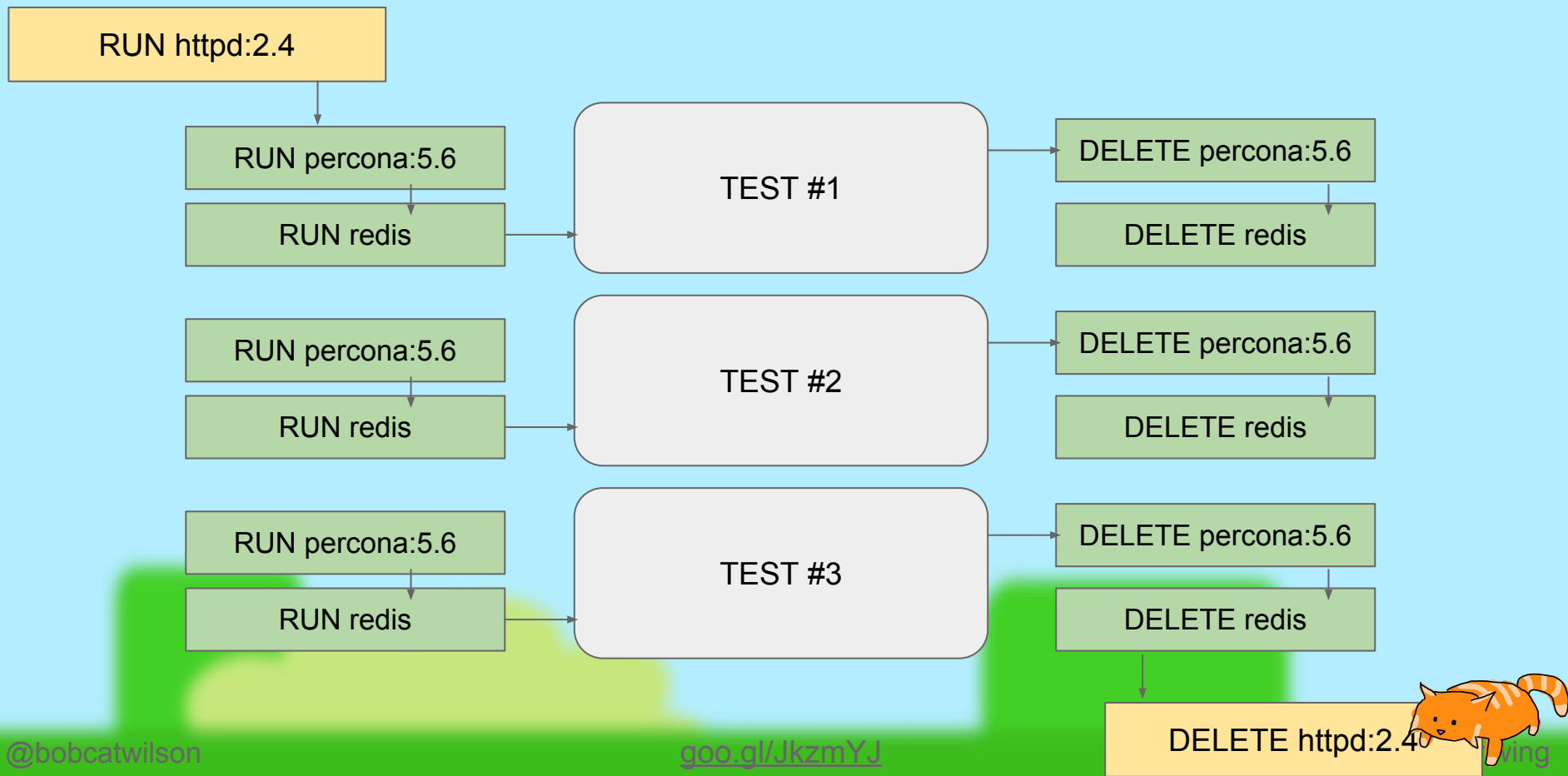


@mtomwing

PYTEST FIXTURES + DOCKER-PY



PYTEST FIXTURES + DOCKER-PY



PYTEST FIXTURES + DOCKER-PY

```
import docker
import pytest
@pytest.yield_fixture
def example_container():
    docker_client = docker.Client('unix://var/run/docker.sock', version='auto')
    docker_client.pull(IMAGE)
    container = docker_client.create_container(
        image=IMAGE,
        detach=True,
        labels=[labels.CONTAINERS_FOR_TESTING_LABEL]
    )
    docker_client.start(container=container["Id"])
    container_info = docker_client.inspect_container(container.get('Id'))

    yield container_info["NetworkSettings"]["IPAddress"]

    docker_client.remove_container(
        container=container["Id"],
        force=True
    )
```

PYTEST HOOKS

- pytest magic!

```
def pytest_runtest_logreport(report):
```

PYTEST FIXTURES - DOCKER LOGS

```
def pytest_runtest_logreport(report):  
    if report.failed:  
        docker_client = _docker_client()  
        test_containers = docker_client.containers(  
            all=True,  
            filters={"label": labels.CONTAINERS_FOR_TESTING_LABEL})  
        for container in test_containers:  
            log_lines = [  
                ("docker inspect {!r}:".format(container['Id'])),  
                (pprint.pformat(docker_client.inspect_container(container['Id']))),  
                ("docker logs {!r}:".format(container['Id'])),  
                (docker_client.logs(container['Id'])),  
            ]  
            report.longrepr.addsection('docker logs', os.linesep.join(log_lines))
```



PYTEST FIXTURES - DOCKER LOGS



WHAT ABOUT DOCKER-COMPOSE?

- Works well when the deployment is static between tests
 - Not as well suited when deployment is different for each test
- Integrates with pytest fixtures!
 - e.g. Use a fixture to run docker-compose up
- docker-py can help



@bobcat.wilson

goo.gl/JkzmYJ



@mtomwing

WHAT ABOUT DOCKER-COMPOSE?

```
@pytest.fixture
def docker_client():
    return docker.Client('unix://var/run/docker.sock', version='auto')

@pytest.fixture
def my_cluster(request):
    def fin():
        subprocess.check_output(
            shlex.split('docker-compose down'))

    request.addfinalizer(fin)
    subprocess.check_output(
        shlex.split('docker-compose up -d'))

@pytest.fixture
def some_container_ip(my_cluster, docker_client):
    output = docker_client.inspect_container(SOME_CONTAINER)
    return output['NetworkSettings']['Networks'][DOCKER_COMPOSE_NETWORK_NAME]['IPAddress']
```



GOTCHAS

- Wait for the service to start (backoff)
- Maximize container startup speed!



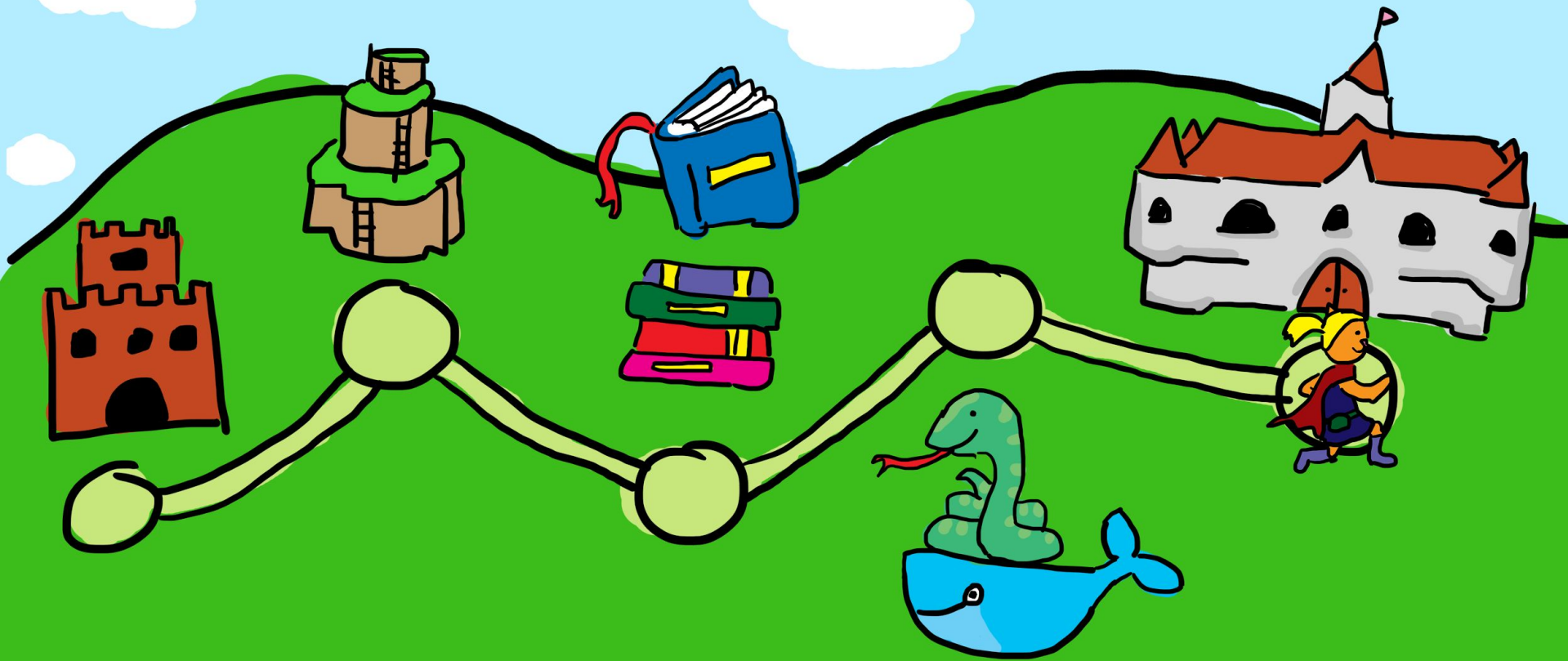
@bobcat.wilson



goo.gl/JkzmYJ

@mtomwing

TAKEAWAYS FOR DEV AND OPS



WHAT DO I DO WITH THIS?

- Developers
- Ops



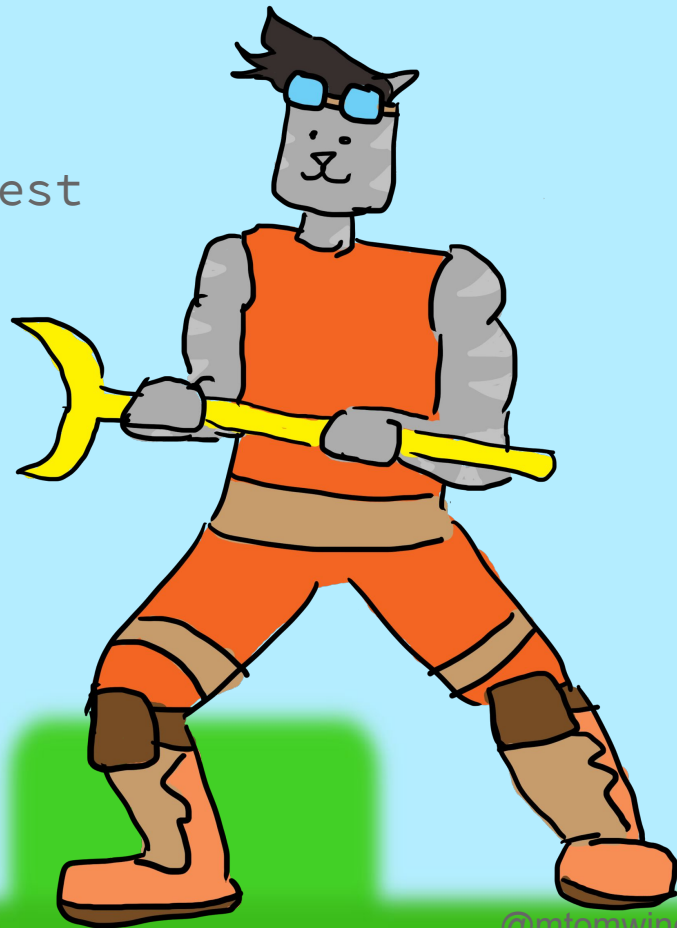
@bobcat.wilson

golang.wilson

@mtomwing

DEVELOPERS

- When to write tests and when not to
- Try some TDD: start with a system test



DEVELOPERS - INTRODUCE SYSTEM TESTS

- Add one system test to each piece of software you own
- Make sure tests can run:
 - With as little setup as possible
 - As quickly as possible
- Add the system tests to your CI



DEVELOPERS - ALREADY HAVE SYSTEM TESTS

- Do you need all the tests you have?
 - Can you replace with integration or unit tests?
 - How many retest functionality?
 - Can some of the tests be removed?
- Can the tests be faster?



Ops

- One off scripts: **don't need system tests**
- Scripts and automation that will be used in the future **need system tests**
 - What is one bare minimum system test you can add?
- Use automation to regularly run your tests
 - e.g. Travis CI



OPS - TESTS FOR TOOLS

- Tools that use services you can run:
 - Use something like `pytest + docker-py`
- Tools that use services you can't run (e.g. `AW$`):
 - Can you run a short system test, e.g. once per week?
 - Is it going to cost you a lot?
 - Make sure the tests clean up after themselves



OVERVIEW

github.com/keeppythonweird/pytest-dockerpy

@bobcatwilson

@mtomwing

