Hunting Bugs While Sleeping

Property-Based Testing with Continuous Integration

Paul Amazona
Developer
@whatevergeek

DataKind


Django Against the Dark Arts
@attacus_au


pyconau
INTERNATIONAL CONVENTION CENTRE
SYDNEY
pyconau


Container image security – Notary
Image repository
Digitally signs images to be published.
The Update Framework (TUF) underlays the tool to maintain up-to-date software.
Allows user to verify content that they download.
Honourable mentions go to Clair


Polylingualism
INTERNATIONAL CONVENTION CENTRE
pyconau


Generative Tests
"I SPEND A LOT OF TIME writing tests
I SHOULD WRITE A PROGRAM AUTOMATING IT!"
WORK
WRITING CODE
WORK ON ORIGINAL TASK
AUTOMATION TAKES OVER
FREE TIME
TIME
Using a library lets you skip the "ongoing development" panel from the original comic
3Ai
Basically automating automated testing!
Hypothesis can generate...
arguments to a test function
entire test programs!
INTERNATIONAL CONVENTION CENTRE
SYDNEY
pyconau

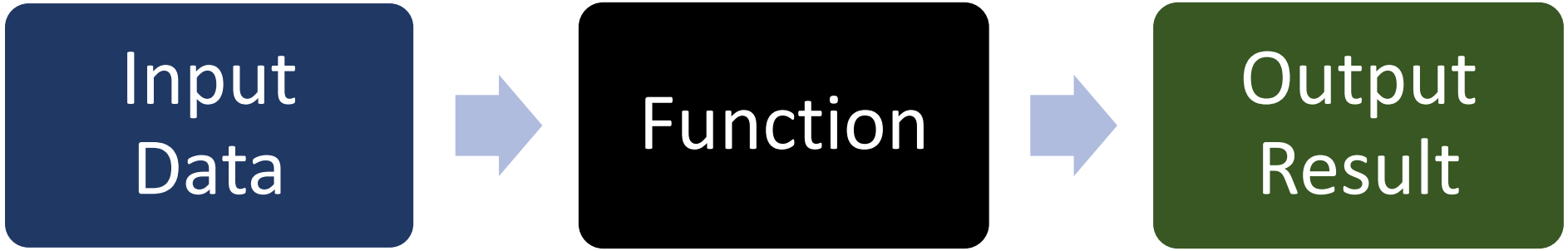

pyconau

Property-Based Testing with **Hypothesis** 

https://tinyurl.com/hypothesis-prezo

Property-Based Testing

A type of testing that asserts based on properties that describe the **relationship** between the **input** and **output** of the **function** being tested.



Testing the Multiply Function

```
def test_multiply():
    actual_output = multiply(2,3)
    expected_output = 2*3
    assert actual_output == expected_output
```

Testing the Multiply Function

Reimplementing

```
def test_multiply():
    actual_output = multiply(2,3)
    expected_output = 2*3
    assert actual_output == expected_output
```

Testing using Example Outputs

```
def test_example1():
    actual_output = multiply(2,3)
    expected_output = 6
    assert actual_output == expected_output

def test_example2():
    actual_output = multiply(4,5)
    expected_output = 20
    assert actual_output == expected_output
```

Parameterized Tests

```
@pytest.mark.parametrize("factor1, factor2, expected_output", [
    (2, 3, 6),
    (4, 5, 20),
])
def test_example2and3(factor1, factor2, expected_output):
    actual_output = multiply(factor1,factor2)
    assert actual_output == expected_output
```

Multiplication Properties

- **Commutative property**

  When two numbers are multiplied together, the product is the same regardless of the order of the multiplicands.
  For example 4 * 2 = 2 * 4

- **Associative Property**

  When three or more numbers are multiplied, the product is the same regardless of the grouping of the factors.
  For example (2 * 3) * 4 = 2 * (3 * 4)

- **Multiplicative Identity Property**

  The product of any number and one is that number.
  For example 5 * 1 = 5.

- **Distributive property**

  The sum of two numbers times a third number is equal to the sum of each addend times the third number.
  For example 4 * (6 + 3) = 4*6 + 4*3

Property-Based Tests (PBT)

with Predetermined Inputs

```
@pytest.mark.parametrize("factor1, factor2", [
    (2, 3),
    (4, 5),
])
def test_commutative_property(factor1, factor2):
    assert multiply(factor1,factor2) == multiply(factor2,factor1)
```

Property-Based Tests (PBT)

with Predetermined Inputs

```
@pytest.mark.parametrize("factor1", [2,4])
def test_identity_property(factor1):
    assert multiply(factor1,1) == factor1

@pytest.mark.parametrize("num1, num2, num3", [
    (2, 3, 4),
    (4, 5, 6),
])
def test_distributive_property(num1, num2, num3):
    assert multiply(num1,(num2 + num3)) == multiply(num1,num2) + multiply(num1,num3)
```

Property-Based Tests (PBT)

with Randomized Inputs

```
from hypothesis import given
import hypothesis.strategies as st

@given(st.integers(), st.integers())
def test_commutative_property(factor1, factor2):
    assert multiply(factor1,factor2) == multiply(factor2,factor1)
```

Property-Based Tests Libraries…

Language	PBT Libraries
python	hypothesis
.NET (C#, F#, etc)	FsCheck
haskell	quickcheck
java	junit-quickchek
javascript	fast-check
swift	SwiftCheck
scala	ScalaCheck

Some PBT library features…

More elaborate input criteria

```
@given(st.integers().filter(lambda x: x > 0), st.integers())
@example(factor1=2,factor2=3)
@settings(max_examples=200)
def test_commutative_property(factor1, factor2):
    assert multiply(factor1,factor2) == multiply(factor2,factor1)
```

Ability to use predetermined input

Adjust sample size

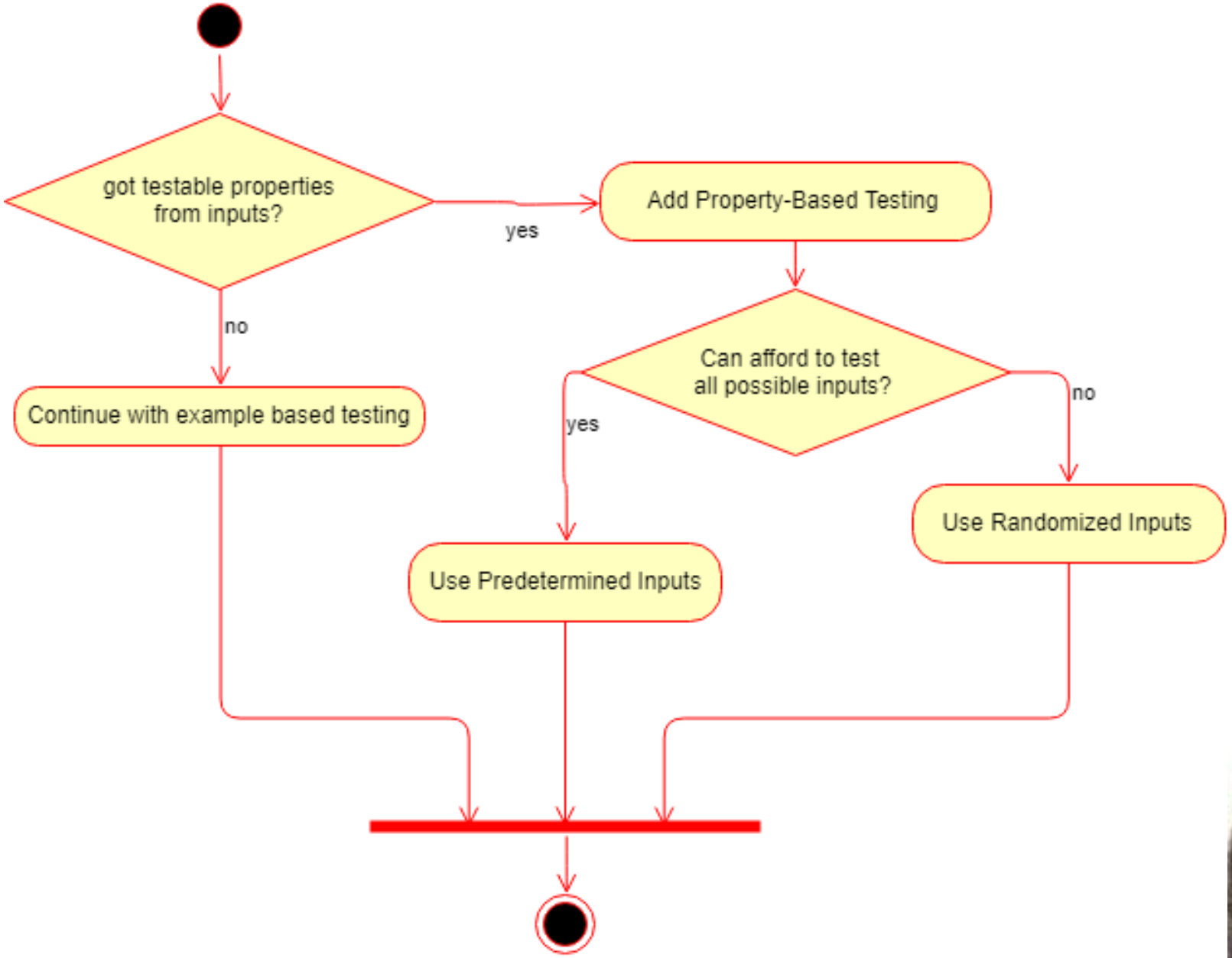
Diamond Kata

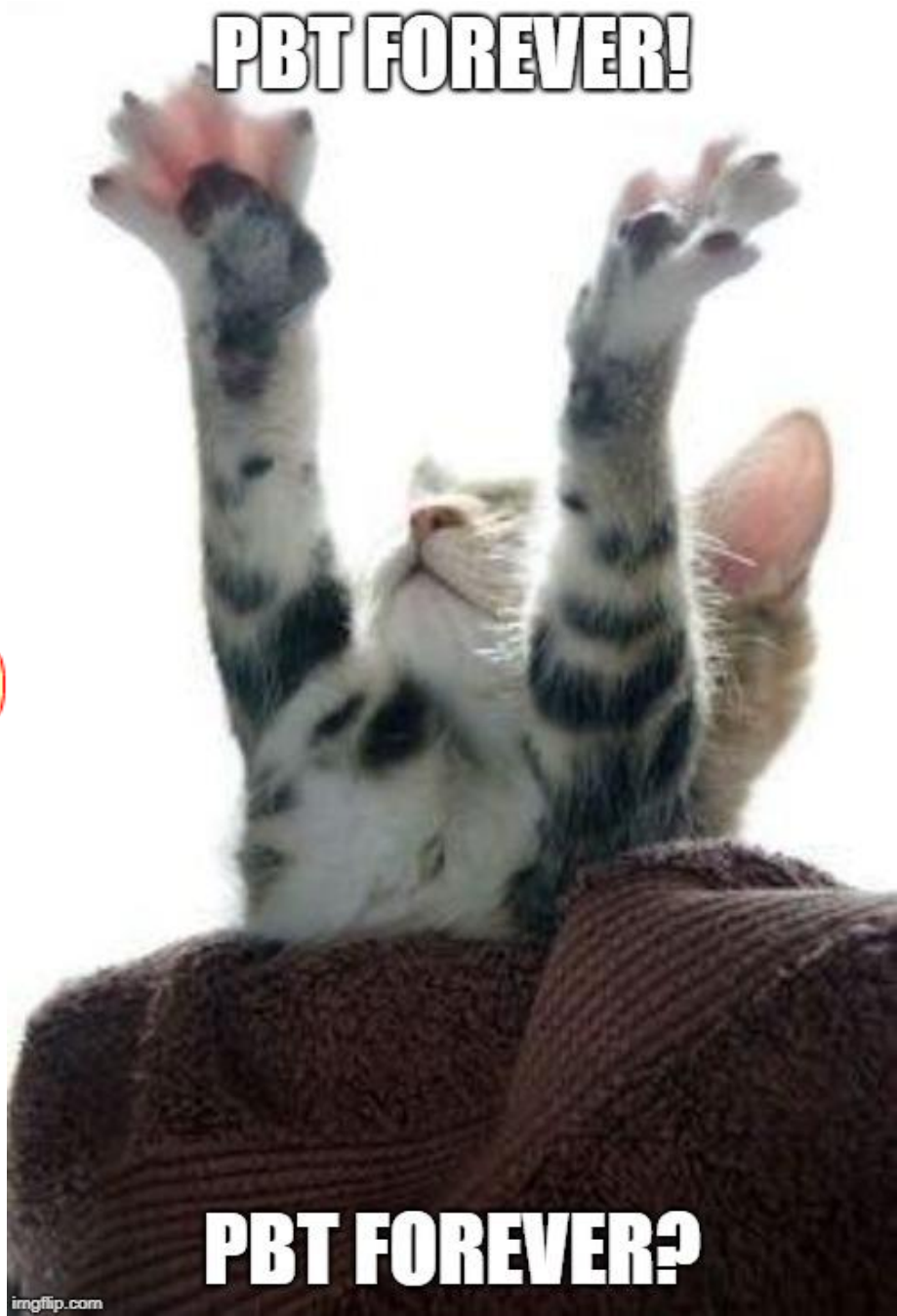
INPUT	OUTPUT
A	A

B	` A`
	`B B`
	` A`

C	`  A`
	` B B`
	`C   C`
	` B B`
	`  A`

D	`   A`
	`  B B`
	` C   C`
	`D     D`
	` C   C`
	`  B B`
	`   A`

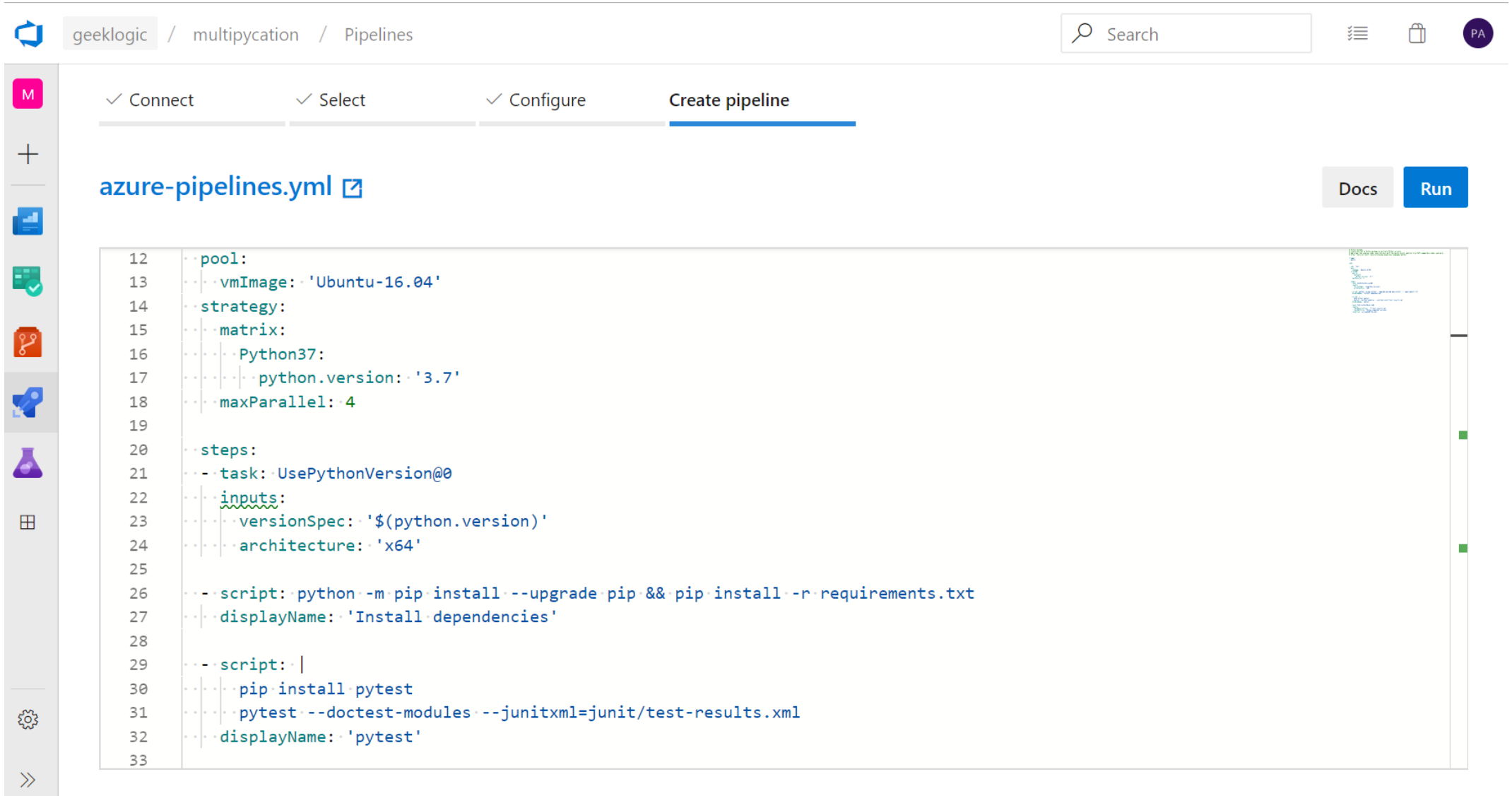

got testable properties from inputs?
yes
Add Property-Based Testing
no
Continue with example based testing
Can afford to test all possible inputs?
yes
no
Use Predetermined Inputs
Use Randomized Inputs


PBT FOREVER!
PBT FOREVER?
imgflip.com

Hunting Bugs with CI

Pipelines	Process	Characteristics
Normal CI	Build-> Tests with Predetermined Inputs	• Trigger: merge/PR/etc • More predictable duration • Purpose: Detect problems early
Bug Hunting CI	Build->Test with Randomized Inputs	• Trigger: Scheduled build (every hour, every day, etc). • Can take time depending on sample size of randomized inputs • Purpose: Hunt Bugs

DEMO: Bug Hunting CI Pipeline



DEMO: Bug Hunting CI Pipeline

Logs Summary Tests

Test Python37

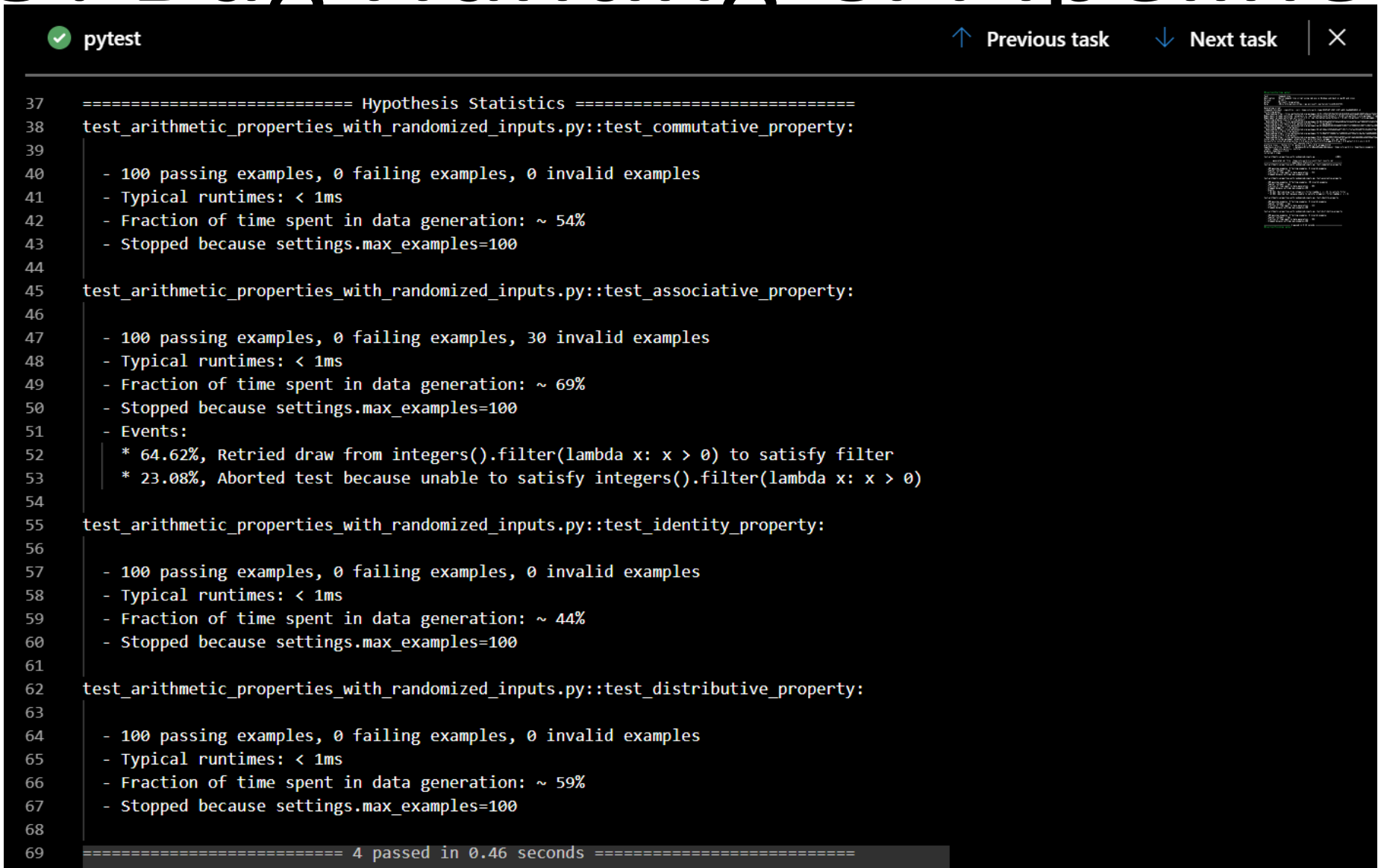
Started: 16/03/2019, 01:01:53

Pool: Hosted Ubuntu 1604 · Agent: Hosted Agent

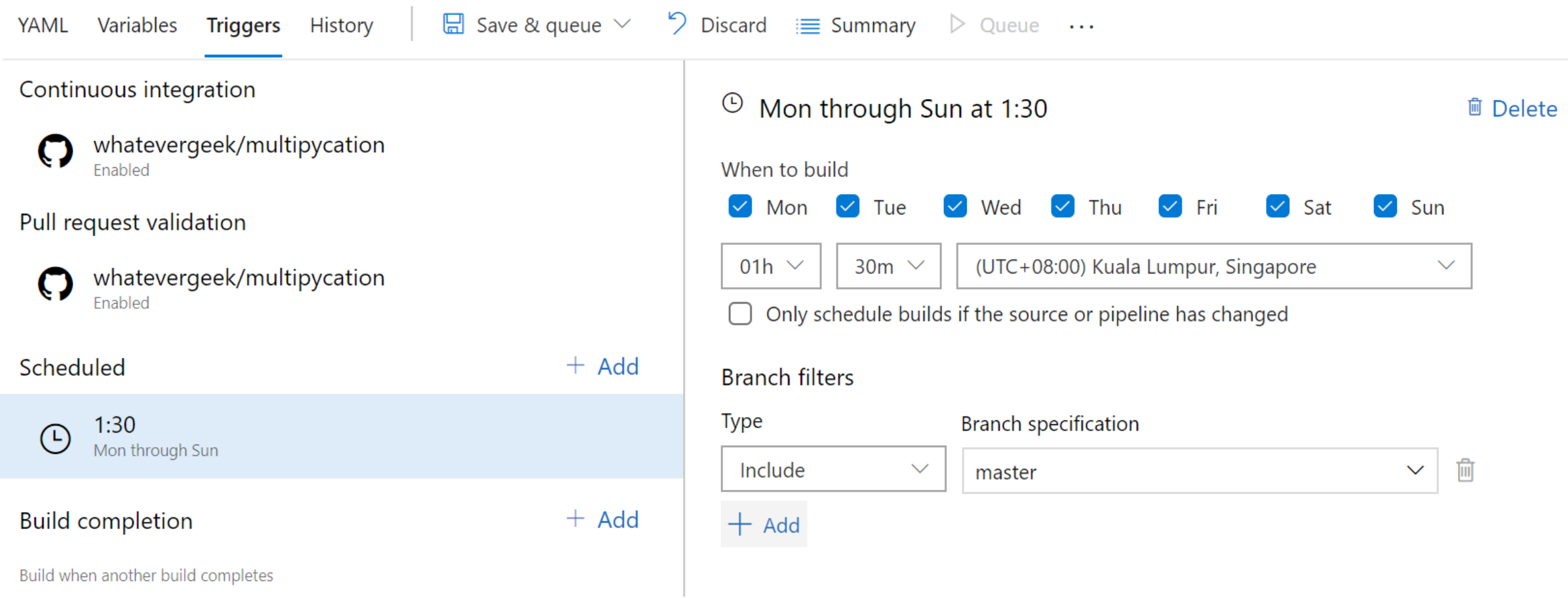
··· 33s

Initialize Agent · succeeded	<1s
Prepare job · succeeded	<1s
Initialize job · succeeded	1s
Checkout · succeeded	8s
UsePythonVersion · succeeded	1s
Install dependencies · succeeded	11s
pytest · succeeded	5s
PublishTestResults · succeeded	3s
Post-job: Checkout · succeeded	<1s
Finalize Job · succeeded	<1s

DEMO: Bug Hunting CI Pipeline



DEMO: Bug Hunting CI Pipeline



https://github.com/whatevergeek/multipycation

Summary and Links

- Testing using Example Outputs
- Parameterized Tests
- Property-based Tests with Predetermined Inputs
- Property-based Tests with Randomized Inputs
- Bug Hunting CI Pipeline

Paul Amazona

@whatevergeek

https://tinyurl.com/hypothesis-prezo
https://github.com/whatevergeek/multipycation
https://gitlab.com/whatevergeek/multipycation-gl-demo