Testeando con Go

DevFest Sevilla 2025

Nicolás Palumbo

Quién soy?

Golang Sevilla

Nicolás Palumbo

Senior Software Engineer | New Relic

devnull talks

Agenda

No alarms and no surprises.

Agenda

- ¿Por qué?
- Pirámide de Tests
- Kata
- Go Test!
- Test Doubles
- Más dependencias!
- Integration Tests
- Conclusión
- Gracias, Pedir!
- Q / A

¿Por qué?

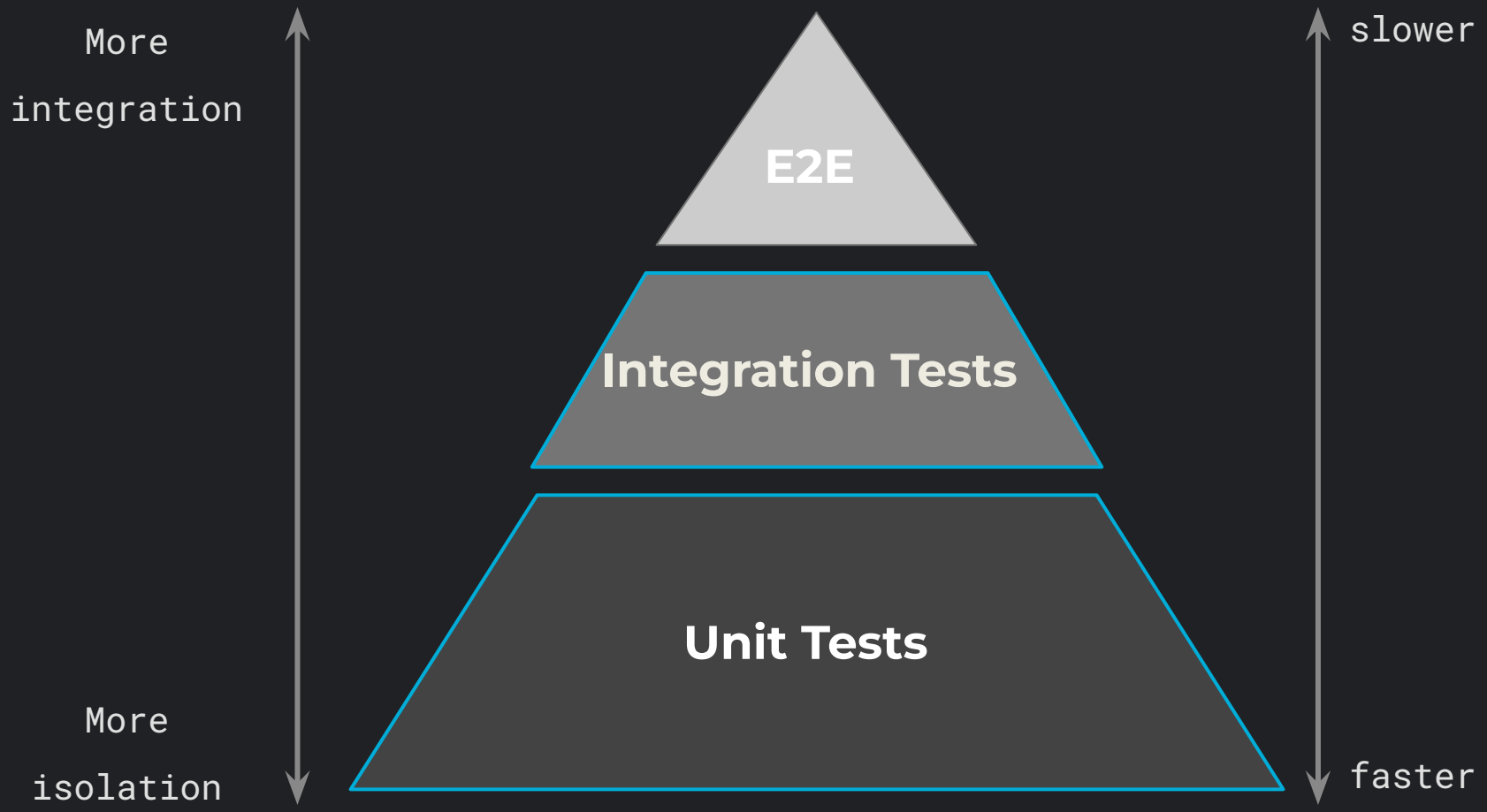
Why Oh Why?

Pirámide de Tests

$$V = \frac{na^2h}{12}\cot\left(\frac{\pi}{n}\right)$$

Piramide de Tests

Test Pyramid



Kata

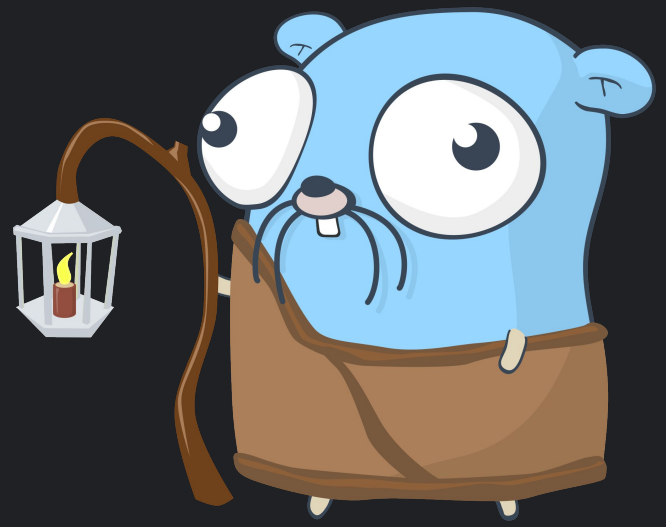
(型 or 形)

Kata

第一 Salud & Nivel

Todos los Personajes, al ser creados, tienen:

- **Salud**, que comienza en 1000
- **Nivel**, que comienza en 1
- Pueden estar **Vivos o Muertos**, empezando Vivos



Kata

第二 Inflingir daño

Los Personajes pueden Infligir Daño a otros Personajes.

- El **Daño** se resta de la Salud.
- Cuando el daño recibido supera la Salud actual, la Salud se convierte en 0 y el personaje muere.

Go Test!

1, 2, 3… Probando

Go Test!

Benchmark Testing

Para identificar problemas de rendimiento.

```go
package charlakata_test

import (
	"charlakata"
	"testing"
)

func BenchmarkPlayerRPG_Attack(b *testing.B) {
   for b.Loop() {

		// Completar aquí!
   }
}
```

Go Test!

Fuzzy Testing

Para descubrir edge cases.

```go
package charlakata_test

import (
     "charlakata"
     "testing"
)
// Fuzz test
func Fuzz_Damage_Boundaries(f *testing.F) {
   f.Add(1000, 1) // Seed corpus

   // Fuzz target
   f.Fuzz(func(t *testing.T, initialHealth, amount int) {

       if targetPlayer.GetHealth() < 0 {
            t.Errorf("got %d for %d", target.GetHealth(), amount)
       }
   })
}
```

Go Test!

Coverage

```
➜  go test -cover ./... -coverpkg=charlakata -coverprofile=coverage.out
ok      charlakata      0.209s  coverage: 100.0% of statements in charlakata

➜  go tool cover -func=coverage.out
charlakata/player.go:16:        TakeDamage      100.0%
charlakata/player.go:20:        DealDamage      100.0%
charlakata/player.go:30:        GetHealth       100.0%
charlakata/player.go:34:        GetLevel        100.0%
charlakata/player.go:38:        IsAlive         100.0%
charlakata/player.go:42:        NewRpgPlayer    100.0%
total:                          (statements)    100.0%

➜ go tool cover -html=coverage.out
```

Test Doubles

doppelgänger

Test Doubles

Stubs

“Los **Stubs** proveen respuestas predefinidas a las peticiones que se hacen durante el test.”

```
type StubPlayer struct{}

func (s *StubPlayer) GetLevel() int {
     return 1
}

func (s *StubPlayer) IsAlive() bool {
     return true
}
```

Test Doubles

Mocks

"Los **Mocks** están preprogramados con expectativas que forman una especificación de las llamadas que se espera que reciban."

```
// Given
targetPlayer.EXPECT().IsAlive().Return
(expected.alive).Times(1)

// When
// ...

// Then
targetPlayer.AssertExpectations(t)
```

Test Doubles

Spies

"Los **Spies** son stubs que también registran cierta información referida a la forma en cómo fueron llamados."

```
type SpyPlayer struct{}


func (s *SpyPlayer) TakeDamage(health int) {
     s.TakeDamageCallCount++
}
```

Más dependencias!

Fatto trenta, facciamo trentuno.

Más dependencias!

Testify

- Los mensajes al describir un error son más amigables y legibles.
- El código de los assertions es más legible.
- Se pueden anotar los assertions con un mensaje.

```
assert.Equal(t, 1000, player.GetHealth())

assert.Equal(t, 1, player.GetLevel())

assert.True(t, player.IsAlive(), "Player should be alive")
```

Más dependencias!

Ginkgo & Gomega

- Ginkgo: Testing Framework.
- Gomega: Matchers.
- Proveen un DSL para escribir tests.

```
It("should start with the correct initial parameters", func() {
    player := charlakata.NewRpgPlayer(1000, 1)

    Expect(player.GetHealth()).To(Equal(1000))
    Expect(player.GetLevel()).To(Equal(1))
    Expect(player.IsAlive()).To(BeTrue())
})
```

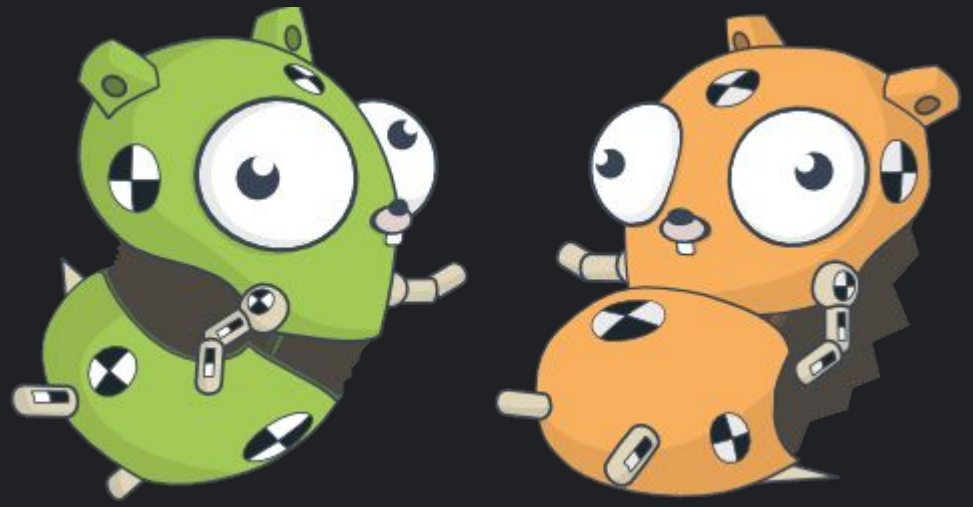
Integration Tests

$$(\int_0^b f(x)dx).$$

Integration Tests

Integration Tests

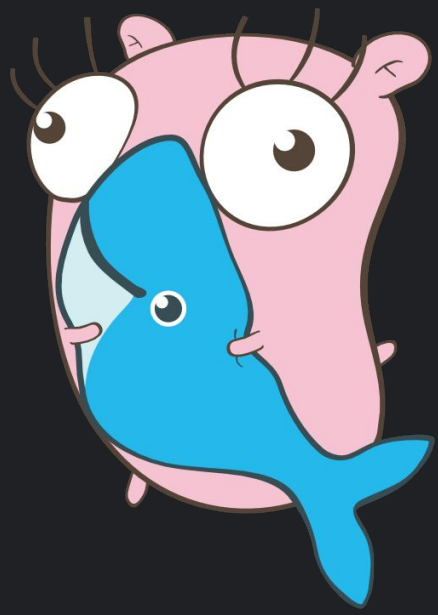
Para identificar los problemas y/o errores que surgen al probar la interacción entre dos o más módulos.



Integration Tests

Testcontainers

- Usa servicios reales que corren en docker containers.
- Se pueden escribir tests que dependen de los mismos servicios que usamos en producción, sin usar mocks o servicios en memoria.



Conclusión

C'est fini

Gracias, Pedir!

Gratias vobis ago

Gracias, Pedir!

Venir a la charla de /dev/null talks

dev/null
talks

Q / A

Въпрос / Отговор

Bonus

Μπόνους

Bonus

Golang Sevilla:

https://www.meetup.com/golang-sevilla/

dev/null talks:

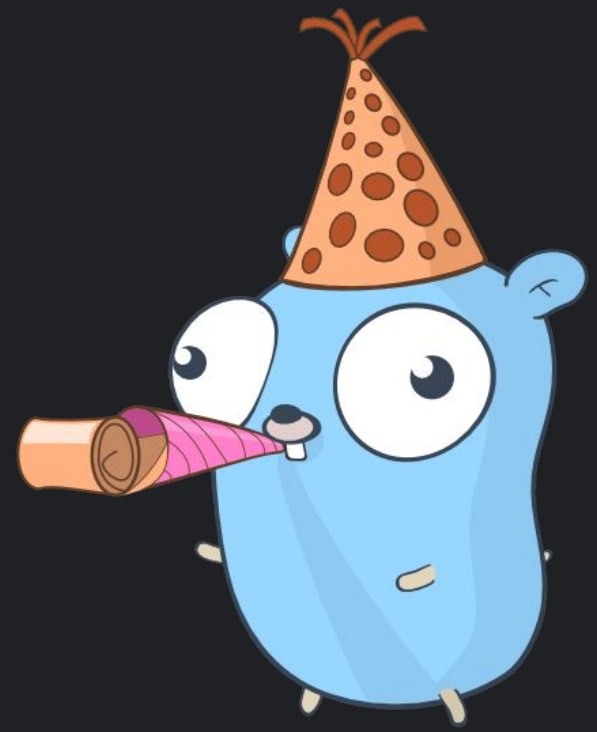
https://www.meetup.com/dev-null/

LinkedIn:

https://www.linkedin.com/in/nicol%C3%A1s-palumbo-9372615/

Gophers:

https://github.com/egonelbre/gophers



Slides originales:

https://docs.google.com/presentation/d/1sXTPhZUY5gDld7kpLMlvnPxa2Sps0ksvNzrrk58Uw9A/view

Contenido de Martin Fowler sobre la Pirámide de Tests y Test Doubles:

https://martinfowler.com/(bliki/TestDouble.html|articles/practical-test-pyramid.html)