Climbing the Testing Pyramid: From Real Service to Interface Mocks in Go

Naveen Ramanathan

Software Engineer @ JPMorgan

Founder golangbot.com

Opinions expressed are solely my own and do not express the views of my employer.

This talk is about...

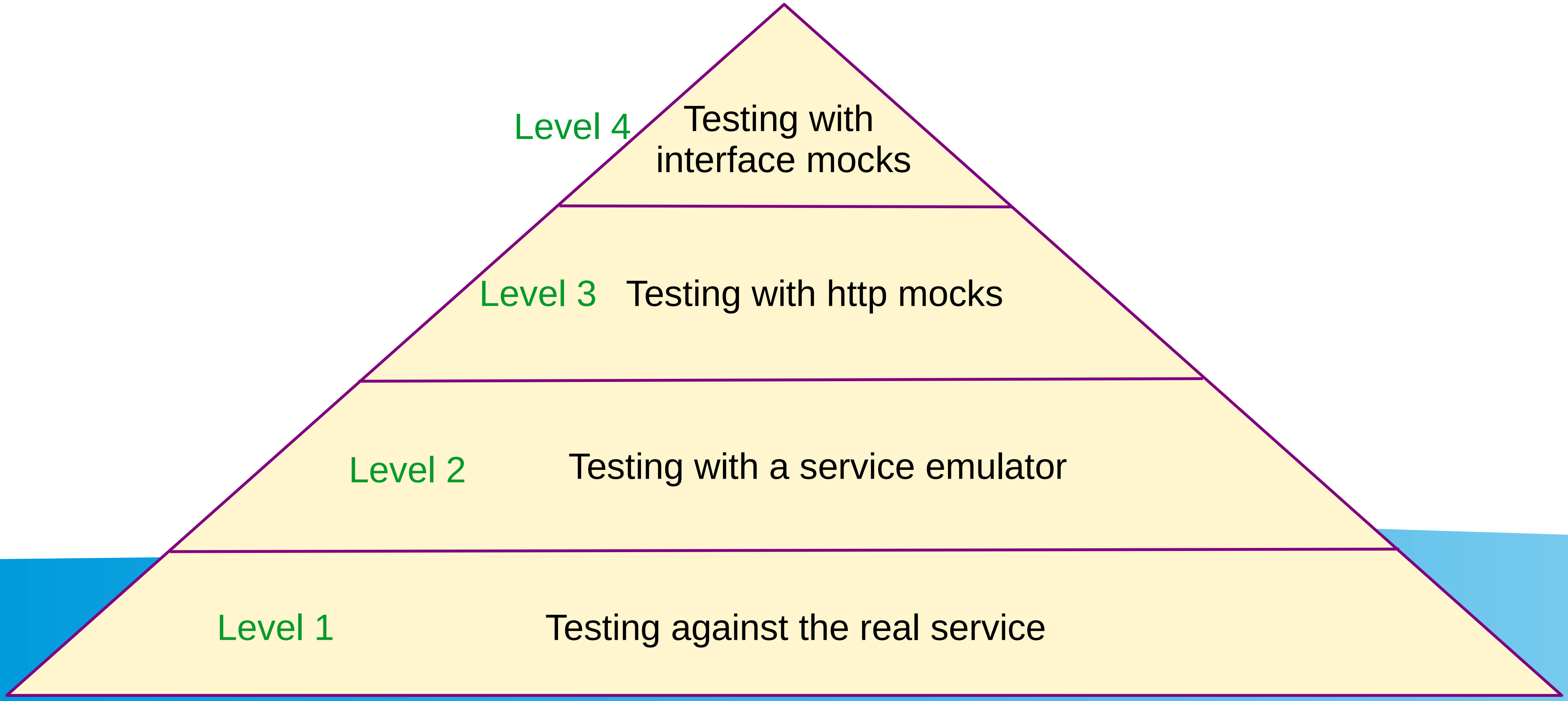
- Challenges when testing code with external (cloud) dependencies
- Different strategies to test code with cloud dependencies

Why is testing code with cloud dependencies hard?

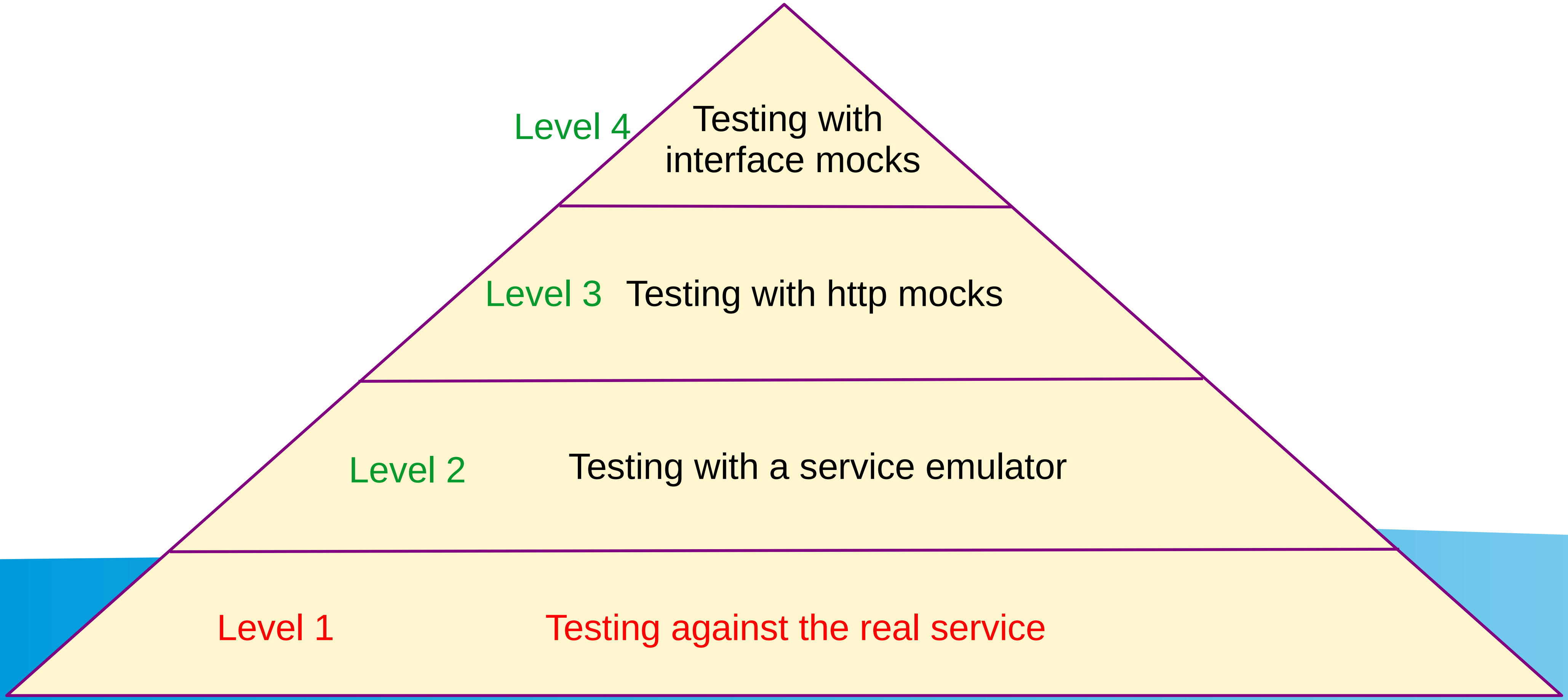
Why is testing code with cloud dependencies hard?

- Testing failure scenarios is hard
- Access restrictions
- High costs (failure to teardown resources after test)

Testing Pyramid


Level 4
Testing with interface mocks
Level 3
Testing with http mocks
Level 2
Testing with a service emulator
Level 1
Testing against the real service

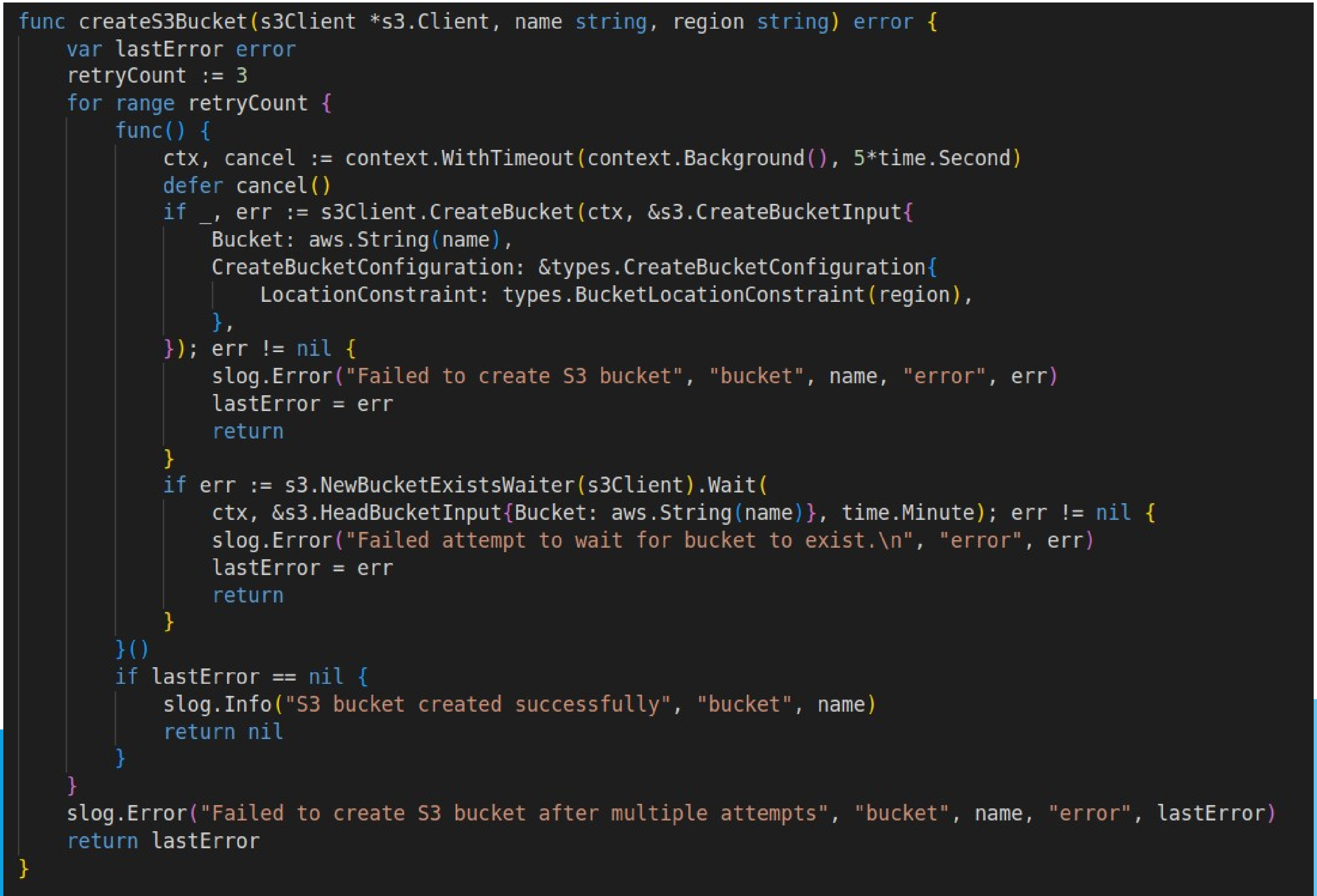
Level 1: Testing against the real service


Level 4
Testing with interface mocks
Level 3
Testing with http mocks
Level 2
Testing with a service emulator
Level 1
Testing against the real service

Function under test

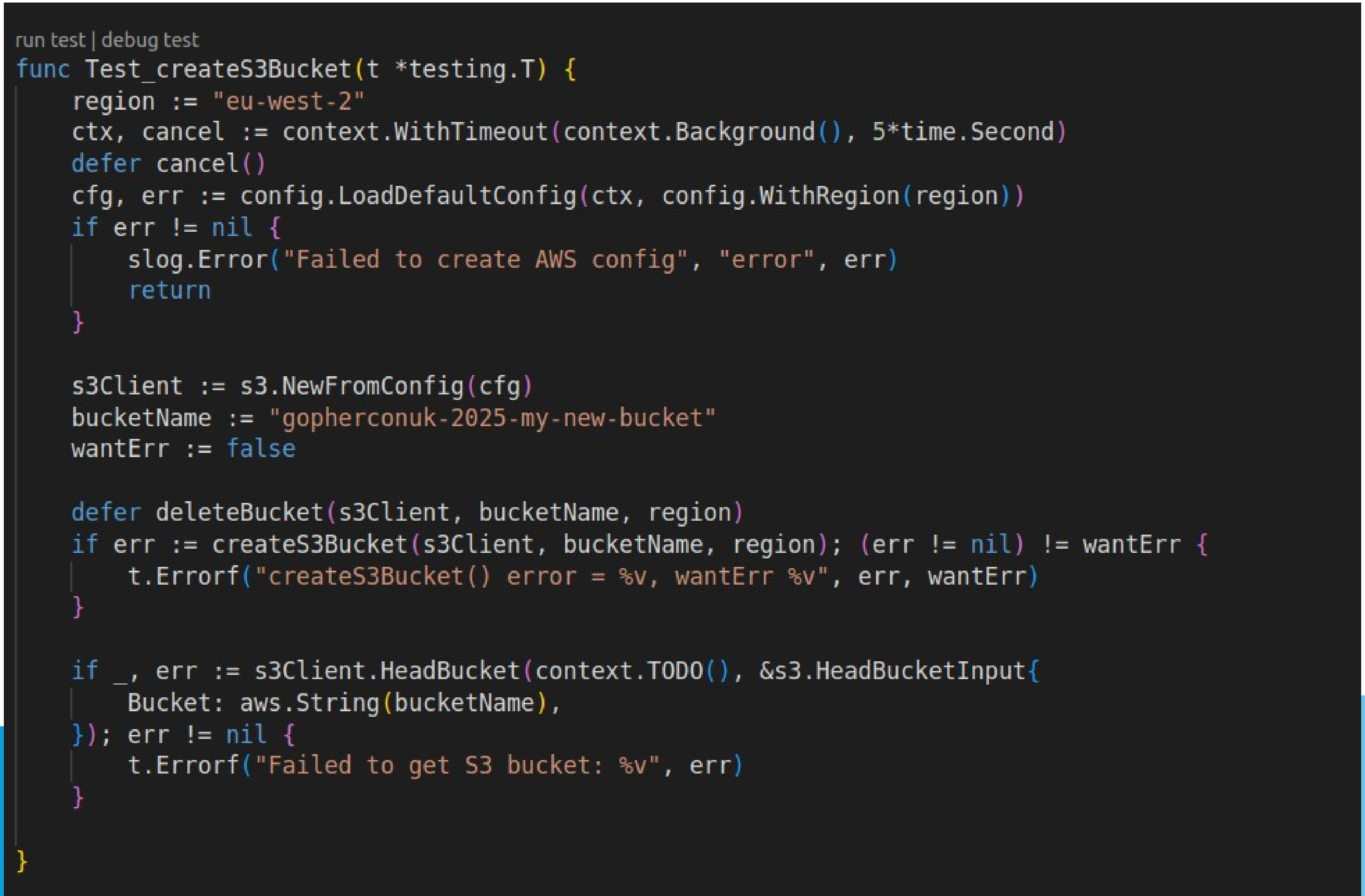
Function to create a S3 bucket

Create S3 bucket Function



```go
func createS3Bucket(s3Client *s3.Client, name string, region string) error {
    var lastError error
    retryCount := 3
    for range retryCount {
        func() {
            ctx, cancel := context.WithTimeout(context.Background(), 5*time.Second)
            defer cancel()
            if _, err := s3Client.CreateBucket(ctx, &s3.CreateBucketInput{
                Bucket: aws.String(name),
                CreateBucketConfiguration: &types.CreateBucketConfiguration{
                    LocationConstraint: types.BucketLocationConstraint(region),
                },
            }); err != nil {
                slog.Error("Failed to create S3 bucket", "bucket", name, "error", err)
                lastError = err
                return
            }
            if err := s3.NewBucketExistsWaiter(s3Client).Wait(
                ctx, &s3.HeadBucketInput{Bucket: aws.String(name)}, time.Minute); err != nil {
                slog.Error("Failed attempt to wait for bucket to exist.\n", "error", err)
                lastError = err
                return
            }
        }()
        if lastError == nil {
            slog.Info("S3 bucket created successfully", "bucket", name)
            return nil
        }
    }
    slog.Error("Failed to create S3 bucket after multiple attempts", "bucket", name, "error", lastError)
    return lastError
}
```

Test happy path

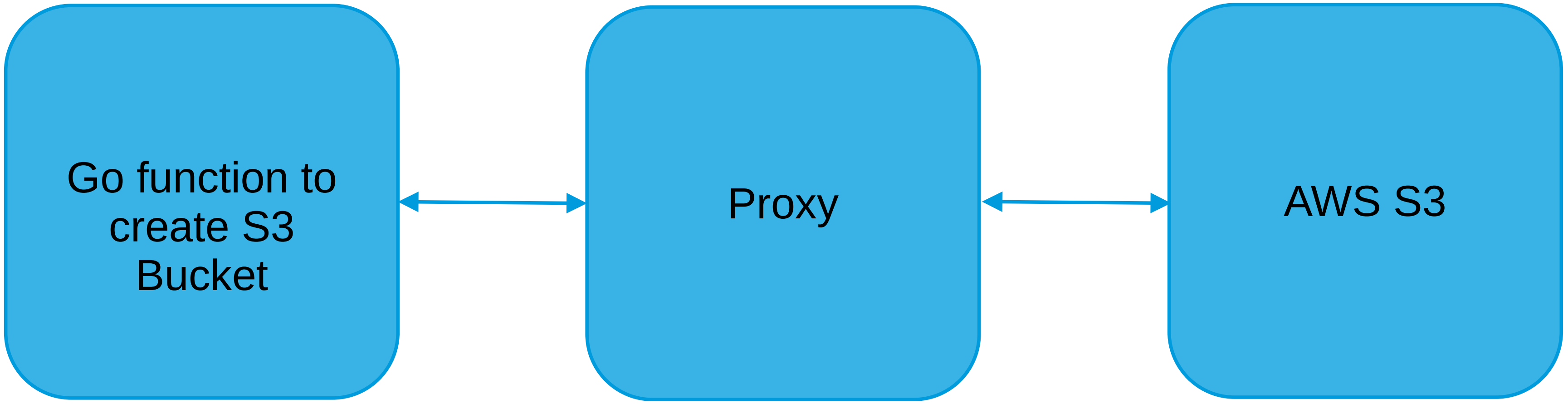


run test | debug test

```go
func Test_createS3Bucket(t *testing.T) {
    region := "eu-west-2"
    ctx, cancel := context.WithTimeout(context.Background(), 5*time.Second)
    defer cancel()
    cfg, err := config.LoadDefaultConfig(ctx, config.WithRegion(region))
    if err != nil {
        slog.Error("Failed to create AWS config", "error", err)
        return
    }

    s3Client := s3.NewFromConfig(cfg)
    bucketName := "gopherconuk-2025-my-new-bucket"
    wantErr := false

    defer deleteBucket(s3Client, bucketName, region)
    if err := createS3Bucket(s3Client, bucketName, region); (err != nil) != wantErr {
        t.Errorf("createS3Bucket() error = %v, wantErr %v", err, wantErr)
    }

    if _, err := s3Client.HeadBucket(context.TODO(), &s3.HeadBucketInput{
        Bucket: aws.String(bucketName),
    }); err != nil {
        t.Errorf("Failed to get S3 bucket: %v", err)
    }

}
```

How to test the unhappy path?

Testing unhappy path

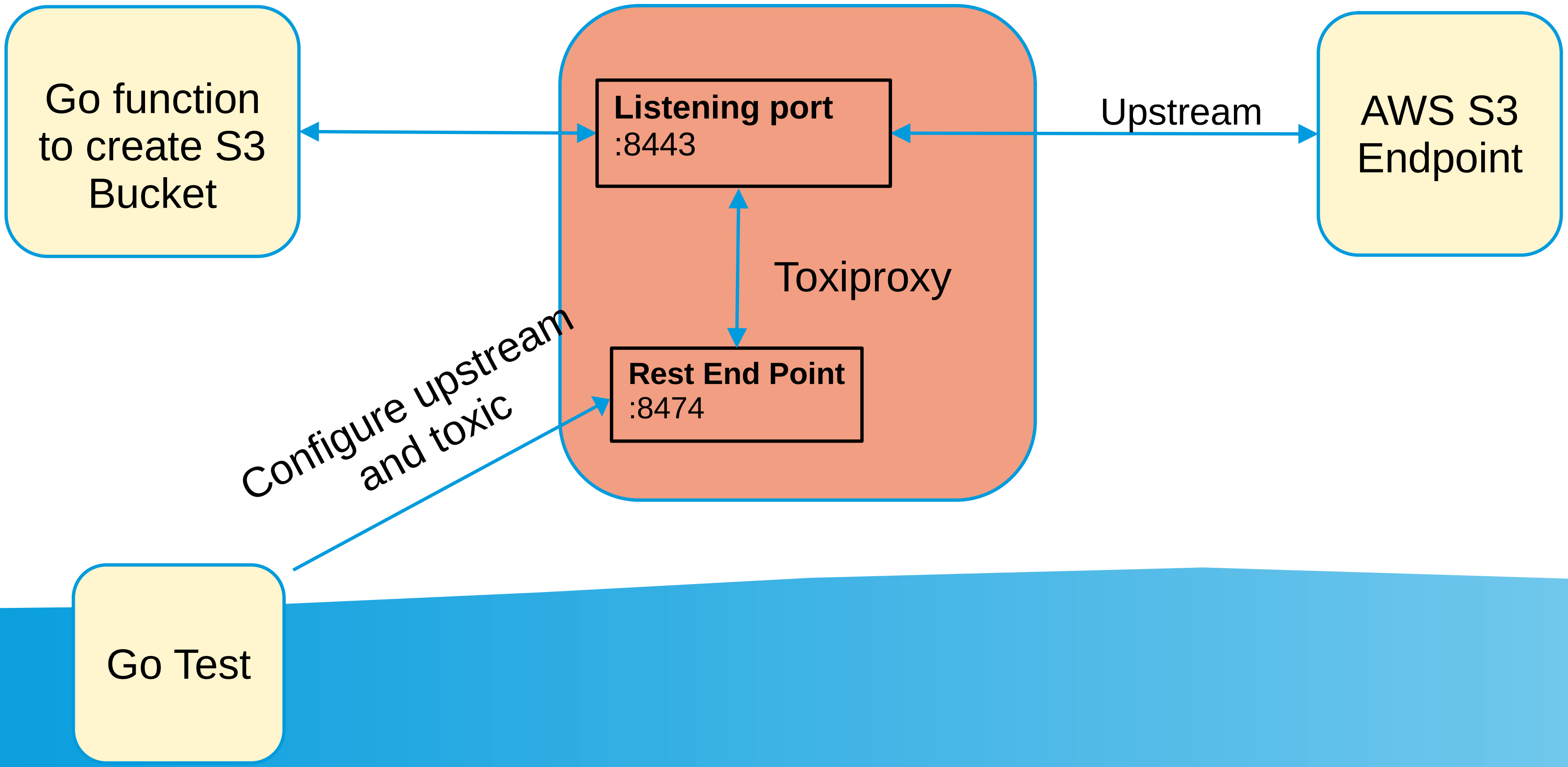


Proxy creates network partition

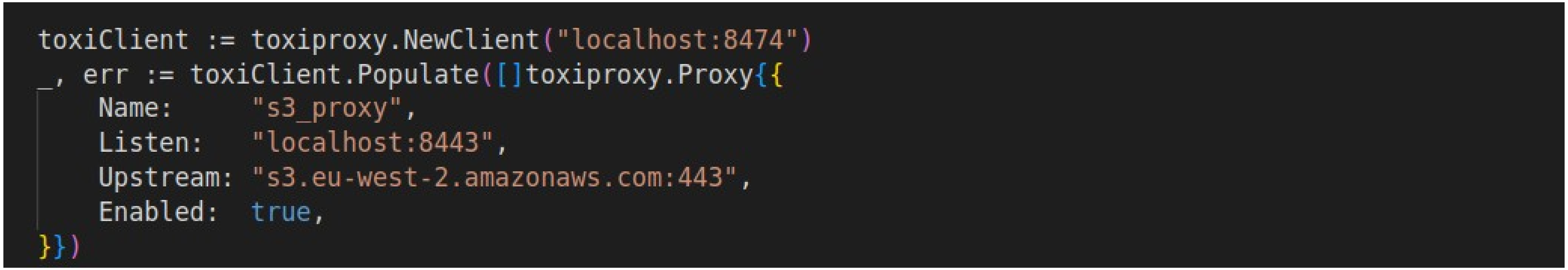
Toxiproxy

- TCP proxy written in Go for simulating network conditions
- Runs as a container
- Can be configured using REST APIs
- Different network conditions can be simulated using **toxics**

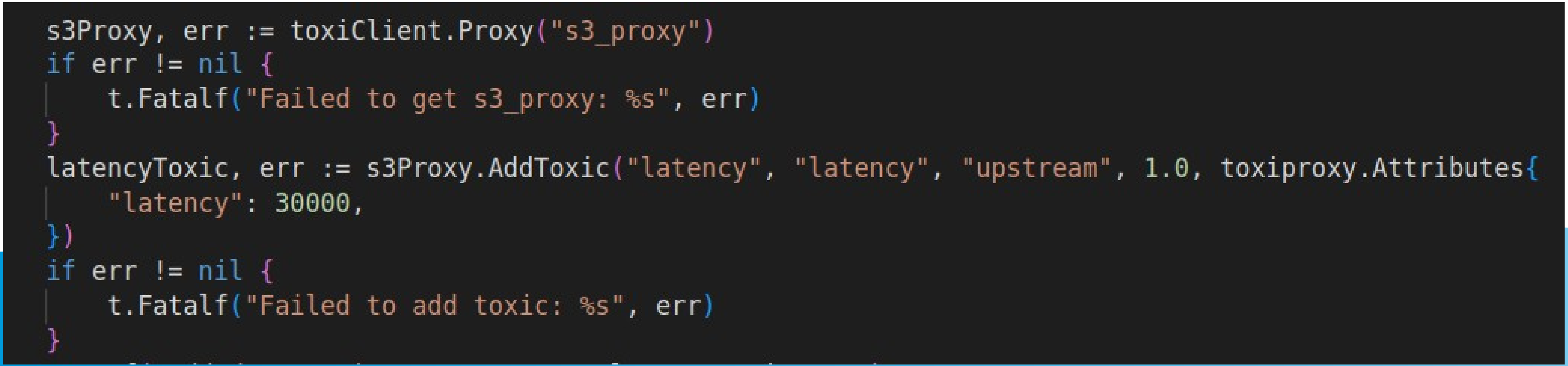
Test setup using Toxiproxy



Configure Toxiproxy

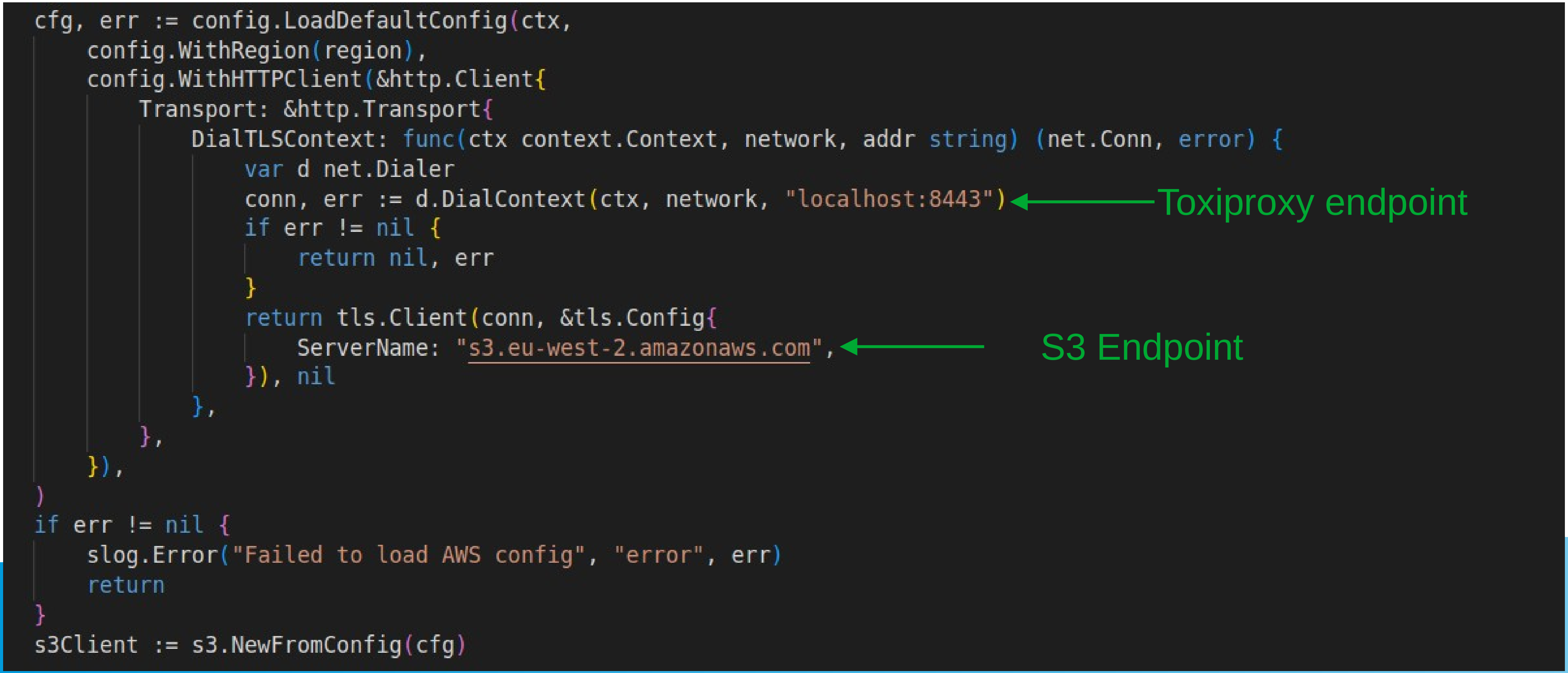


```go
toxiClient := toxiproxy.NewClient("localhost:8474")
_, err := toxiClient.Populate([]toxiproxy.Proxy{{
    Name:     "s3_proxy",
    Listen:   "localhost:8443",
    Upstream: "s3.eu-west-2.amazonaws.com:443",
    Enabled:  true,
}})
```



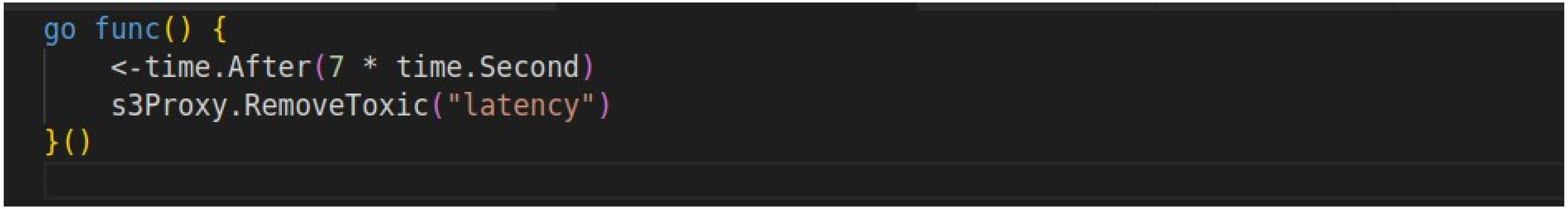
```go
s3Proxy, err := toxiClient.Proxy("s3_proxy")
if err != nil {
    t.Fatalf("Failed to get s3_proxy: %s", err)
}
latencyToxic, err := s3Proxy.AddToxic("latency", "latency", "upstream", 1.0, toxiproxy.Attributes{
    "latency": 30000,
})
if err != nil {
    t.Fatalf("Failed to add toxic: %s", err)
}
```

Configure AWS client to use Toxiproxy



```go
cfg, err := config.LoadDefaultConfig(ctx,
    config.WithRegion(region),
    config.WithHTTPClient(&http.Client{
        Transport: &http.Transport{
            DialTLSContext: func(ctx context.Context, network, addr string) (net.Conn, error) {
                var d net.Dialer
                conn, err := d.DialContext(ctx, network, "localhost:8443")
                if err != nil {
                    return nil, err
                }
                return tls.Client(conn, &tls.Config{
                    ServerName: "s3.eu-west-2.amazonaws.com",
                }), nil
            },
        },
    }),
)
if err != nil {
    slog.Error("Failed to load AWS config", "error", err)
    return
}
s3Client := s3.NewFromConfig(cfg)
```

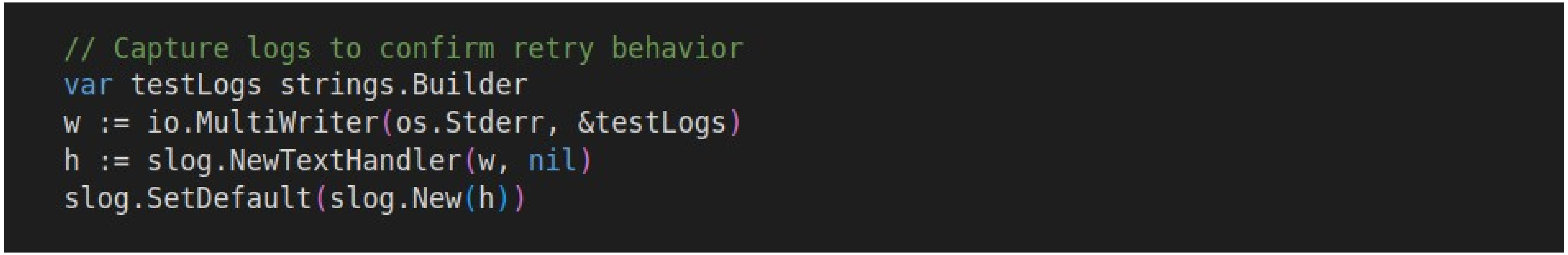
Remove toxic to enable successful retry



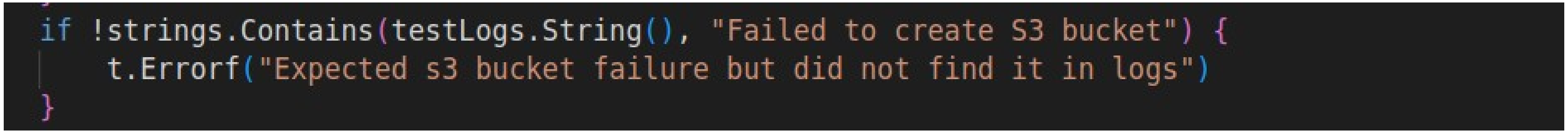
```go
go func() {
    <-time.After(7 * time.Second)
    s3Proxy.RemoveToxic("latency")
}()
```

How to validate retry ?

Monitoring logs



```go
// Capture logs to confirm retry behavior
var testLogs strings.Builder
w := io.MultiWriter(os.Stderr, &testLogs)
h := slog.NewTextHandler(w, nil)
slog.SetDefault(slog.New(h))
```



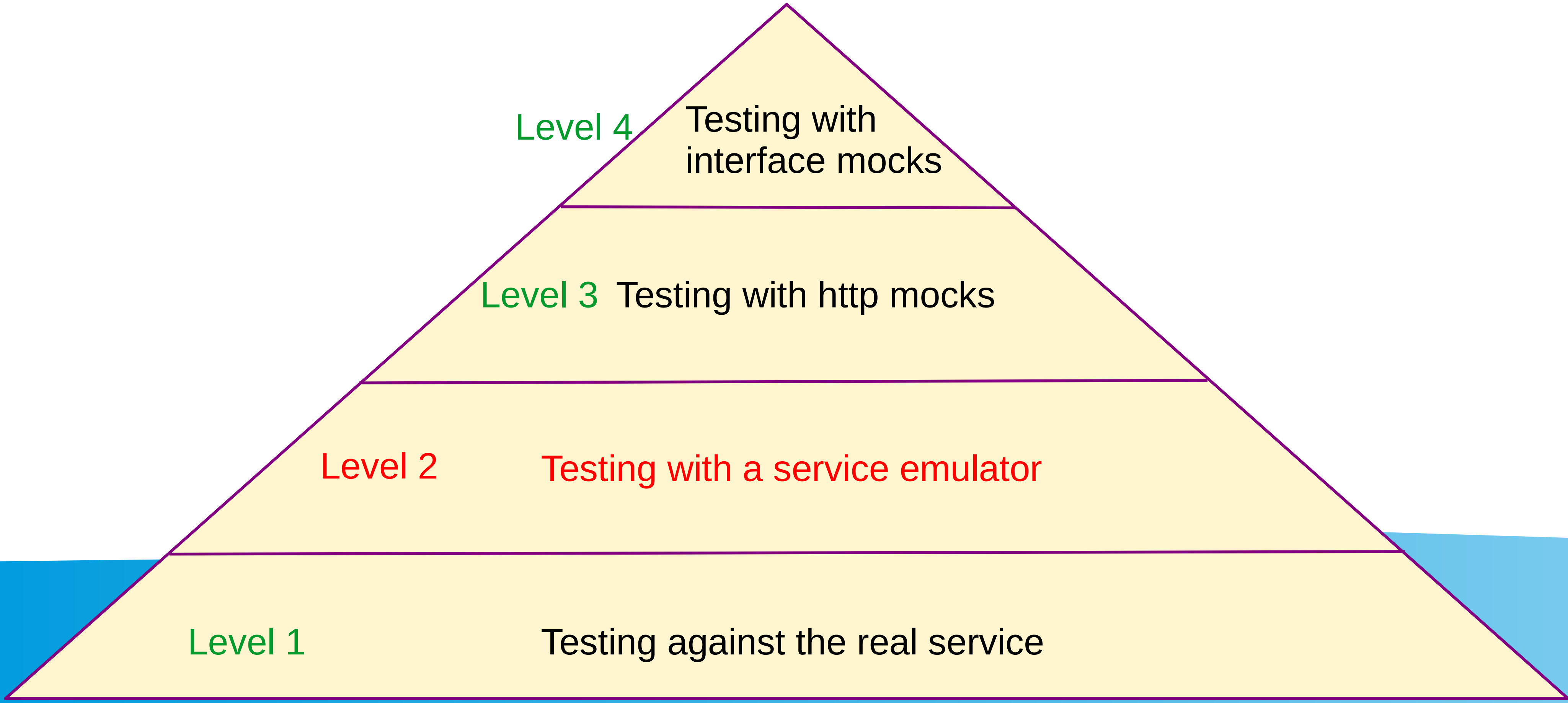
```go
if !strings.Contains(testLogs.String(), "Failed to create S3 bucket") {
    t.Errorf("Expected s3 bucket failure but did not find it in logs")
}
```

Why testing againt the real cloud service is not possible always?

Why testing againt the real cloud service is not possible always?

- Access restrictions (lack of permissions), quite common in regulated environments
- Cost

Level 2: Testing with a service emulator


Level 4
Testing with interface mocks
Level 3
Testing with http mocks
Level 2
Testing with a service emulator
Level 1
Testing against the real service

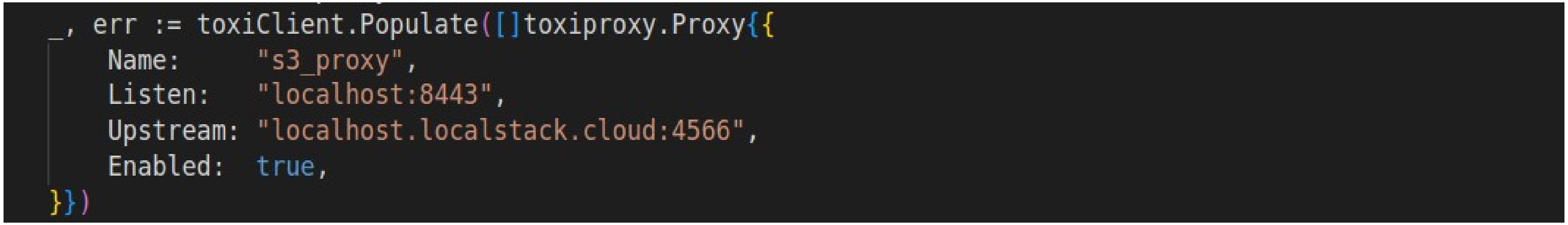
Emulators

- Provide mocked implementations which emulate the external dependency
- Runs in a container

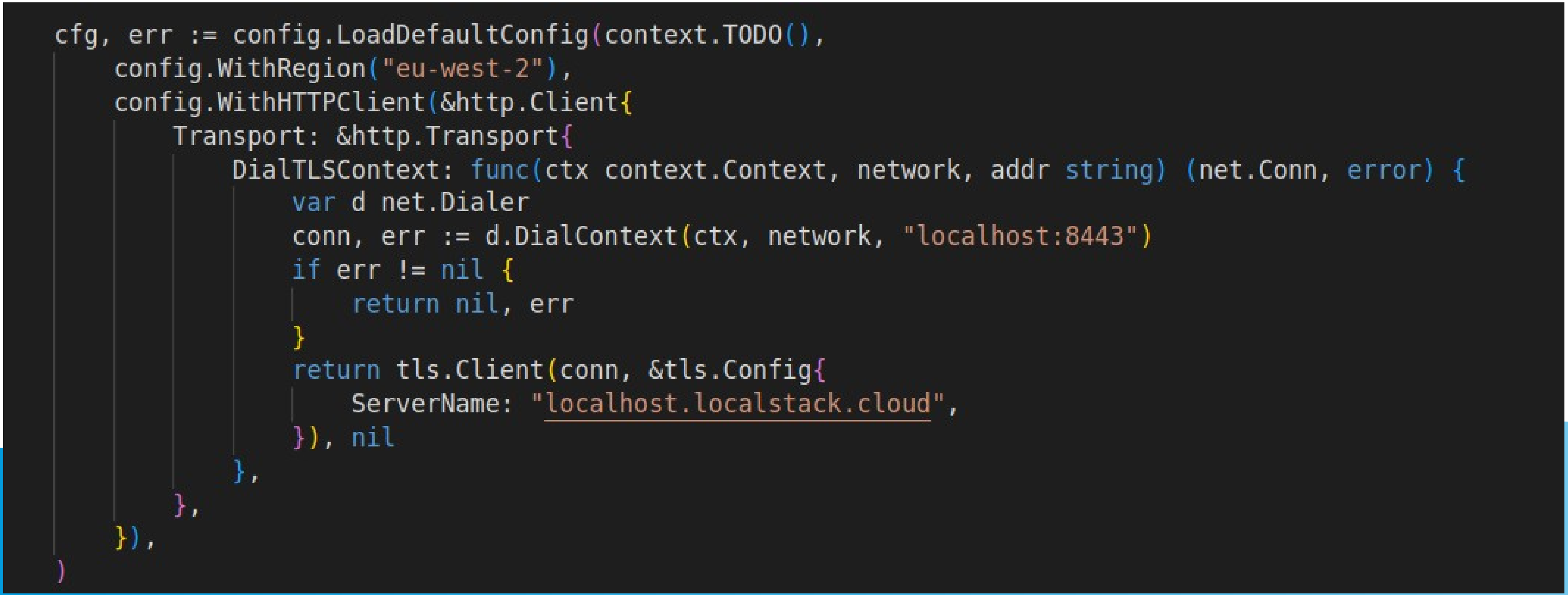
LocalStack

- AWS cloud service emulator that runs in a single container
- Supports a number of AWS services like AWS EC2, S3, DynamoDB,….
- Works with AWS CLI

LocalStack Configuration



```go
_, err := toxiClient.Populate([]toxiproxy.Proxy{{
    Name:     "s3_proxy",
    Listen:   "localhost:8443",
    Upstream: "localhost.localstack.cloud:4566",
    Enabled:  true,
}})
```



```go
cfg, err := config.LoadDefaultConfig(context.TODO(),
    config.WithRegion("eu-west-2"),
    config.WithHTTPClient(&http.Client{
        Transport: &http.Transport{
            DialTLSContext: func(ctx context.Context, network, addr string) (net.Conn, error) {
                var d net.Dialer
                conn, err := d.DialContext(ctx, network, "localhost:8443")
                if err != nil {
                    return nil, err
                }
                return tls.Client(conn, &tls.Config{
                    ServerName: "localhost.localstack.cloud",
                }), nil
            },
        },
    }),
)
```

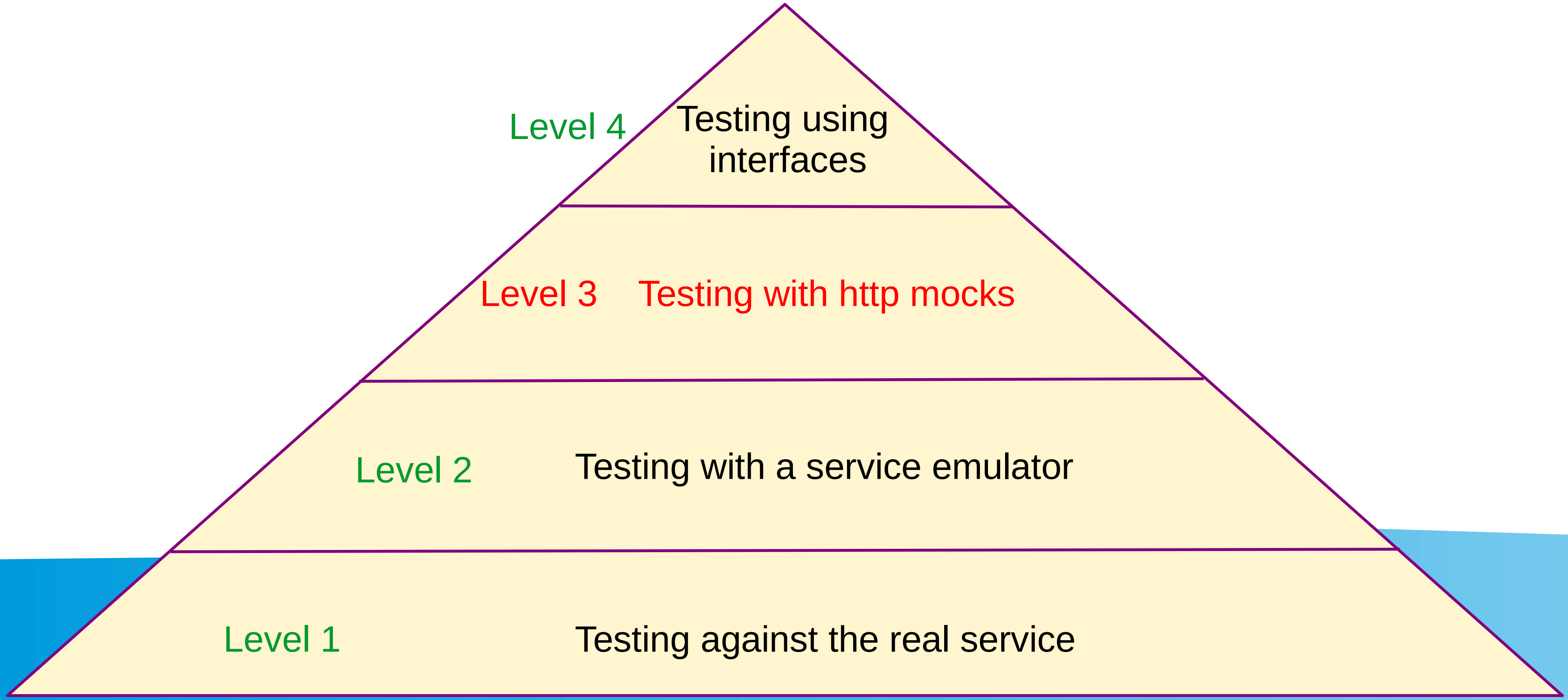
LocalStack DNS Resolution

- Can be accessed using the domain name localhost.localstack.cloud
- Offers a valid SSL cert(not self signed)

The barriers to testing with emulators

- Not 100% parity with the cloud dependency
- Emulators are not available for all cloud providers

Level 3: Testing with http mocks



HTTP Mocks

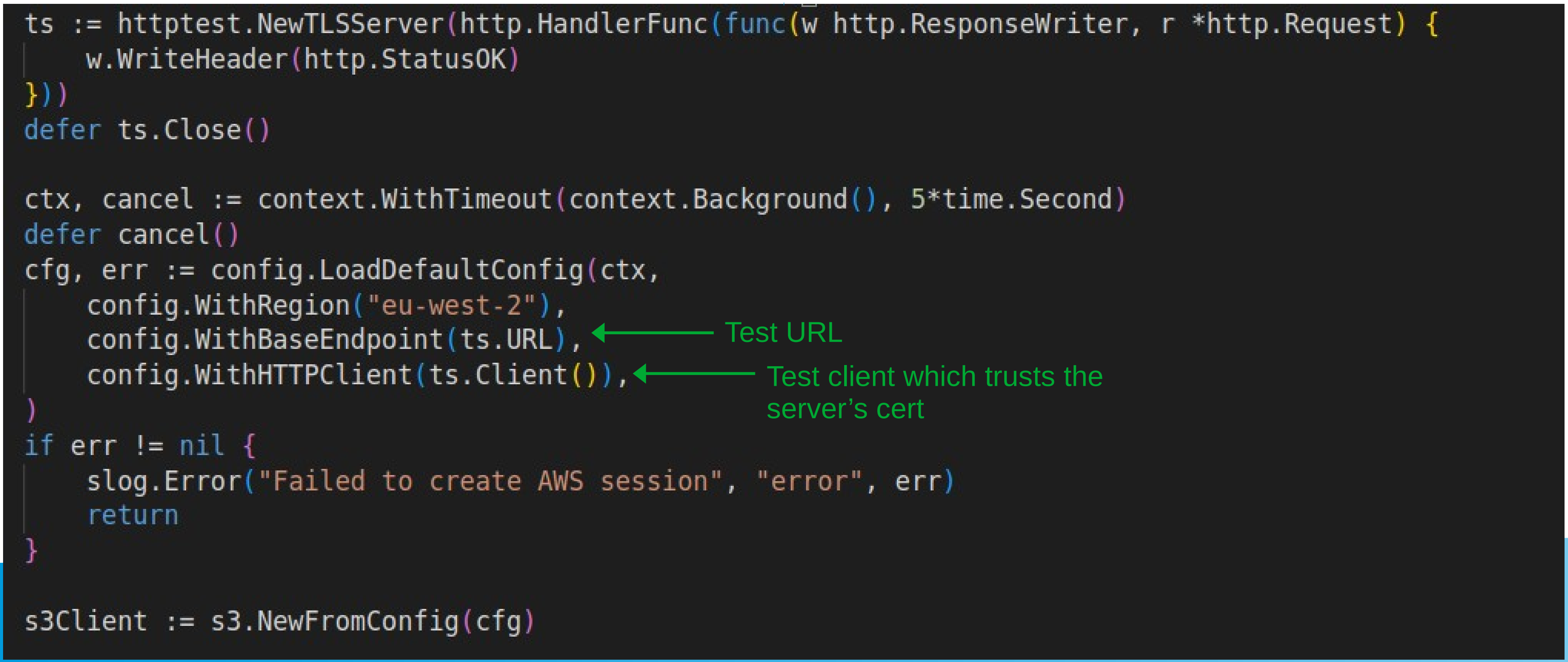
- Creating mock http services that functionally match the real dependency
- Can be thought of as writing our own emulator
- httptest standard library has support to write mock http services

httptest standary library

- Provides functions to create test server with TLS
- Provides a client which trusts the server’s SSL cert
- Exposes host name and port can be used as upstream in Toxiproxy

Testing happy path using http test

Configuring s3 client using http test server



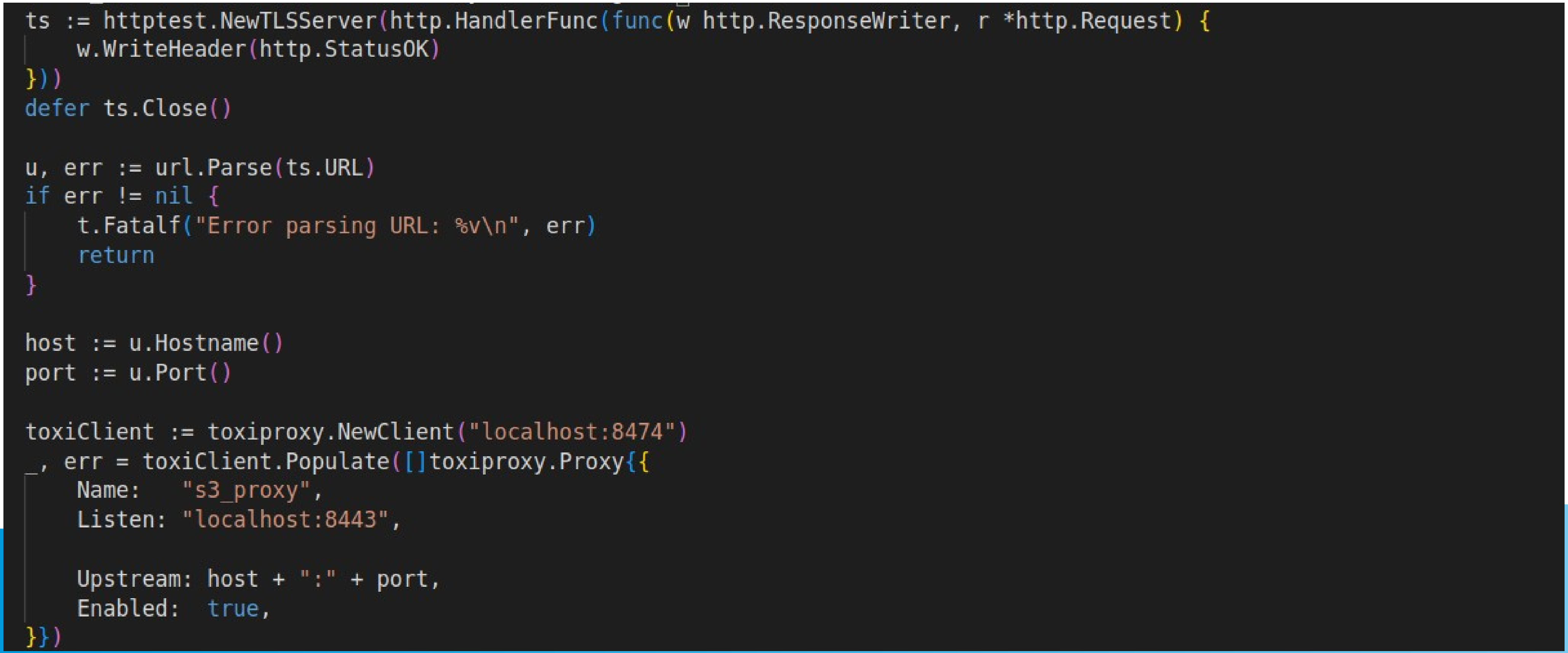
```go
ts := httptest.NewTLSServer(http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
    w.WriteHeader(http.StatusOK)
}))
defer ts.Close()

ctx, cancel := context.WithTimeout(context.Background(), 5*time.Second)
defer cancel()
cfg, err := config.LoadDefaultConfig(ctx,
    config.WithRegion("eu-west-2"),
    config.WithBaseEndpoint(ts.URL),
    config.WithHTTPClient(ts.Client()),
)
if err != nil {
    slog.Error("Failed to create AWS session", "error", err)
    return
}

s3Client := s3.NewFromConfig(cfg)
```

Test URL

Test client which trusts the server's cert

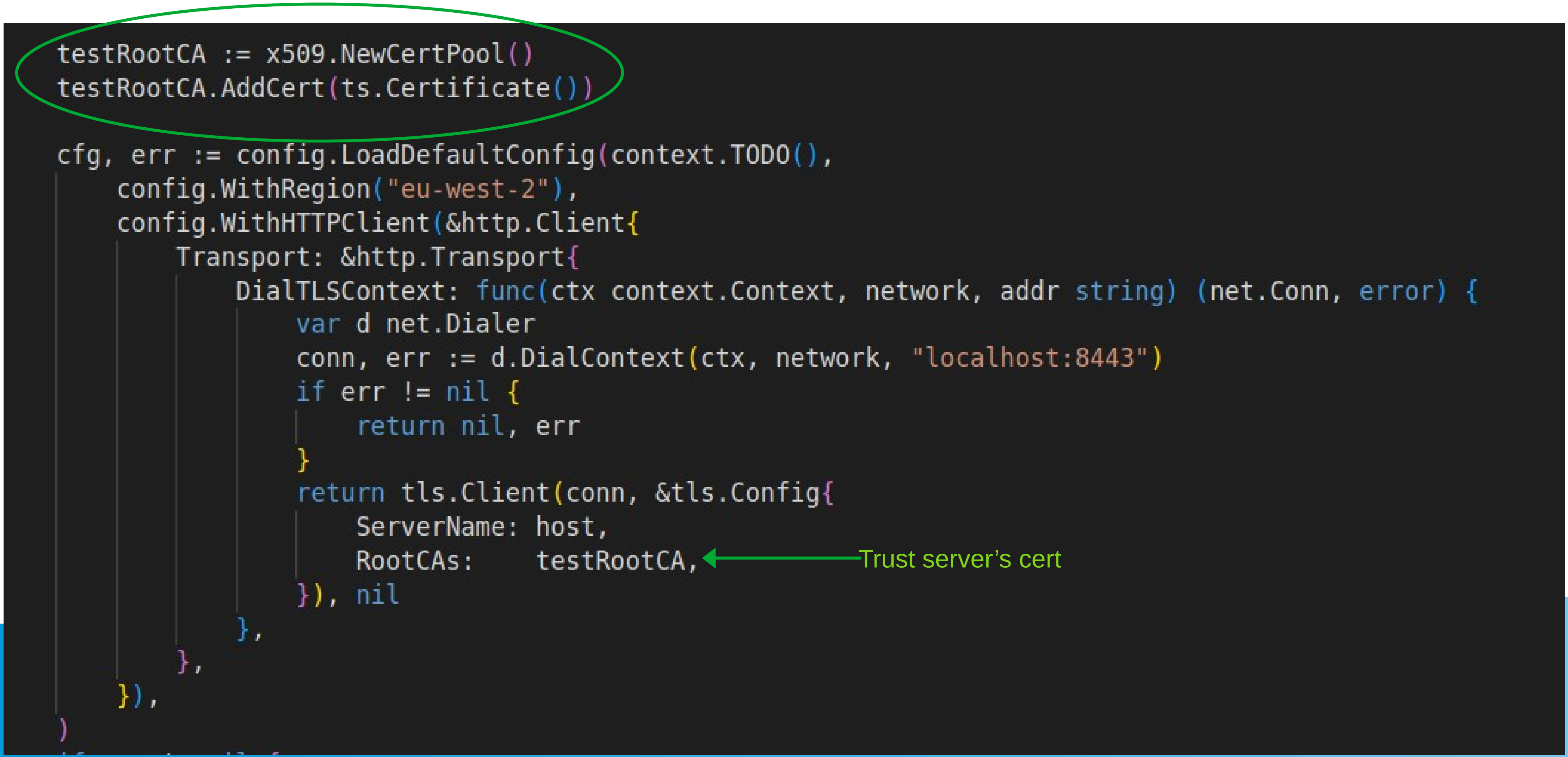
Testing unhappy path

Configuring Toxiproxy to use http test server as upstream



```go
ts := httptest.NewTLSServer(http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
    w.WriteHeader(http.StatusOK)
}))
defer ts.Close()

u, err := url.Parse(ts.URL)
if err != nil {
    t.Fatalf("Error parsing URL: %v\n", err)
    return
}

host := u.Hostname()
port := u.Port()

toxiClient := toxiproxy.NewClient("localhost:8474")
_, err = toxiClient.Populate([]toxiproxy.Proxy{{
    Name:   "s3_proxy",
    Listen: "localhost:8443",

    Upstream: host + ":" + port,
    Enabled:  true,
}})
```

Configuring CA cert for the S3 client



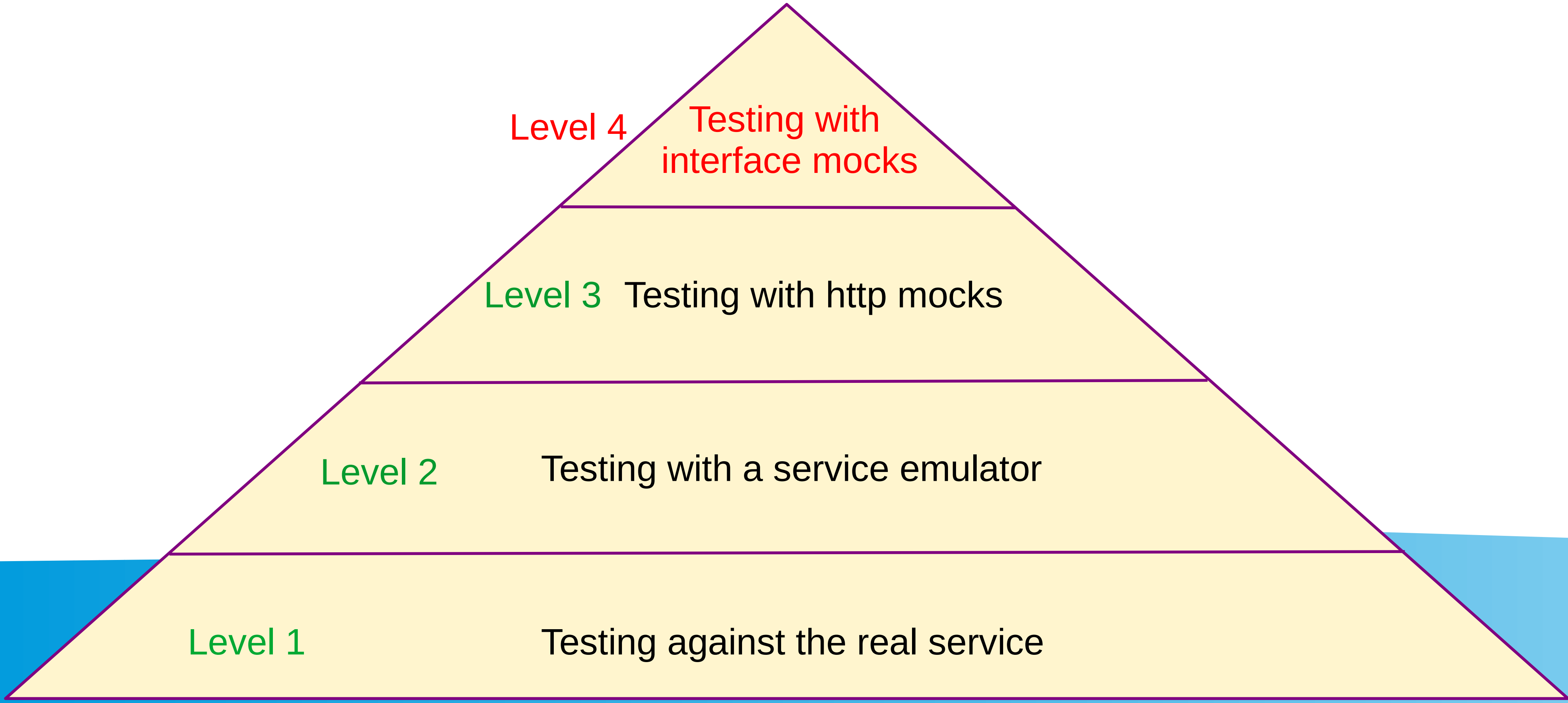
```go
testRootCA := x509.NewCertPool()
testRootCA.AddCert(ts.Certificate())

cfg, err := config.LoadDefaultConfig(context.TODO(),
    config.WithRegion("eu-west-2"),
    config.WithHTTPClient(&http.Client{
        Transport: &http.Transport{
            DialTLSContext: func(ctx context.Context, network, addr string) (net.Conn, error) {
                var d net.Dialer
                conn, err := d.DialContext(ctx, network, "localhost:8443")
                if err != nil {
                    return nil, err
                }
                return tls.Client(conn, &tls.Config{
                    ServerName: host,
                    RootCAs:    testRootCA,
                }), nil
            },
        },
    }),
)
```

Why testing using a http mock is not possible always?

Why testing using a http mock is not possible always?

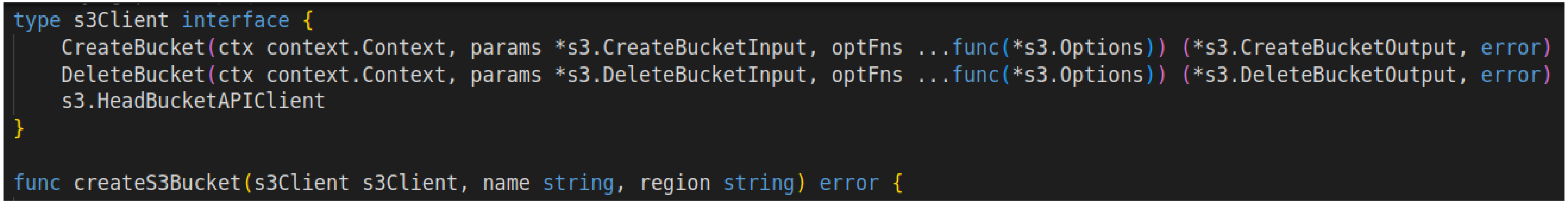
- Closed source dependencies where the SDK doesn’t provide a way to set the http client/transport
- Complex logic involved in the external service and writing a mock proves to be difficult to maintain

Level 4: Testing using interface mocks


Level 4
Testing with interface mocks
Level 3
Testing with http mocks
Level 2
Testing with a service emulator
Level 1
Testing against the real service

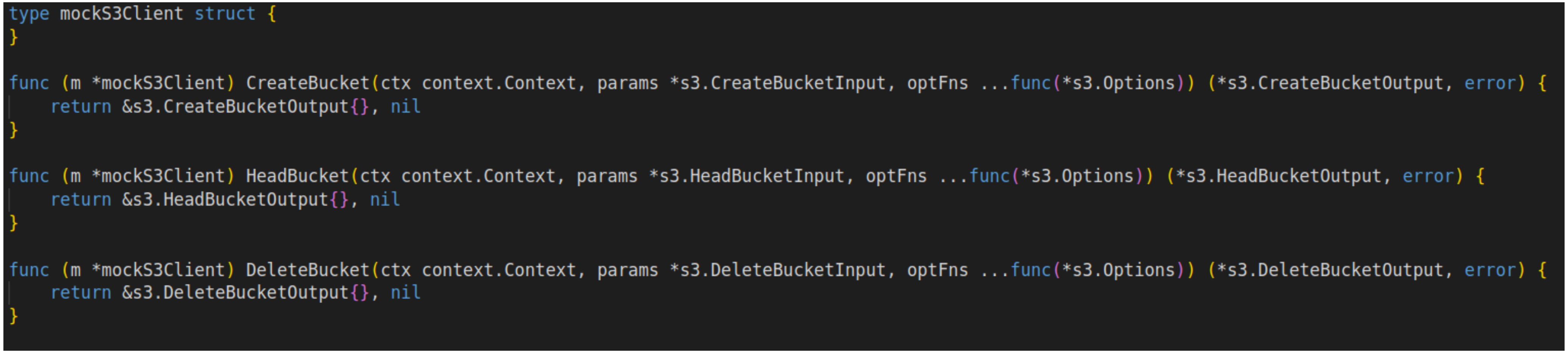
Test happy path with interfaces

Refactor code to accept interfaces



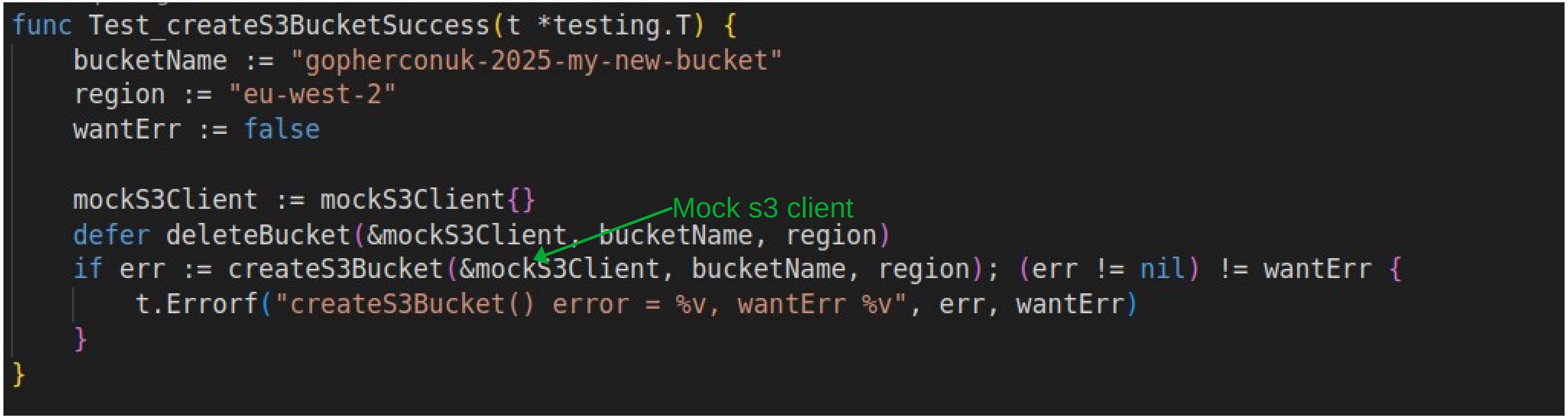
```go
type s3Client interface {
    CreateBucket(ctx context.Context, params *s3.CreateBucketInput, optFns ...func(*s3.Options)) (*s3.CreateBucketOutput, error)
    DeleteBucket(ctx context.Context, params *s3.DeleteBucketInput, optFns ...func(*s3.Options)) (*s3.DeleteBucketOutput, error)
    s3.HeadBucketAPIClient
}

func createS3Bucket(s3Client s3Client, name string, region string) error {
```

Create struct which implements the interface



```go
type mockS3Client struct {
}

func (m *mockS3Client) CreateBucket(ctx context.Context, params *s3.CreateBucketInput, optFns ...func(*s3.Options)) (*s3.CreateBucketOutput, error) {
    return &s3.CreateBucketOutput{}, nil
}

func (m *mockS3Client) HeadBucket(ctx context.Context, params *s3.HeadBucketInput, optFns ...func(*s3.Options)) (*s3.HeadBucketOutput, error) {
    return &s3.HeadBucketOutput{}, nil
}

func (m *mockS3Client) DeleteBucket(ctx context.Context, params *s3.DeleteBucketInput, optFns ...func(*s3.Options)) (*s3.DeleteBucketOutput, error) {
    return &s3.DeleteBucketOutput{}, nil
}
```

Use mock s3 client for test

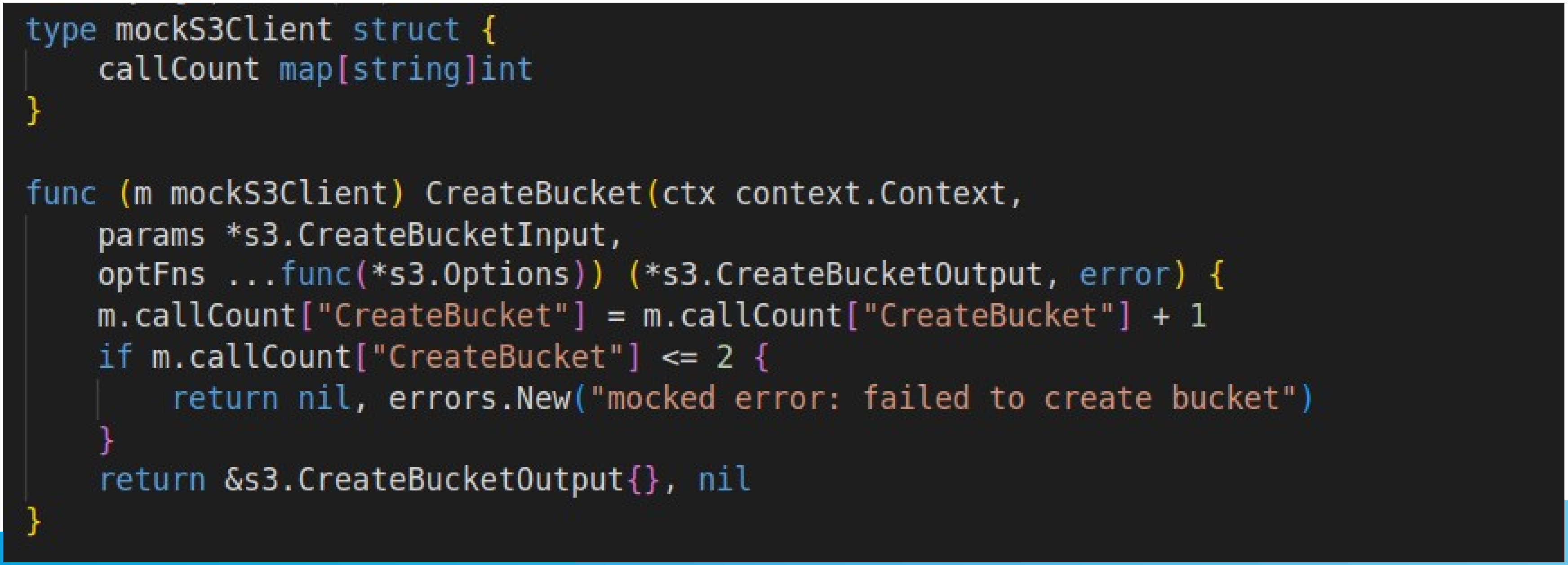


```go
func Test_createS3BucketSuccess(t *testing.T) {
    bucketName := "gopherconuk-2025-my-new-bucket"
    region := "eu-west-2"
    wantErr := false

    mockS3Client := mockS3Client{}
    defer deleteBucket(&mockS3Client, bucketName, region)
    if err := createS3Bucket(&mockS3Client, bucketName, region); (err != nil) != wantErr {
        t.Errorf("createS3Bucket() error = %v, wantErr %v", err, wantErr)
    }
}
```

Test unhappy path with interfaces

- No need for Toxiproxy
- Errors can be returned from the implementation

Return error from mock implementation



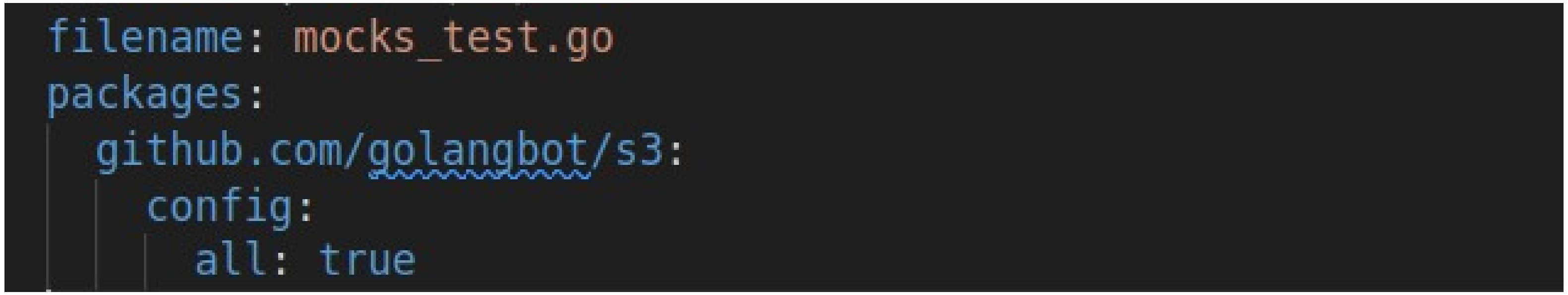
```go
type mockS3Client struct {
    callCount map[string]int
}

func (m mockS3Client) CreateBucket(ctx context.Context,
    params *s3.CreateBucketInput,
    optFns ...func(*s3.Options)) (*s3.CreateBucketOutput, error) {
    m.callCount["CreateBucket"] = m.callCount["CreateBucket"] + 1
    if m.callCount["CreateBucket"] <= 2 {
        return nil, errors.New("mocked error: failed to create bucket")
    }
    return &s3.CreateBucketOutput{}, nil
}
```

Problems with writing mocks manually

- Tedious to maintain mocks
- Mock has to be updated everytime the interface changes

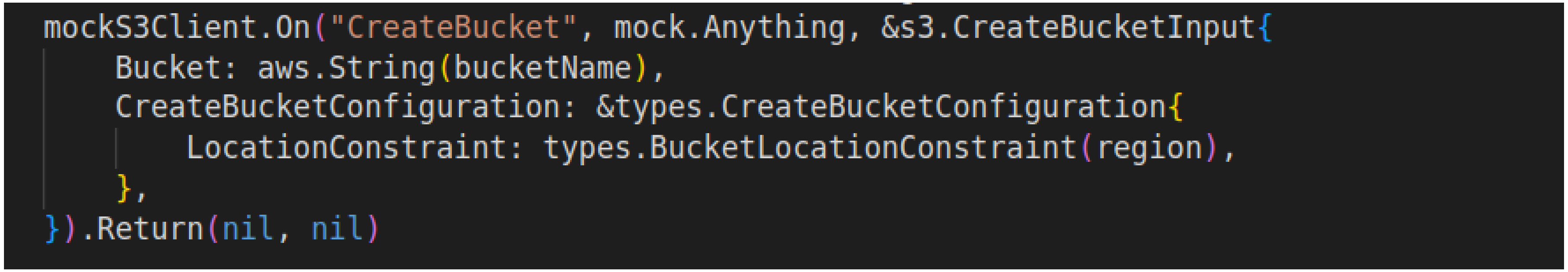
Automatic mock generation using mockery

.mockery.yaml



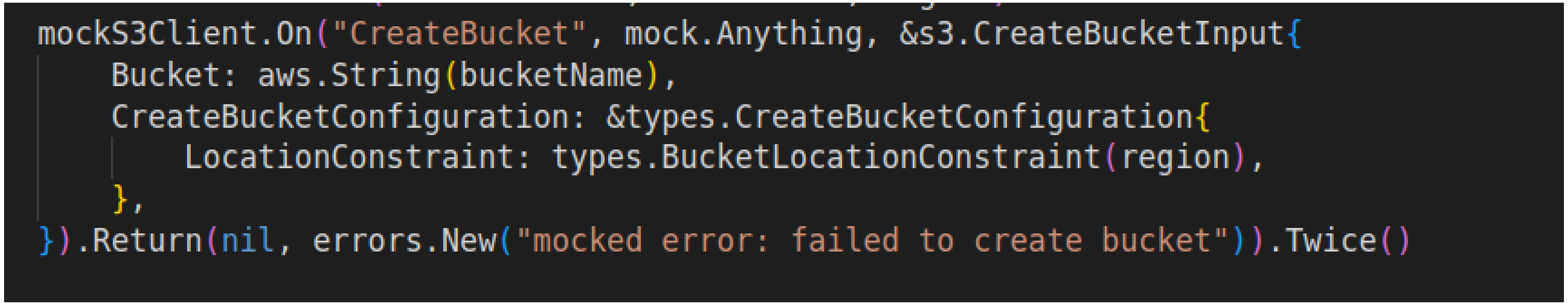
```yaml
filename: mocks_test.go
packages:
  github.com/golangbot/s3:
    config:
      all: true
```

Asserting method calls



```
mockS3Client.On("CreateBucket", mock.Anything, &s3.CreateBucketInput{
    Bucket: aws.String(bucketName),
    CreateBucketConfiguration: &types.CreateBucketConfiguration{
        LocationConstraint: types.BucketLocationConstraint(region),
    },
}).Return(nil, nil)
```

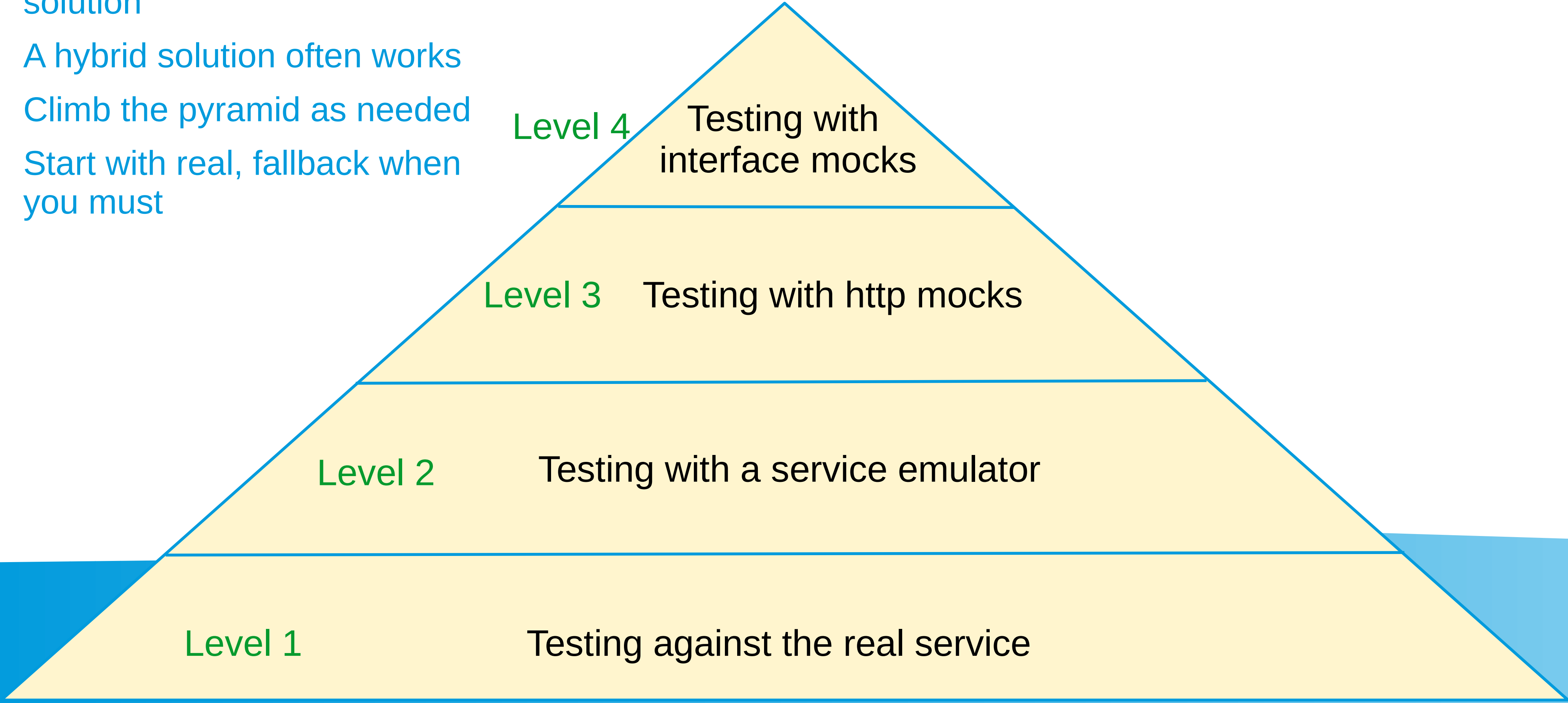
Returning errors from the mock



```
mockS3Client.On("CreateBucket", mock.Anything, &s3.CreateBucketInput{
    Bucket: aws.String(bucketName),
    CreateBucketConfiguration: &types.CreateBucketConfiguration{
        LocationConstraint: types.BucketLocationConstraint(region),
    },
}).Return(nil, errors.New("mocked error: failed to create bucket")).Twice()
```

Conclusion

- There is no one size fits all solution
- A hybrid solution often works
- Climb the pyramid as needed
- Start with real, fallback when you must


Level 4
Testing with interface mocks
Level 3
Testing with http mocks
Level 2
Testing with a service emulator
Level 1
Testing against the real service

Thank you

@bot_golang

naveen@golangbot.com

msgtonaveen