**Sample Jenkinsfile:**

pipeline {

agent any

stages {

stage('Checkout') {

steps {

git 'your\_repository\_url' // Checkout source code from version control

}

}

stage('Unit Tests') {

steps {

sh 'go test ./...' // Run unit tests

}

}

stage('Code coverage') {

steps {

sh 'go tool cover -html=coverage.out -o coverage.html' // Publish code coverage report

publishHTML(target: [

alwaysLinkToLastBuild: false,

reportFiles: 'coverage.html’ ])

}

}

stage('Deploy') {

steps {

// Example: Deploy to a staging server

sh 'scp your\_binary user@staging\_server:/path/to/deployment/directory/'

// Additional steps such as restarting services or running migrations can be added here

}

}

}

post {

always {

emailext ( // Send email notification on build completion

subject: "Jenkins Build Notification: ${currentBuild.result}",

body: "Build ${currentBuild.result}",

to: "recipient@example.com")

cleanWs() // Cleanup or other post-build actions

}

}

}

**How to write testcase:**

// add\_test.go

package main

import (

"testing"

)

func TestAdd(t \*testing.T) {

result := Add(1, 2)

if result != 3 {

t.Errorf("Add(1, 2) = %d; want 3", result)

}

}

**How to write test suites**

// calculator\_test.go

package calculator

import (

"testing"

)

func TestAdd(t \*testing.T) {

// Test cases for Add function

// ...

}

func TestSubtract(t \*testing.T) {

// Test cases for Subtract function

// ...

}

// This function is executed before any tests in the suite are run.

func TestMain(m \*testing.M) {

// Add setup logic, like initializing resources or the environment.

// Run all tests

exitCode := m.Run()

// Teardown logic can go here if needed like clean up resources, restore test environment

// Exit with the status of the test suite run

os.Exit(exitCode)

}

**Mocking dependencies:**

// service.go

package service

type DataFetcher interface {

FetchData() (string, error)

}

type MyService struct {

DataFetcher DataFetcher

}

func (s \*MyService) ProcessData() (string, error) {

data, err := s.DataFetcher.FetchData()

if err != nil {

return "", err

}

// Process data...

return data, nil

}

**Mock Implementation of DataFetcher:**

// mock\_datafetcher.go

package service

type MockDataFetcher struct{}

func (m \*MockDataFetcher) FetchData() (string, error) {

// Simulate data fetching behavior

return "Mocked Data", nil

}

// service\_test.go

package service\_test

import (

"testing"

"your-package-path/service"

)

func TestMyService\_ProcessData(t \*testing.T) {

mockDataFetcher := &service.MockDataFetcher{}

myService := service.MyService{DataFetcher: mockDataFetcher}

result, err := myService.ProcessData()

// Perform assertions on the result and error

}