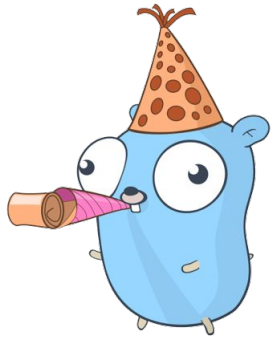
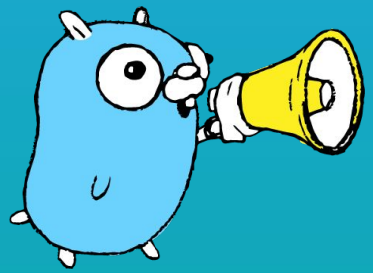


Building CLI Apps with GO

kevsersrca kev_src

$> whoami

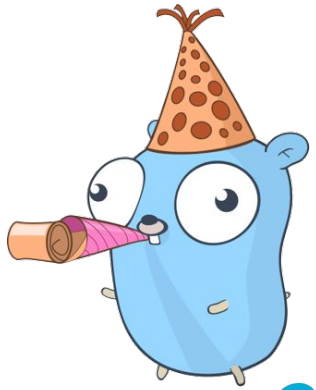
```go
package main

import "fmt"

func main() {
    fmt.Println("Name: Kevser Sırça Alkış")
    fmt.Println("Company/Position: Binalyze/Software Developer")
    fmt.Println("Email: kevser.sirca@gmail.com")
    fmt.Println("Experience: +6 years")
    fmt.Println("Motivation: I love GO <3")
    fmt.Println("Twitter: @kev_src")
    fmt.Println("Github: kevsersrca")
}
```



Agenda

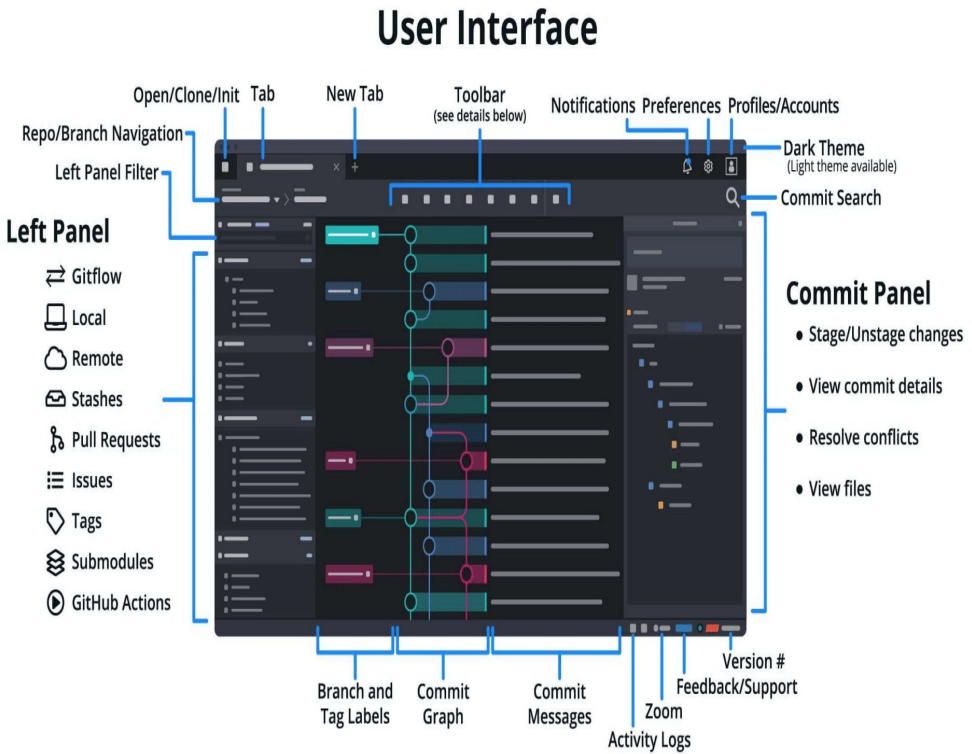
Command Line İnterface	01
CLI Structure	02
Flag Package	03
Mini Docker with Flag	04
Virus total IP Address Scanner with urfave/cli	05
Port Scanner with Cobra CLI	06

```
                                    go run main.go
ls -lah

          curl --proto '=https' --tlsv1.2 -sSf
          https://sh.rustup.rs

                                                    git push
docker ps

                          kubectl proxy


  aws s3 mb s3://new-bucket
```



CLI vs GUI

```
kevsersirca:~/ $ git help
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           [--super-prefix=<path>] [--config-env=<name>=<envvar>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone             Clone a repository into a new directory
   init              Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add               Add file contents to the index
   mv                Move or rename a file, a directory, or a symlink
   restore           Restore working tree files
   rm                Remove files from the working tree and from the index
   sparse-checkout   Initialize and modify the sparse-checkout

examine the history and state (see also: git help revisions)
   bisect            Use binary search to find the commit that introduced a bug
   diff              Show changes between commits, commit and working tree, etc
   grep              Print lines matching a pattern
   log               Show commit logs
   show              Show various types of objects
   status            Show the working tree status

grow, mark and tweak your common history
   branch            List, create, or delete branches
   commit            Record changes to the repository
   merge             Join two or more development histories together
   rebase            Reapply commits on top of another base tip
   reset             Reset current HEAD to the specified state
   switch            Switch branches
   tag               Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch             Download objects and refs from another repository
   pull              Fetch from and integrate with another repository or a local branch
   push              Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.
```



Command Line Interface

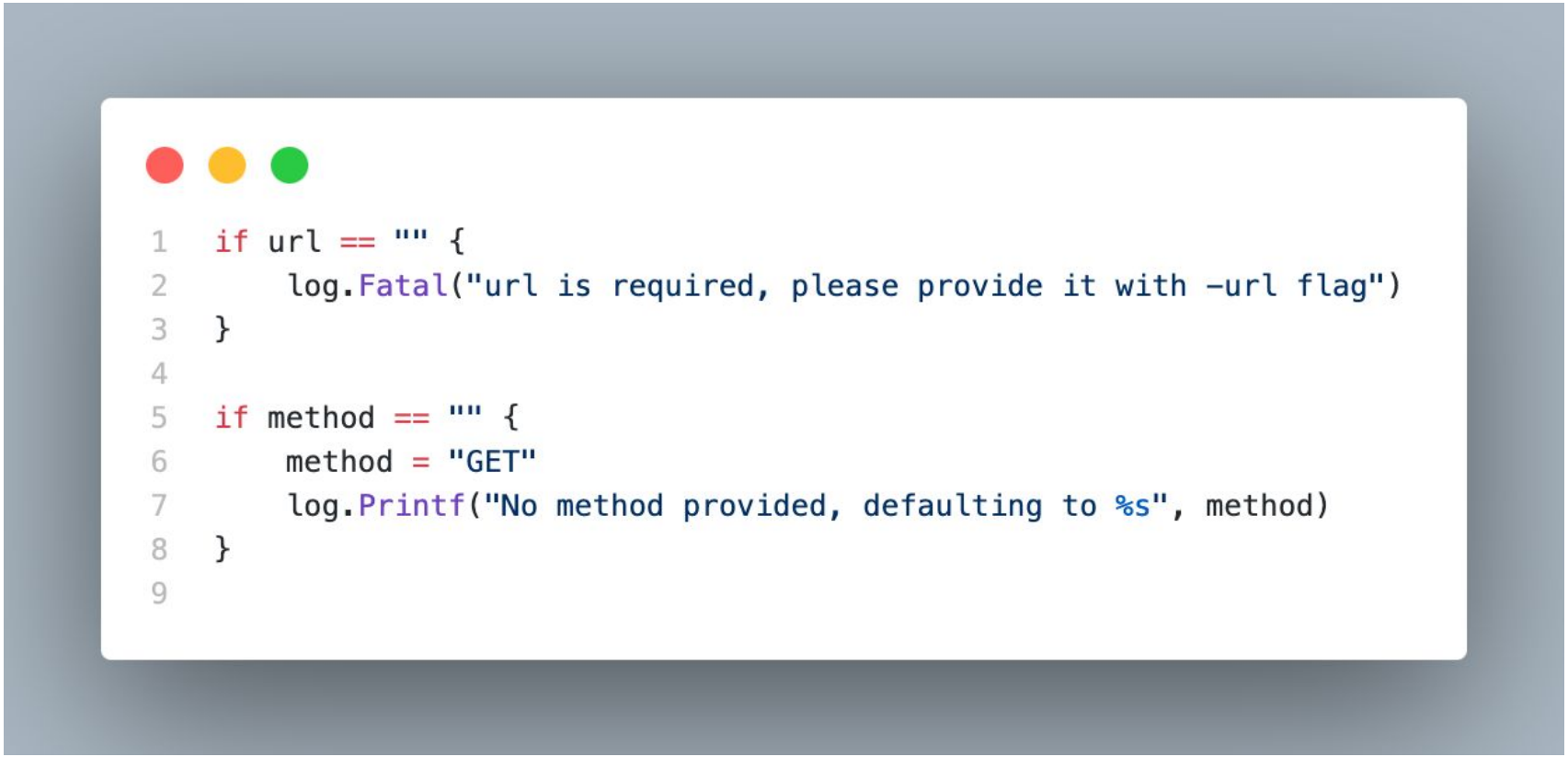
All command-line applications essentially perform the following steps:

- Accept user input
- Perform some validation
- Use the input to perform some custom task
- Present the result to the user; that is, a success or a failure

Command Line Interface

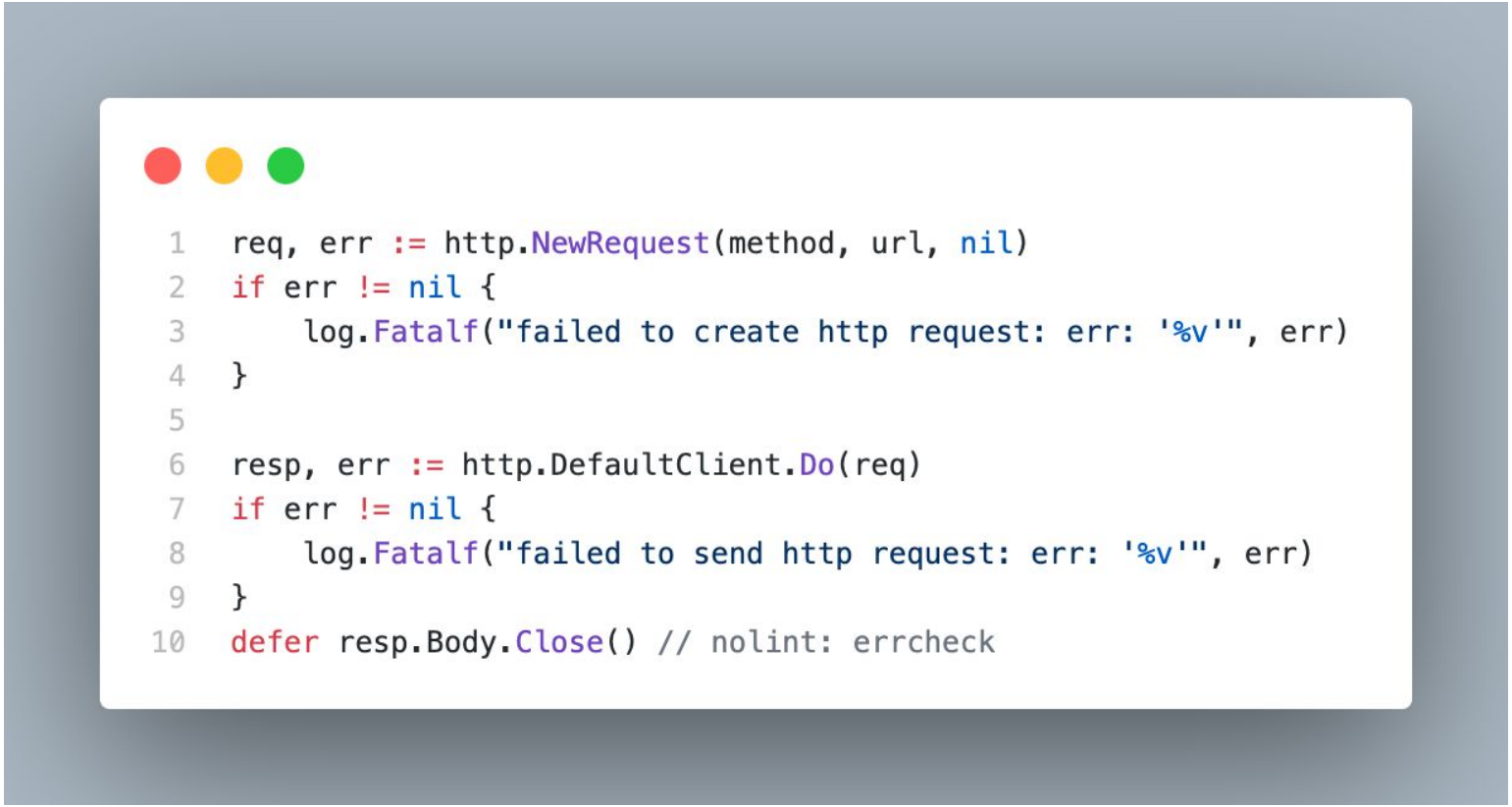
```
1  var url, method string
2  flag.StringVar(&url, "url", "", "URL to send the request")
3  flag.StringVar(&method, "method", "", "HTTP method to use")
4  flag.Parse()
5
```

Command Line Interface



```go
if url == "" {
    log.Fatal("url is required, please provide it with -url flag")
}

if method == "" {
    method = "GET"
    log.Printf("No method provided, defaulting to %s", method)
}

```

Command Line Interface



```go
req, err := http.NewRequest(method, url, nil)
if err != nil {
    log.Fatalf("failed to create http request: err: '%v'", err)
}

resp, err := http.DefaultClient.Do(req)
if err != nil {
    log.Fatalf("failed to send http request: err: '%v'", err)
}
defer resp.Body.Close() // nolint: errcheck
```

Command Line Interface

```
 % go run main.go
simple-cli: main.go:18: url is required, please provide it with -url flag
exit status 1
 % echo $?
1
 %
```

```
 % go run main.go -url="http://google.com"
simple-cli: main.go:23: No method provided, defaulting to GET
simple-cli: main.go:37: Response status: 200 OK
 % echo $?
0
```

Command Line Interface

$? returns the exit status of the last command executed, with 0 indicating success and a non-zero value indicating failure.

PowerShell on Windows, you can use `$LastExitCode` to see the exit code.

Why should we choose Go for CLI Application?

Simple

```go
package main

import (
    "flag"
    "fmt"
)

func main() {
    var port int
    flag.IntVar(&port, "p", 8000, "specify port to use.  defaults to 8000.")
    flag.Parse()

    fmt.Printf("port = %d", port)
}
```

Why should we choose Go for CLI Application?

Modular

```go
package main

import (
    "gokonf-cli/internal/cli"
)

func main() {
    cli.Execute()
}
```

Why should we choose Go for CLI Application?

Single Binary

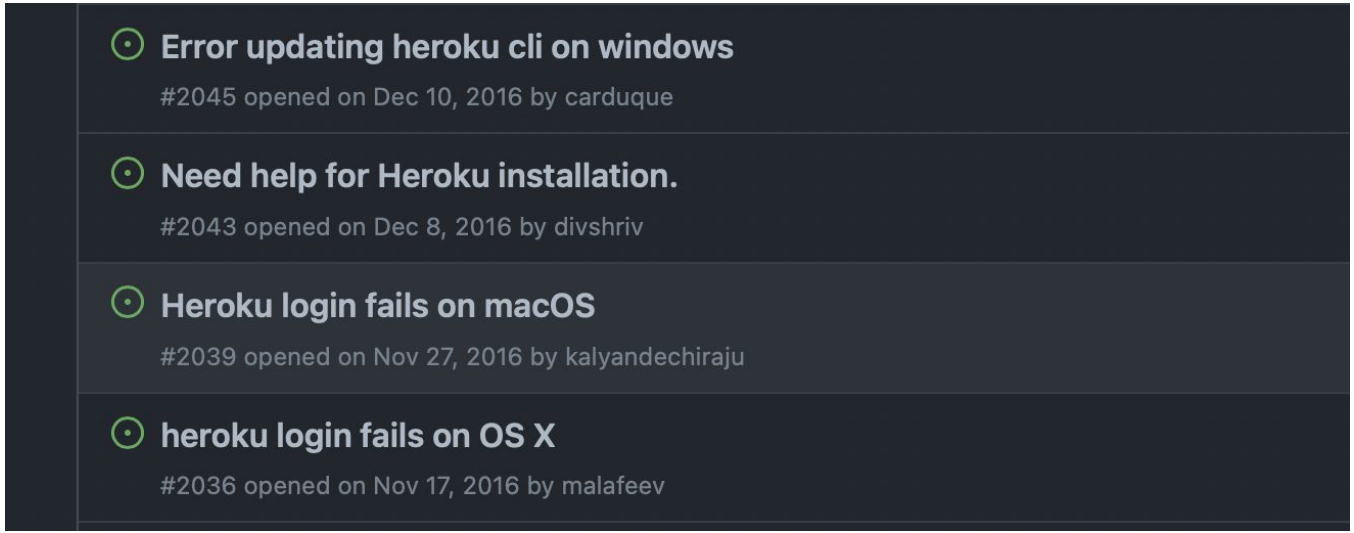
can be compiled into a single executable file (binary) that does not require dependencies, external libraries, or runtimes to be included.

Why should we choose Go for CLI Application?

Cross-platform

```
BINARY_NAME=gokonf-cli

all: windows linux darwin

windows:
    GOOS=windows GOARCH=amd64 go build -o bin/$(BINARY_NAME)-windows-amd64 main.go

linux:
    GOOS=linux GOARCH=amd64 go build -o bin/$(BINARY_NAME)-linux-amd64 main.go

darwin:
    GOOS=darwin GOARCH=amd64 go build -o bin/$(BINARY_NAME)-darwin-amd64 main.go

clean:
    rm -f bin/*
```

Why should we choose Go for CLI Application?



“

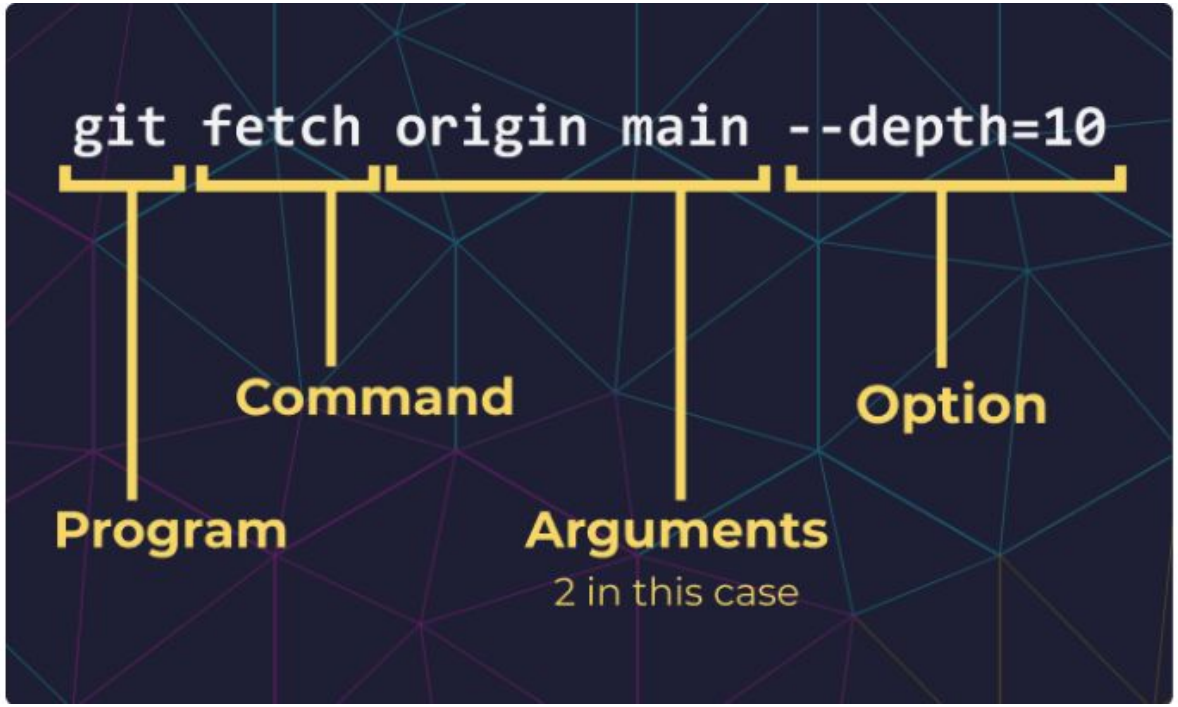
Write once,
run anywhere

”

CLI Structure

- **Arguments**: Allow users to send data to the application. Sometimes required in a command context.
- **Options**: Act as named parameters passed to a command, represented by key-value pairs.
- **Flags**: Options that do not require a value. They are boolean; their presence indicates "true," and their absence indicates "false."

CLI Structure



CLI Structure

```
var os.Args []string

Args hold the command-line arguments, starting with the program name.

os.Args on pkg.go.dev
```

```go
func main() {
    // Program Name is always the first (implicit) argument
    cmd := os.Args[0]
    fmt.Printf("Program Name: %s\n", cmd)

    argCount := len(os.Args[1:])
    fmt.Printf("All Arguments count %d\n", argCount)
}
```

CLI Structure

```
func main() {
    for i, a := range os.Args[1:] {
        fmt.Printf("Argument %d is %s\n", i+1, a)
    }

}
```

```
$ ./example -local u=admin --help
Argument 1 is -local
Argument 2 is u=admin
Argument 3 is --help
```

Flag Package

```
func flag.StringVar(p *string, name string, value string, usage string)

StringVar defines a string flag with specified name, default value, and usage string. The argument p points to a string variable in which to store the value of the flag.

flag.StringVar on pkg.go.dev
```

```go
func main() {
    var stringOption string
    flag.StringVar(&stringOption, "s", "defaultString", "A string flag")

    var intOption int
    flag.IntVar(&intOption, "i", 42, "An int flag")

    var boolOption bool
    flag.BoolVar(&boolOption, "b", false, "A bool flag")

    var float64Option float64
    flag.Float64Var(&float64Option, "f", 3.14, "A float64 flag")

    var durationOption time.Duration
    flag.DurationVar(&durationOption, "d", time.Second, "A duration flag")

    flag.Parse()
}
```

Flag Package

```
func flag.String(name string, value string, usage string) *string
```

String defines a string flag with specified name, default value, and usage string. The return value is the address of a string variable that stores the value of the flag.

flag.String on pkg.go.dev

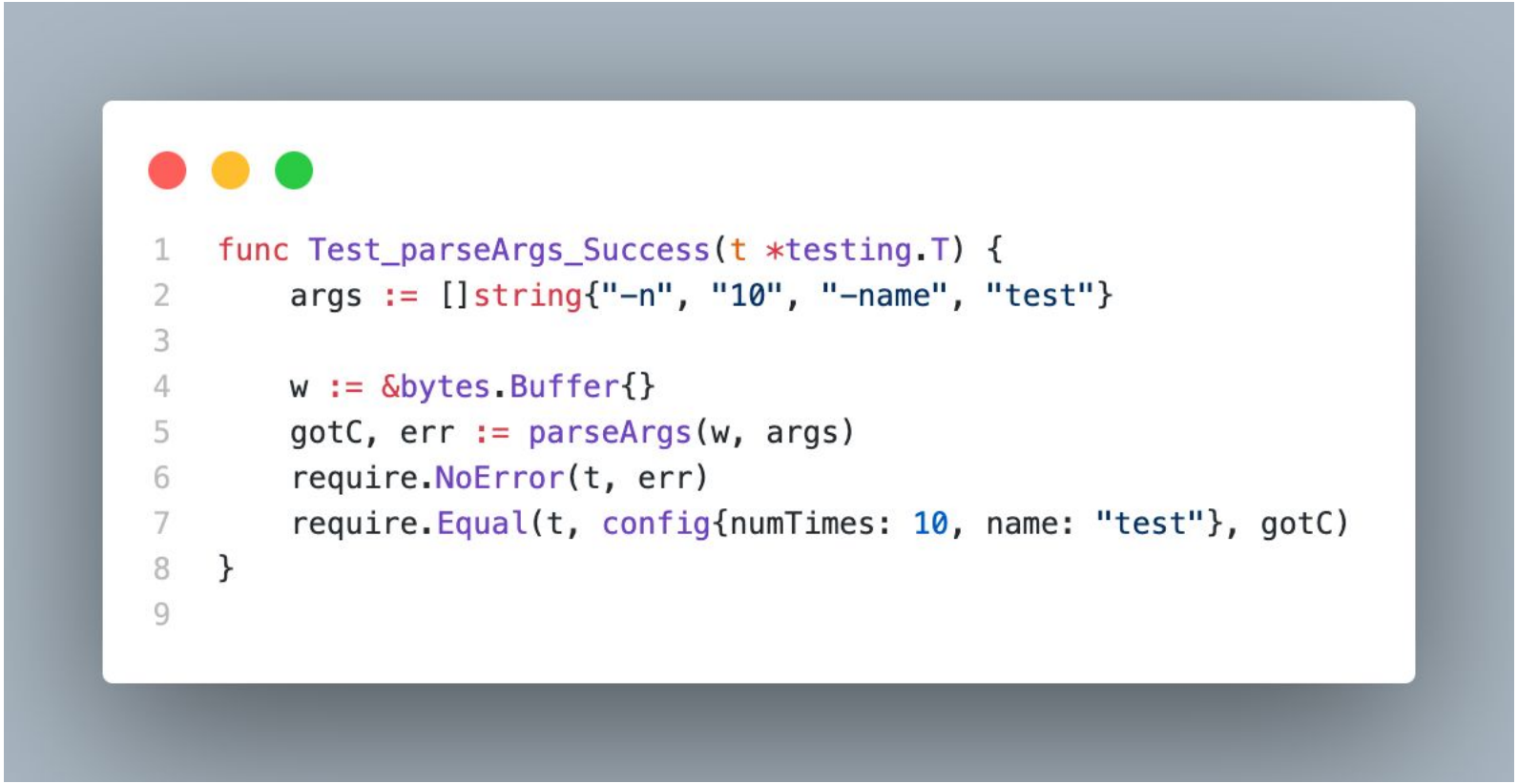
```go
stringOption := flag.String("s", "defaultString", "A string flag")

intOption := flag.Int("i", 42, "An int flag")

boolOption := flag.Bool("b", false, "A bool flag")

float64Option := flag.Float64("f", 3.14, "A float64 flag")

durationOption := flag.Duration("d", time.Second, "A duration flag")
```

Flagset

```go
func parseArgs(w io.Writer, args []string) (config, error) {
    var c config

    fs := flag.NewFlagSet("greeter", flag.ContinueOnError)
    fs.SetOutput(w)

    fs.IntVar(&c.numTimes, "n", 0, "Number of times to greet")
    fs.StringVar(&c.name, "name", "", "Name to greet")

    err := fs.Parse(args)
    if err != nil {
        return c, err
    }
    if c.numTimes <= 0 || c.name == "" {
        err = errors.New("numTimes and name are required")
    }
    return c, err
}
```

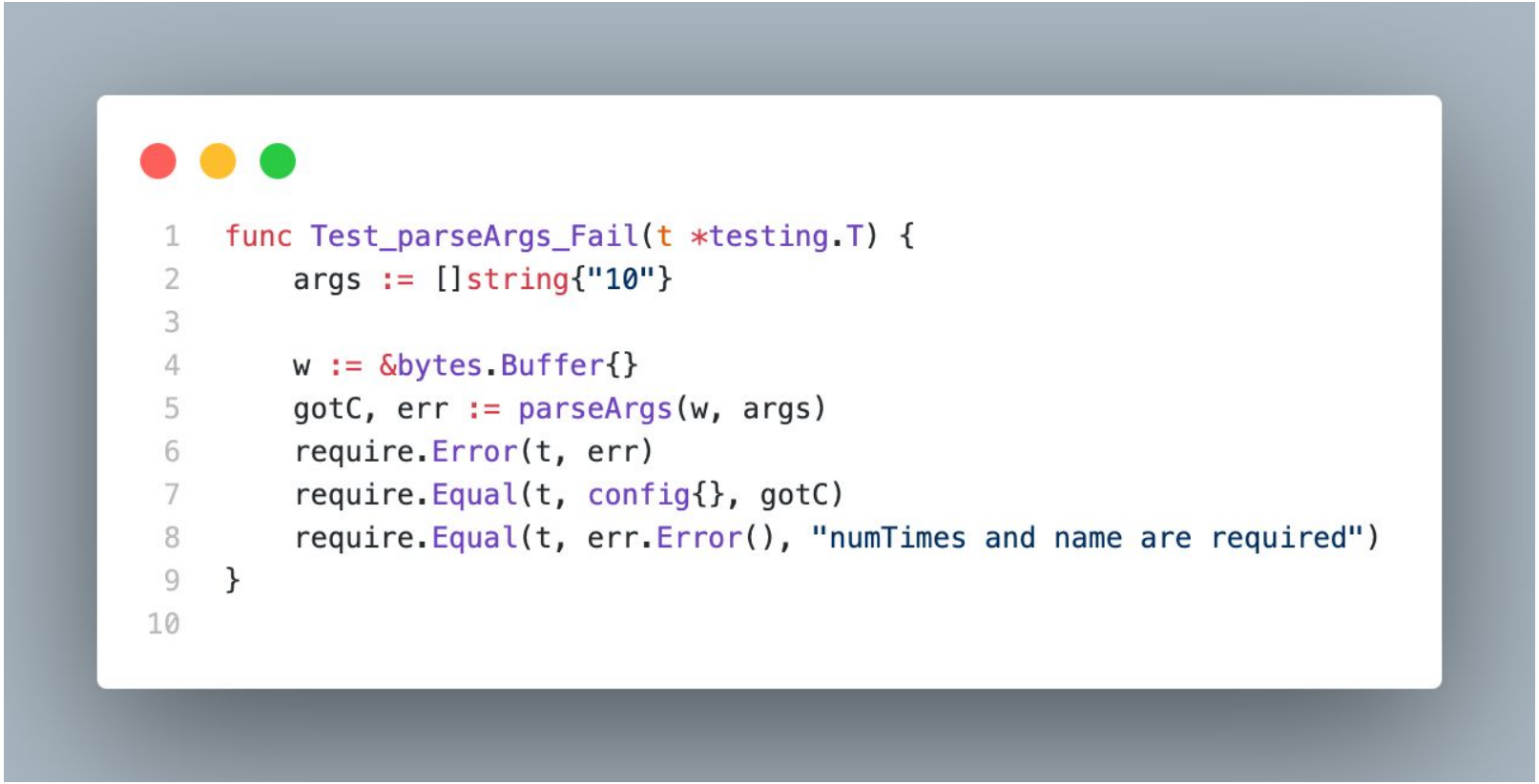
```go
type config struct {
    numTimes int
    name     string
}
```

Writing Unit Tests



```go
func Test_parseArgs_Success(t *testing.T) {
    args := []string{"-n", "10", "-name", "test"}

    w := &bytes.Buffer{}
    gotC, err := parseArgs(w, args)
    require.NoError(t, err)
    require.Equal(t, config{numTimes: 10, name: "test"}, gotC)
}
```

Writing Unit Tests

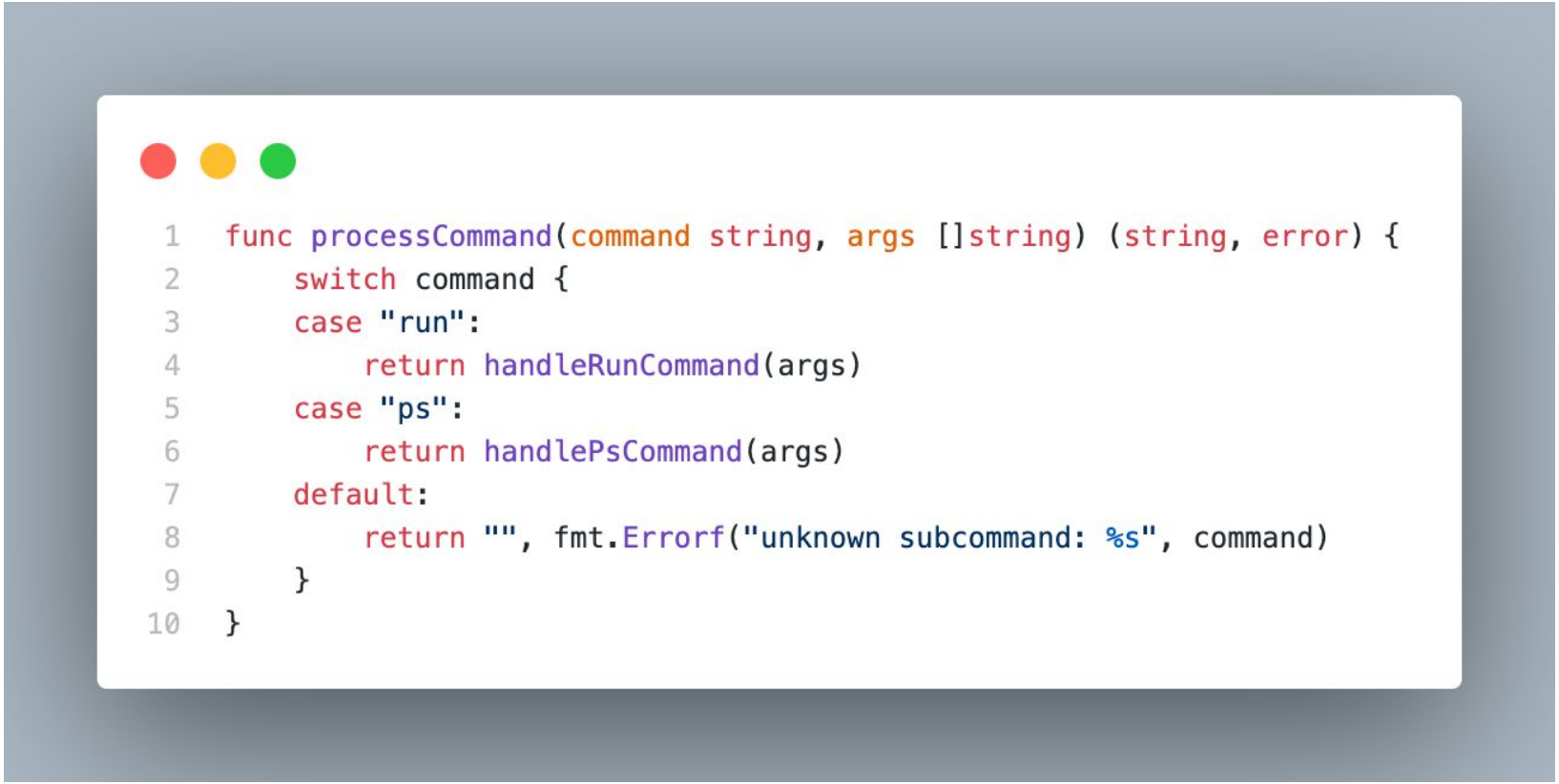


```go
func Test_parseArgs_Fail(t *testing.T) {
    args := []string{"10"}

    w := &bytes.Buffer{}
    gotC, err := parseArgs(w, args)
    require.Error(t, err)
    require.Equal(t, config{}, gotC)
    require.Equal(t, err.Error(), "numTimes and name are required")
}

```

Mini Docker with Flag

```go
func main() {
    if len(os.Args) < 2 {
        fmt.Println("Expected 'run' or 'ps' subcommands")
        return
    }

    output, err := processCommand(os.Args[1], os.Args[2:])
    if err != nil {
        fmt.Println(err)
        return
    }

    fmt.Println(output)
}

```

Mini Docker with Flag



```go
func processCommand(command string, args []string) (string, error) {
    switch command {
    case "run":
        return handleRunCommand(args)
    case "ps":
        return handlePsCommand(args)
    default:
        return "", fmt.Errorf("unknown subcommand: %s", command)
    }
}
```

Mini Docker with Flag

```go
func handlePsCommand(args []string) (string, error) {
    psCommand := flag.NewFlagSet("ps", flag.ExitOnError)
    psAll := psCommand.Bool("a", false, "Show all containers")
    if err := psCommand.Parse(args); err != nil {
        return "", err
    }

    if *psAll {
        return "Listing all containers", nil
    }

    return "Listing all running containers", nil
}
```

Mini Docker with Flag

```go
func handleRunCommand(args []string) (string, error) {
    runCommand := flag.NewFlagSet("run", flag.ExitOnError)
    runImageName := runCommand.String("image", "", "Image name to run")
    if err := runCommand.Parse(args); err != nil {
        return "", err
    }

    if *runImageName == "" {
        return "", fmt.Errorf("you must specify an image name with the -image flag")
    }

    return fmt.Sprintf("Running container from image: %s", *runImageName), nil
}
```

Mini Docker with Flag

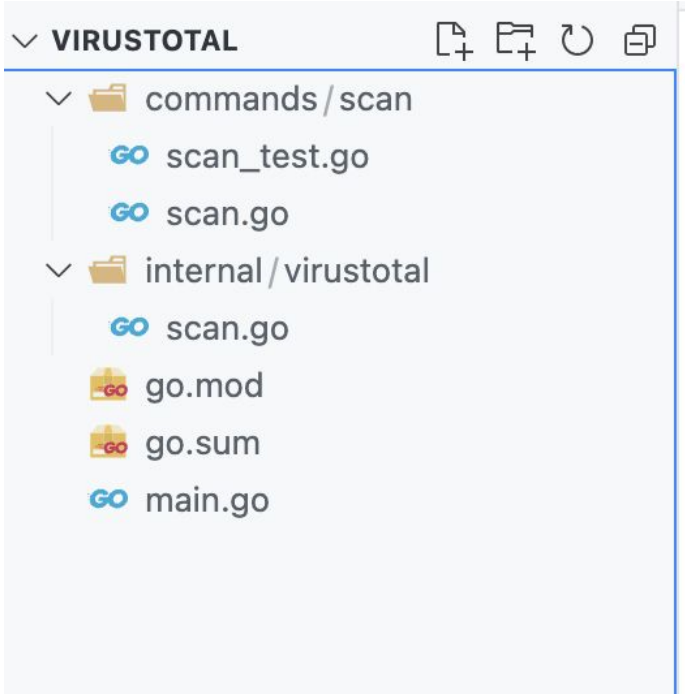
```go

func TestHandleRunCommand(t *testing.T) {
    tests := []struct {
        args []string
        want string
        err  bool
    }{
        {[]string{"-image", "nginx"}, "Running container from image: nginx", false},
        {[]string{}, "", true},
    }

    for _, tt := range tests {
        got, err := handleRunCommand(tt.args)
        require.Equal(t, tt.want, got)
        require.Equal(t, tt.err, err != nil)
    }
}

```

Virus total IP Address Scanner

```go
func main() {
    app := &cli.App{
        Name:     "Virustotal",
        Usage:    "Virustotal CLI tool to scan files using the Virustotal API.",
        Commands: Commands(os.Stdin),
        Version:  "0.0.1",
    }

    if err := app.Run(os.Args); err != nil {
        log.Fatal(err)
    }
}
```

Virus total IP Address Scanner

```go
func Commands(reader io.Reader) []*cli.Command {
    return []*cli.Command{
        scan.Command(),
    }
}

```



Virus total IP Address Scanner

```go
const (
    CmdScan = "scan"
)

func Command() *cli.Command {
    return &cli.Command{
        Name:        CmdScan,
        HelpName:    CmdScan,
        Action:      Action,
        ArgsUsage:   `<ip address> <api-key>`,
        Usage:       `virustotal scan <ip-address> <api-key>`,
        Description: `virustotal ip address scan using the virustotal api-key.`,
        Flags:       Flags(),
    }
}
```

Virus total IP Address Scanner

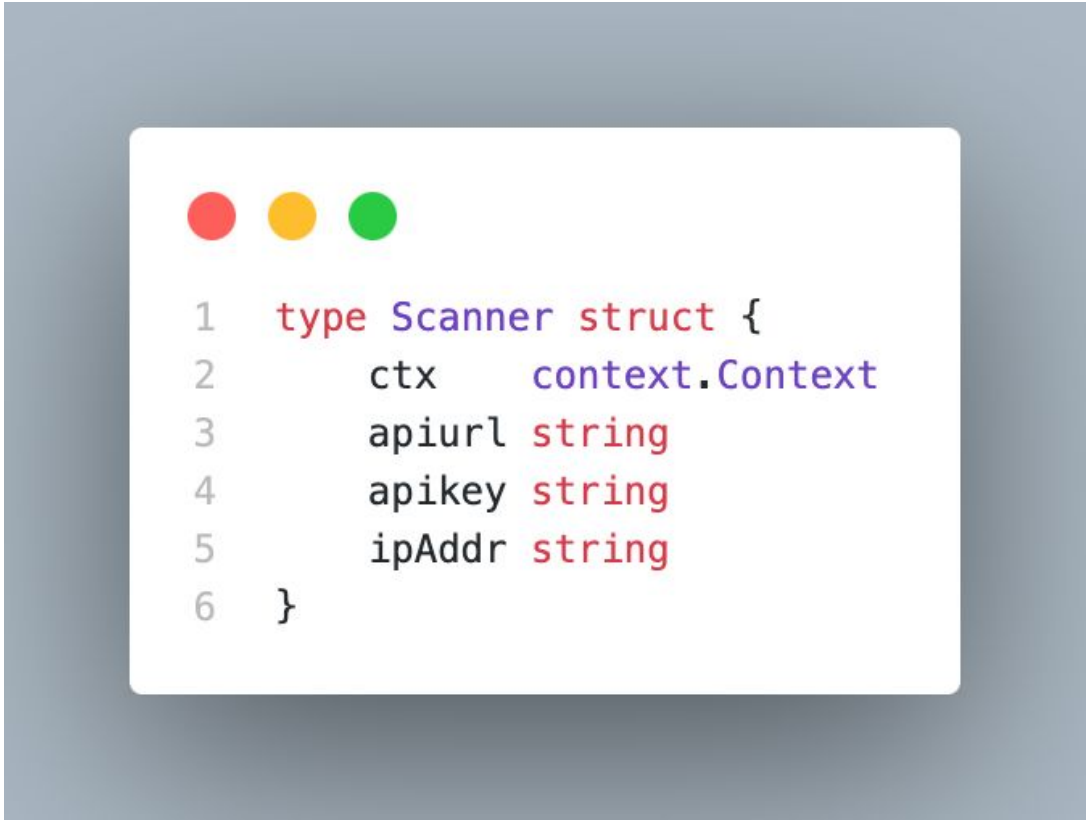
```go
func Flags() []cli.Flag {
    return []cli.Flag{
        &cli.StringFlag{
            Name:     flagIPAddress,
            Usage:    "virustotal ip address to scan",
            Required: true,
        },
        &cli.StringFlag{
            Name:     flagApiKey,
            Usage:    "virustotal api key",
            Required: true,
        },
        &cli.StringFlag{
            Name:        flagApiUrl,
            Usage:       "virustotal api url",
            DefaultText: "",
        },
    }
}
```

```go
const (
    flagIPAddress = "ip-address"
    flagApiKey    = "api-key"
    flagApiUrl    = "api-url"
)
```

Virus total IP Address Scanner

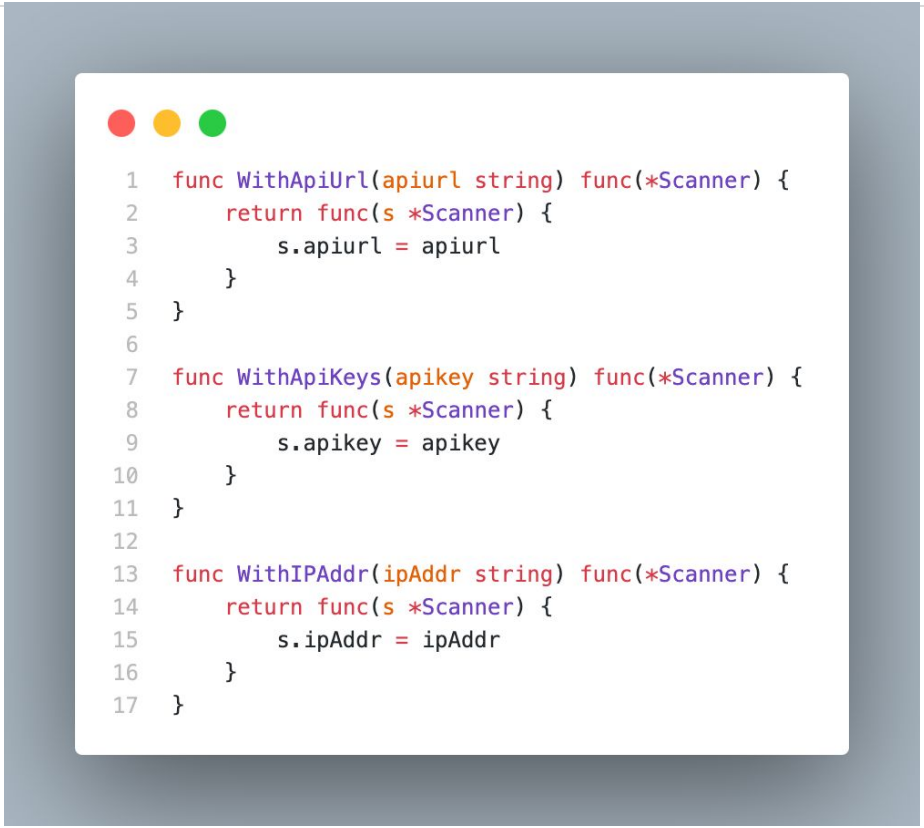
```go
func Action(c *cli.Context) error {
    ipAddr := c.String(flagIPAddress)
    apiKey := c.String(flagApiKey)
    apiurl := c.String(flagApiUrl)

    // validate options

    ctx := context.Background()
    s := virustotal.NewScanner(
        ctx,
        virustotal.WithApiUrl(apiurl),
        virustotal.WithApiKeys(apiKey),
        virustotal.WithIPAddr(ipAddr),
    )
    results := s.Run()
    fmt.Printf("Results: %v", results)
    return nil
}

```

Virus total IP Address Scanner



```
type Scanner struct {
    ctx    context.Context
    apiurl string
    apikey string
    ipAddr string
}
```

Virus total Path Scanner



```go
func WithApiUrl(apiurl string) func(*Scanner) {
    return func(s *Scanner) {
        s.apiurl = apiurl
    }
}

func WithApiKeys(apikey string) func(*Scanner) {
    return func(s *Scanner) {
        s.apikey = apikey
    }
}

func WithIPAddr(ipAddr string) func(*Scanner) {
    return func(s *Scanner) {
        s.ipAddr = ipAddr
    }
}
```

Virus total IP Address Scanner

```
1
2  func NewScanner(ctx context.Context, options ...func(*Scanner)) *Scanner {
3      s := &Scanner{
4          ctx: ctx,
5      }
6
7      for _, option := range options {
8          option(s)
9      }
10
11     return s
12 }
```

Virus total IP Address Scanner

```go
func (s *Scanner) Run() string {
    body, err := s.sendScanRequest()
    if err != nil {
        return err.Error()
    }

    return string(body)
}
```

Virus total IP Address Scanner

```go
func (s *Scanner) sendScanRequest() ([]byte, error) {
    url := fmt.Sprintf("%s/%s", s.apiurl, s.ipAddr)

    req, err := http.NewRequestWithContext(s.ctx, http.MethodGet, url, nil)
    if err != nil {
        log.Printf("error creating request: %v", err)
        return nil, err
    }
    req.Header.Add("x-apikey", s.apikey)

    resp, err := http.DefaultClient.Do(req)
    if err != nil {
        return nil, err
    }
    defer resp.Body.Close()

    var buf bytes.Buffer
    _, err = io.Copy(&buf, resp.Body)
    if err != nil {
        return nil, err
    }
    if resp.StatusCode != 200 {
        return nil, err
    }

    return buf.Bytes(), nil
}
```

Unit test

run test | debug test | pin test

```go
func TestScan(t *testing.T) {
    server := httptest.NewServer(http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        if r.Header.Get("x-apikey") != "test-api-key" {
            w.WriteHeader(http.StatusUnauthorized)
            return
        }

        w.WriteHeader(http.StatusOK)
        w.Write([]byte(`{"response": "test"}`))
    }))
    defer server.Close()
```

Unit test

```
app := &cli.App{
    Commands: []*cli.Command{
        scan.Command(),
    },
}

execName, err := os.Executable()
require.NoError(t, err)
```

Unit test

```
testCases := []struct {
    name      string
    ipAddr    string
    apiKey    string
    apiurl    string
    shouldErr bool
}{
    {
        name:      "success",
        ipAddr:    "127.0.0.1",
        apiKey:    "test-api-key",
        apiurl:    server.URL,
        shouldErr: false,
    },
    {
        name:      "no-ip-address",
        ipAddr:    "",
        apiKey:    "test-api-key",
        apiurl:    server.URL,
        shouldErr: true,
    },
}
```

Unit test

```
    for _, tC := range testCases {
        t.Run(tC.name, func(t *testing.T) {
            testArgs := []string{execName, scan.CmdScan}
            if tC.ipAddr != "" {
                testArgs = append(testArgs, "--ip-address", tC.ipAddr)
            }
            if tC.apiKey != "" {
                testArgs = append(testArgs, "--api-key", tC.apiKey)
            }
            if tC.apiurl != "" {
                testArgs = append(testArgs, "--api-url", tC.apiurl)
            }

            err = app.Run(testArgs)
            if tC.shouldErr {
                require.Error(t, err)
            } else {
                require.NoError(t, err)
            }
        })
    }
}
```

Output

```
kevsersirca@kevsersirca:cmd/virustotal $ go run main.go scan --ip-address "8.8.8.8"
NAME:
   Virustotal scan - virustotal scan <ip-address> <api-key>

USAGE:
   Virustotal scan [command options] <ip address> <api-key>

DESCRIPTION:
   virustotal ip address scan using the virustotal api-key.

OPTIONS:
   --ip-address value  virustotal ip address to scan
   --api-key value     virustotal api key
   --api-url value     virustotal api url
   --help, -h          show help
2024/02/15 20:25:03 Required flag "api-key" not set
exit status 1
```

Output

```
kevsersirca@kevsersirca:cmd/virustotal $ go run main.go scan --ip-address "8.8.8.8" --api-key [redacted]
Result: {
    "data": {
        "attributes": {
            "jarm": "29d3fd00029d29d00042d43d00041d598ac0c1012db967bb1ad0ff2491b3ae",
            "network": "8.8.8.0/24",
            "last_https_certificate_date": 1708016792,
            "tags": [],
            "whois": "NetRange: 8.8.8.0 - 8.8.8.255\nCIDR: 8.8.8.0/24\nNetName: GOGL\nNetHandle: NET-8-8-8-0-2\nParent: NET8 (NET-8-0-0-0-0)\n
NetType: Direct Allocation\nOriginAS: \nOrganization: Google LLC (GOGL)\nRegDate: 2023-12-28\nUpdated: 2023-12-28\nRef: https://rdap.arin.net/
registry/ip/8.8.8.0\nOrgName: Google LLC\nOrgId: GOGL\nAddress: 1600 Amphitheatre Parkway\nCity: Mountain View\nStateProv: CA\nPostalCode: 940
43\nCountry: US\nRegDate: 2000-03-30\nUpdated: 2019-10-31\nComment: Please note that the recommended way to file abuse complaints are located
in the following links. \nComment: \nComment: To report abuse and illegal activity: https://www.google.com/contact/\nComment: \nComment: For l
egal requests: http://support.google.com/legal \nComment: \nComment: Regards, \nComment: The Google Team\nRef: https://rdap.arin.net/registry/
entity/GOGL\nOrgTechHandle: ZG39-ARIN\nOrgTechName: Google LLC\nOrgTechPhone: +1-650-253-0000 \nOrgTechEmail: arin-contact@google.com\nOrgTech
Ref: https://rdap.arin.net/registry/entity/ZG39-ARIN\nOrgAbuseHandle: ABUSE5250-ARIN\nOrgAbuseName: Abuse\nOrgAbusePhone: +1-650-253-0000 \nOr
gAbuseEmail: network-abuse@google.com\nOrgAbuseRef: https://rdap.arin.net/registry/entity/ABUSE5250-ARIN\n",
            "last_analysis_date": 1708016489,
            "as_owner": "GOOGLE",
            "last_analysis_stats": {
                "harmless": 66,
                "malicious": 2,
                "suspicious": 0,
                "undetected": 22,
                "timeout": 0
            },
            "asn": 15169,
            "whois_date": 1707741931,
            "reputation": 575,
            "last_analysis_results": {
                "Bkav": {
                    "category": "undetected",
                    "result": "unrated",
                    "method": "blacklist",
                    "engine_name": "Bkav"
                },
                "CMC Threat Intelligence": {
                    "category": "harmless",
                    "result": "clean",
```

Cobra CLI

```
~/go/src/gokonf-cli/cobra cobra-cli
Cobra is a CLI library for Go that empowers applications.
This application is a tool to generate the needed files
to quickly create a Cobra application.

Usage:
  cobra-cli [command]

Available Commands:
  add         Add a command to a Cobra Application
  completion  Generate the autocompletion script for the specified shell
  help        Help about any command
  init        Initialize a Cobra Application

Flags:
  -a, --author string    author name for copyright attribution (default "YOUR NAME")
      --config string    config file (default is $HOME/.cobra.yaml)
  -h, --help             help for cobra-cli
  -l, --license string   name of license for the project
      --viper            use Viper for configuration

Use "cobra-cli [command] --help" for more information about a command.
```

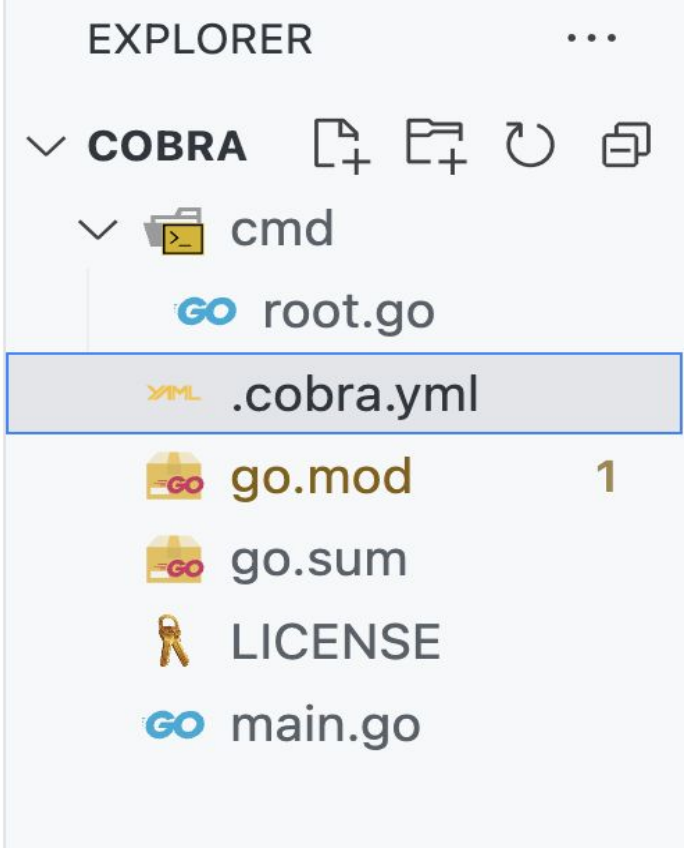
Port scanner

.cobra.yml

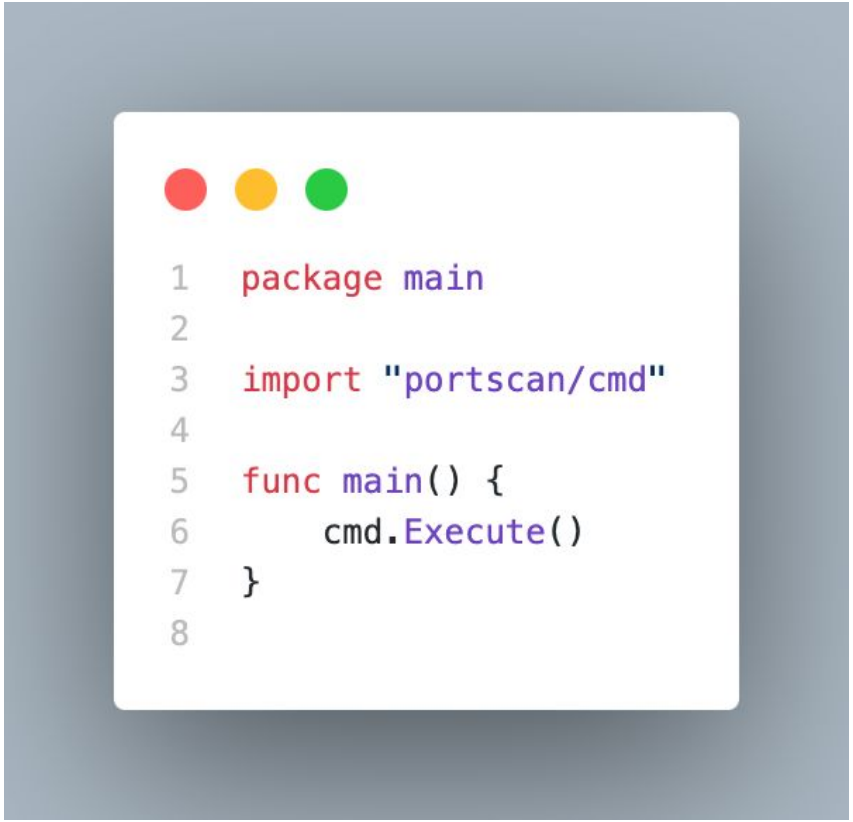
```yaml
author: İstanbul Go Konferansı
license: MIT
```

```
~/go/src/gokonf-cli/cobra cobra-cli init --config .cobra.yml
Using config file: .cobra.yml
Your Cobra application is ready at
/Users/kevsersirca/go/src/gokonf-cli/cobra
```

Port scanner



Port scanner

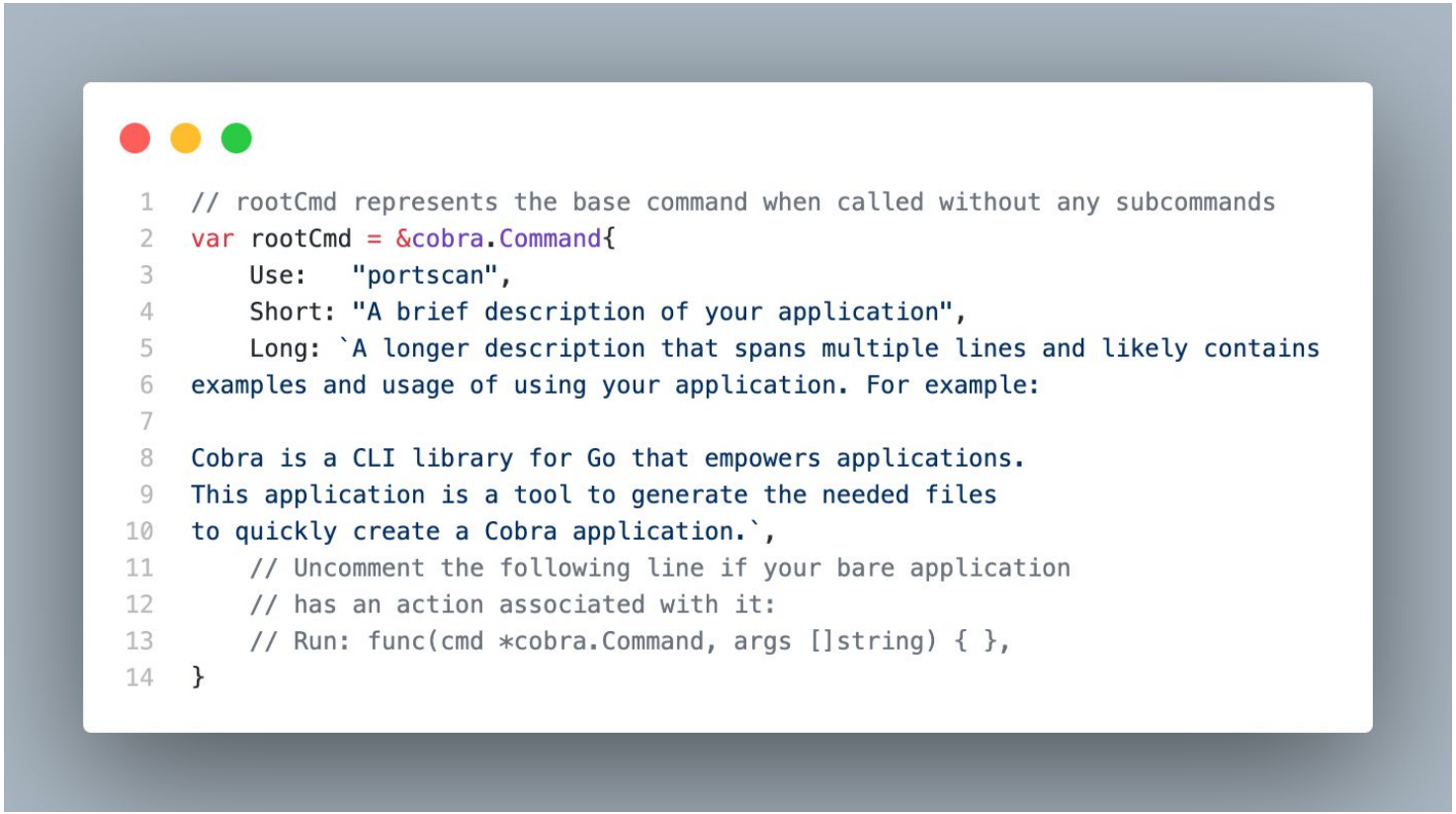


```go
package main

import "portscan/cmd"

func main() {
    cmd.Execute()
}

```

Port scanner

```go
// Execute adds all child commands to the root command and sets flags appropriately.
// This is called by main.main(). It only needs to happen once to the rootCmd.
func Execute() {
    err := rootCmd.Execute()
    if err != nil {
        os.Exit(1)
    }
}
```

Port scanner



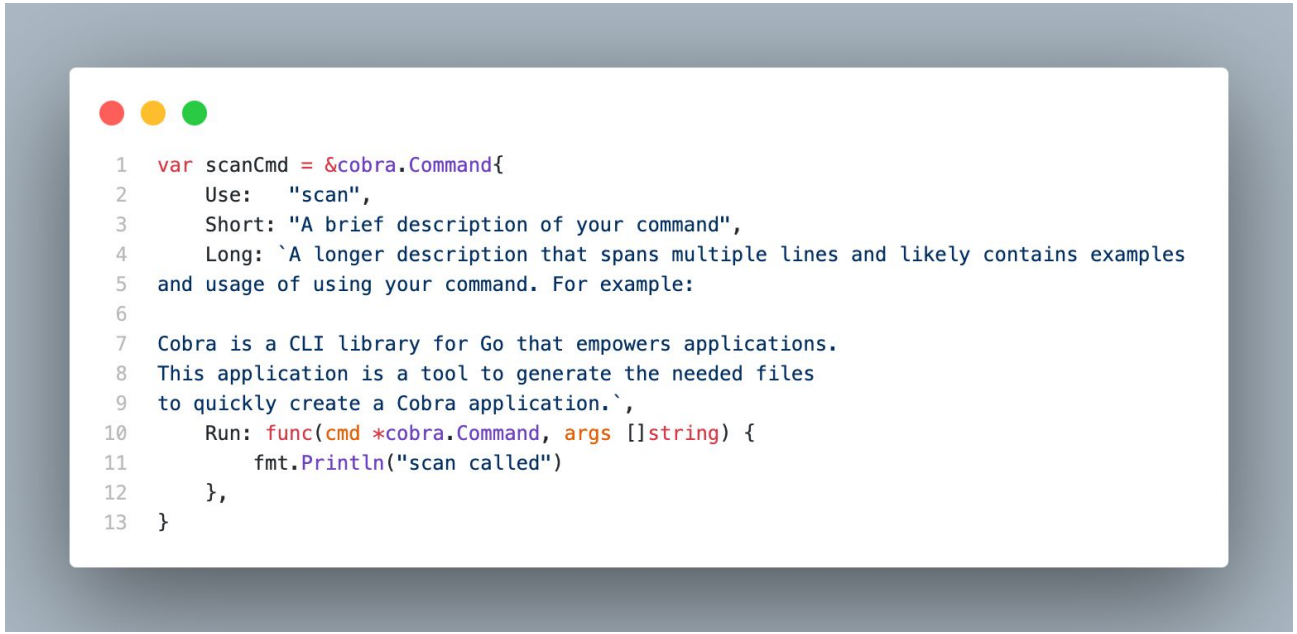
```go
// rootCmd represents the base command when called without any subcommands
var rootCmd = &cobra.Command{
    Use:   "portscan",
    Short: "A brief description of your application",
    Long: `A longer description that spans multiple lines and likely contains
examples and usage of using your application. For example:

Cobra is a CLI library for Go that empowers applications.
This application is a tool to generate the needed files
to quickly create a Cobra application.`,
    // Uncomment the following line if your bare application
    // has an action associated with it:
    // Run: func(cmd *cobra.Command, args []string) { },
}
```

Port scanner

```go
// rootCmd represents the base command when called without any subcommands
var rootCmd = &cobra.Command{
    Use:   "portscan",
    Short: "Fast TCP port scanner",
    Long: `portscan is a command-line tool for executing TCP port scans on a list of hosts.
It allows you to add, list, and delete hosts from the scan list.
portscan performs a port scan on specified TCP ports. You can customize the target ports using command-line flags.`,
    Version: "0.1.0",
}

```

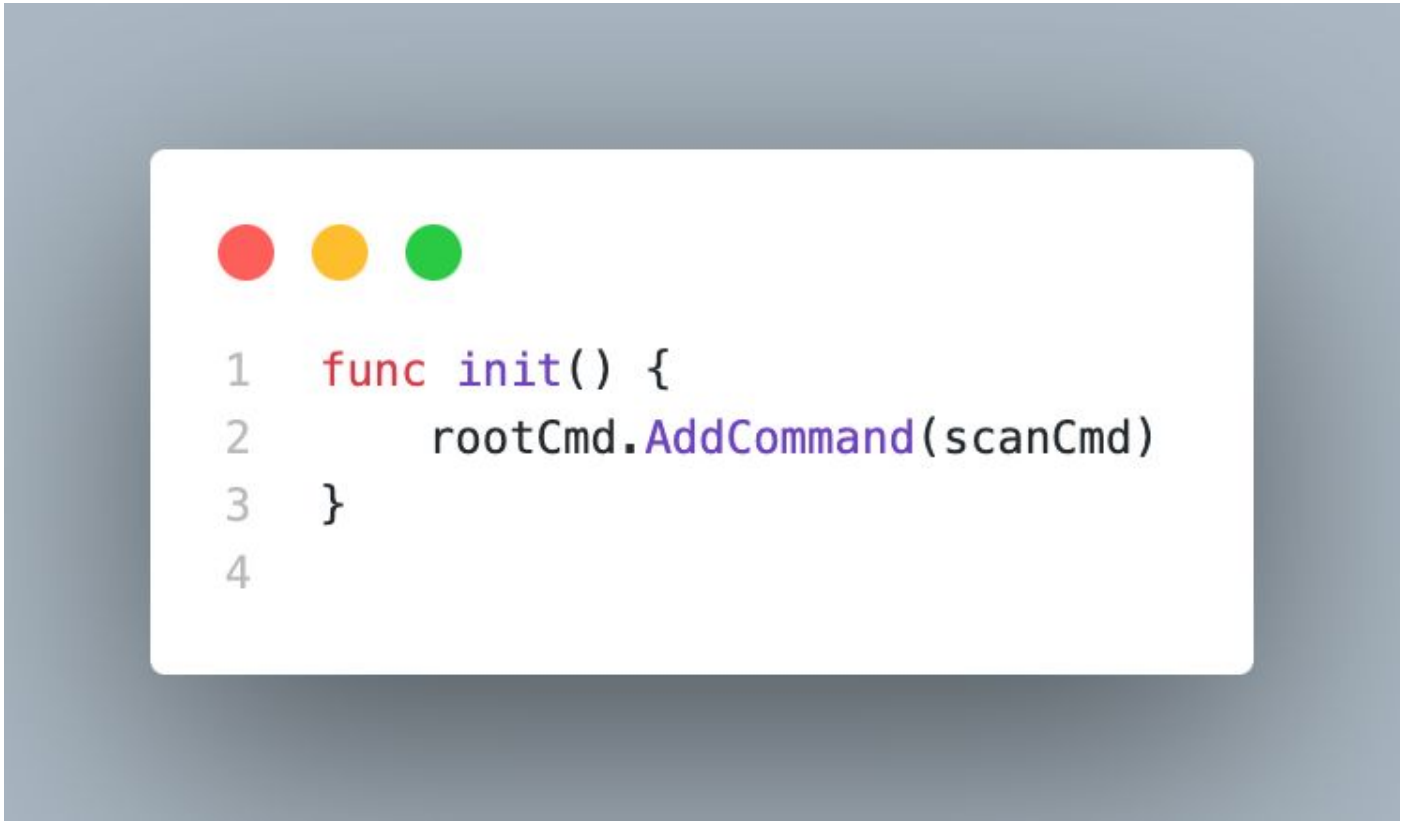
Port scanner

```
[cobra] cobra-cli add scan
scan created at /Users/kevsersirca/go/src/gokonf-cli/cobra
```

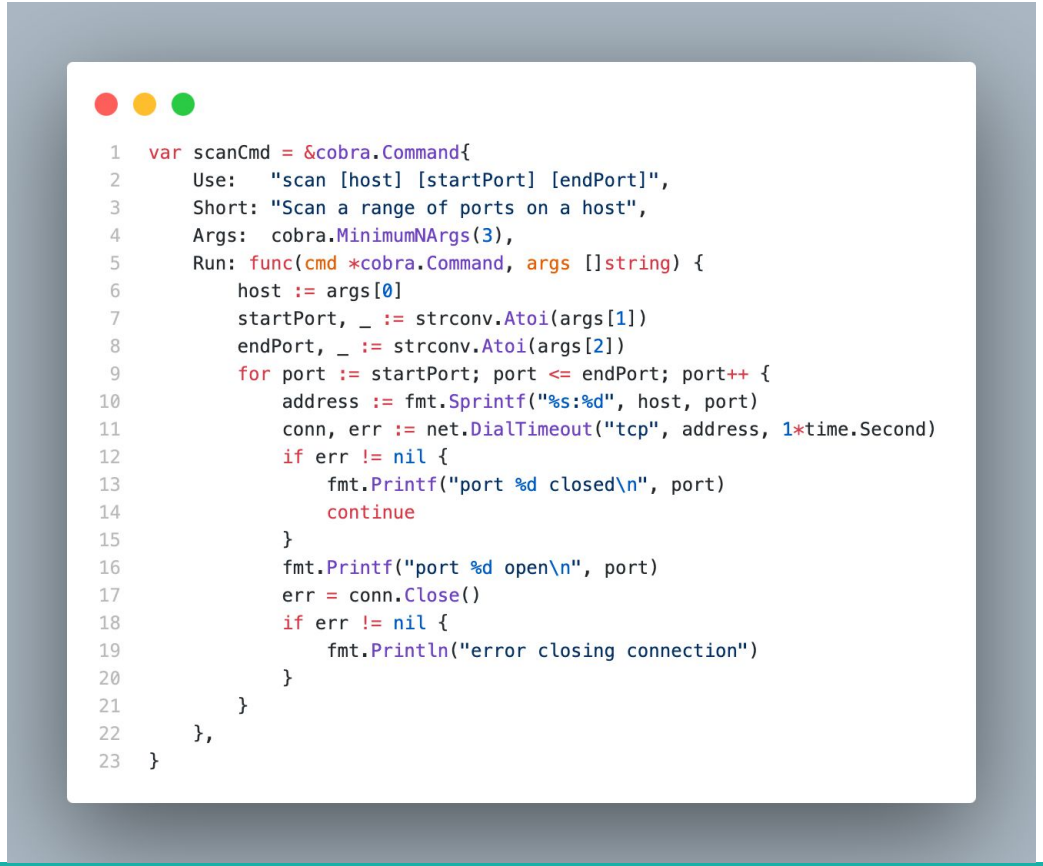


```go
var scanCmd = &cobra.Command{
    Use:   "scan",
    Short: "A brief description of your command",
    Long: `A longer description that spans multiple lines and likely contains examples
and usage of using your command. For example:

Cobra is a CLI library for Go that empowers applications.
This application is a tool to generate the needed files
to quickly create a Cobra application.`,
    Run: func(cmd *cobra.Command, args []string) {
        fmt.Println("scan called")
    },
}
```

Port scanner



Port scanner



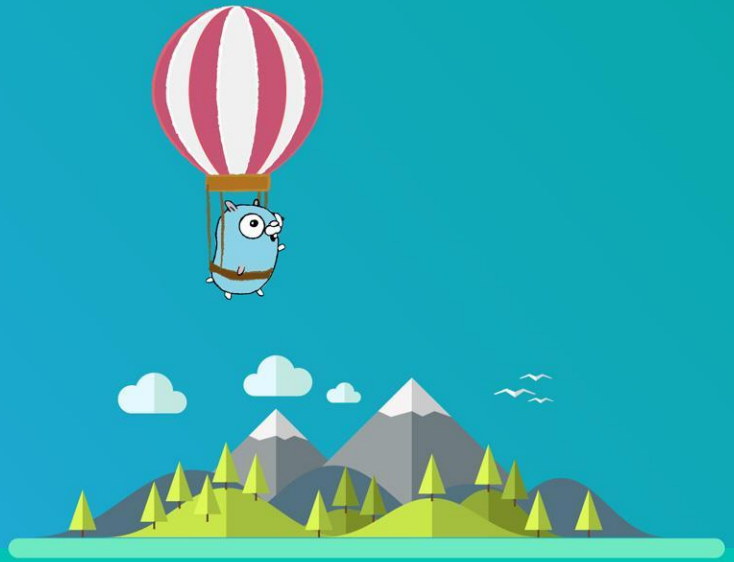
```go
var scanCmd = &cobra.Command{
    Use:   "scan [host] [startPort] [endPort]",
    Short: "Scan a range of ports on a host",
    Args:  cobra.MinimumNArgs(3),
    Run: func(cmd *cobra.Command, args []string) {
        host := args[0]
        startPort, _ := strconv.Atoi(args[1])
        endPort, _ := strconv.Atoi(args[2])
        for port := startPort; port <= endPort; port++ {
            address := fmt.Sprintf("%s:%d", host, port)
            conn, err := net.DialTimeout("tcp", address, 1*time.Second)
            if err != nil {
                fmt.Printf("port %d closed\n", port)
                continue
            }
            fmt.Printf("port %d open\n", port)
            err = conn.Close()
            if err != nil {
                fmt.Println("error closing connection")
            }
        }
    },
}
```

Port scanner

```
Ξ gokonf-cli/cobra → go run main.go scan 192.168.68.108 80 90
port 80 closed
port 81 closed
port 82 closed
port 83 closed
port 84 closed
port 85 closed
port 86 closed
port 87 closed
port 88 open
port 89 closed
port 90 closed
Ξ gokonf-cli/cobra →
```

Port scanner

```
⊗ 8404 ○  go run main.go scan 192.168.68.108 80
Error: requires at least 3 arg(s), only received 2
Usage:
  portscan scan [host] [startPort] [endPort] [flags]

Flags:
  -h, --help   help for scan

exit status 1
```



Thank you!

https://clig.dev/