L32-exercise-stack-resource-lister

List Stack resources status

Exercise

1. List all stack resources in a given stack name
2. Create functions in separate file
3. Use simple struct to hold result
4. Use os.Args to pass stack name as a parameter

Task 1

- Create `stacks.go` file

Task 2

- Create `main.go` file

Task 3

- Create simple cloudformation stack
- Test program with stack

Task - Overview

Directory structure

```
.
├── go.mod
├── go.sum
├── main
│   └── main.go
└── stacks.go
└── testdata
    └── template.yml
```

- Module name: `crlist`
- Main in its own directory
- `stacks.go` contains functions to list stack resources
- `testdata` contains CloudFormation template

Task 1 - Step 1

CloudFormation client

- create `stacks.go`
- add client initialization
- handle imports

```
 1: var Client *cloudformation.Client
 2:
 3: func init() {
 4:     cfg, err := config.LoadDefaultConfig(context.TODO())
 5:
 6:     if err != nil {
 7:         panic("unable to load SDK config, " + err.Error())
 8:     }
 9:
10:     Client = cloudformation.NewFromConfig(cfg)
11:
12: }
13:
```

Task 1 - Step 2

Result data type

- update `stacks.go`
- add `ResourceStatus` struct

```
1: // Type for holding logicalid and status
2: type ResourceStatus struct {
3:     LogicalID string
4:     Status    string
5: }
6:
```

Task 1 - Step 3

Define function

- update `stacks.go`
- add function `GetStatus`

```
1: func GetStatus(client *cloudformation.Client, stackname *string) (*[]ResourceStatus, error) {
```

Task 1 - Step 4

Call CloudFormation service

- update `stacks.go`
- `DescribeStackResources` returns status of all resources in a stack
- add AWS call
- handle errors
  - you can use `fmt` package for printing errors
  - you can use `log` package for logging errors
  - you can use `slog` package for structured logging

```
 1:     states := &[]ResourceStatus{}
 2:     // Get resource status for stack stackname
 3:     parms := &cloudformation.DescribeStackResourcesInput{
 4:         StackName: stackname,
 5:     }
 6:     resp, err := client.DescribeStackResources(context.Background(), parms)
 7:     if err != nil {
 8:         slog.Error("Error in getting stack status", err)
 9:         return nil, err
10:     }
```

Task 1 - Step 5

Collect results

- update `stacks.go`
- `resp.StackResources` contains list of resources
- See `DescribeStackResourcesOutput` struct
- append to the states slice
- GO will maintain the local variable `states`

```
1:      for _, resource := range resp.StackResources {
2:          *states = append(*states, ResourceStatus{
3:              LogicalID: *resource.LogicalResourceId,
4:              Status:    string(resource.ResourceStatus),
5:          })
6:      }
7:      return states, nil
```

Task 2 - Step 6

Get arguments from command line

- update `main/main.go`

```
1:      argLength := len(os.Args[1:])
2:      if argLength == 0 {
3:          fmt.Printf("Please provide a stack name as an argument\n")
4:          os.Exit(1)
5:      }
6:      stackName := os.Args[1]
```

Task 2 - Step 7

Call the prepared function "GetStatus"

- update `main/main.go`

```
1:      resources, err := crlist.GetStatus(crlist.Client,&stackName)
2:      if err != nil {
3:          panic(err)
4:      }
```

Task 2 - Step 8

Display the results

- update `main/main.go`

```
1:      fmt.Printf("%-32s %-32s \n", "Logical ID",  "Status")
2:      fmt.Printf("%-32s %-32s\n", "----------", "-----------")
3:      for _, resource := range *resources {
4:          fmt.Printf("%-32s %-32s\n", resource.LogicalID, resource.Status)
5:      }
```

Task 2 - Step 9

Optional

Challenge: get command line arguments with flags

Task 3 - Step 10

Setup test environment

Get the testing CloudFormation template

- load file from `github/megaproaktiv/aws-go-sdk-v2/L32-exercise-stack-resource-lister/code-task1/testdata/template.yml`
- Create stack with the AWS CLI:

```
aws cloudformation create-stack --stack-name somecheapressources --template-body file://template.yml --capabilities CAPABILITY_IAM
```

You need to have a working AWS connection and the AWS CLI installed.

- Wait for stack completion

Task 3 - Step 11

Test the program

- Start the program with the stack name as argument

```
go run main/main.go somecheapressource
```

- Output should be:

```
Logical ID                      Status
----------                      -----------
MyLambdaFunction                CREATE_COMPLETE
MyLambdaRole                    CREATE_COMPLETE
MySNSTopic                      CREATE_COMPLETE
```

Congratulations! You have finished the exercise. Now let`s clean up the test environment as the last step.

Task 3 - Step 12

TearDown test environment

- Delete CloudFormation stack

```
aws cloudformation delete-stack --stack-name somecheapressources
```

(Optional) start the program while the stack is deleting

```
go run main/main.go somecheapressources
```

Output is like:

```
Logical ID                          Status
----------                          -----------
MyLambdaFunction                    DELETE_COMPLETE
MyLambdaRole                        DELETE_COMPLETE
MySNSTopic                          DELETE_IN_PROGRESS
``
```

Task 3 - Step 13

(Optional) Handle errors

- When you run the program with a non existing stack name, you get an error message:

```
2023/05/09 08:57:57 ERROR Error in getting stack status
!BADKEY="operation error CloudFormation: DescribeStackResources,
https response error StatusCode: 400, RequestID: 5863dc4f-37a7-460f-a67f-ceeb009f4f24,
api error ValidationError: Stack with id somecheapressources does not exist"
```

- Update the program to handle this error