Converting Your Existing Web App to AWS Lambda The Easy Way

What is AWS Lambda?

- AWS’s function as a service offering
- Often referred to as serverless
- Event driven
  - API Gateway, S3, SES, DynamoDB streams and more
- Billed per 100ms of execution
  - $0.000000834 per 100ms for 512MB
  - 800,000 free seconds at 512MB
- Supports: Go, Node.js, Python, Java and C#

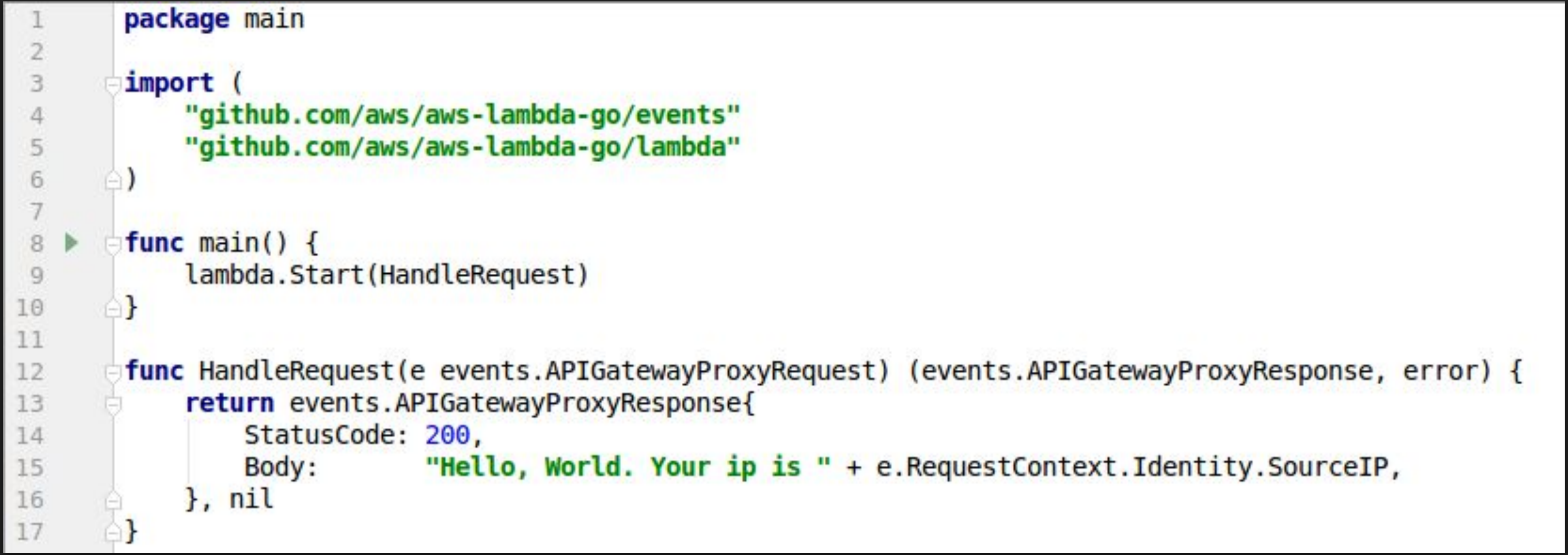
Benefits

- Utility based pricing
- Automatic “unlimited” scaling up
- Scale to zero
- No servers to manage
- Built in high availability and fault tolerance

Drawbacks

- Cold start time
- Vendor lock in
  - Can’t write standard HTTP handler funcs
- Not suitable for all apps
  - Check if yours is at https://servers.lol
- Higher costs than equivalent EC2 instance

Hello World Example



```go
package main

import (
    "github.com/aws/aws-lambda-go/events"
    "github.com/aws/aws-lambda-go/lambda"
)

func main() {
    lambda.Start(HandleRequest)
}

func HandleRequest(e events.APIGatewayProxyRequest) (events.APIGatewayProxyResponse, error) {
    return events.APIGatewayProxyResponse{
        StatusCode: 200,
        Body:       "Hello, World. Your ip is " + e.RequestContext.Identity.SourceIP,
    }, nil
}
```

APIGatewayProxyrequest → http.Request

http.ReponseWriter → APIGatewayProxyResponse

Libraries

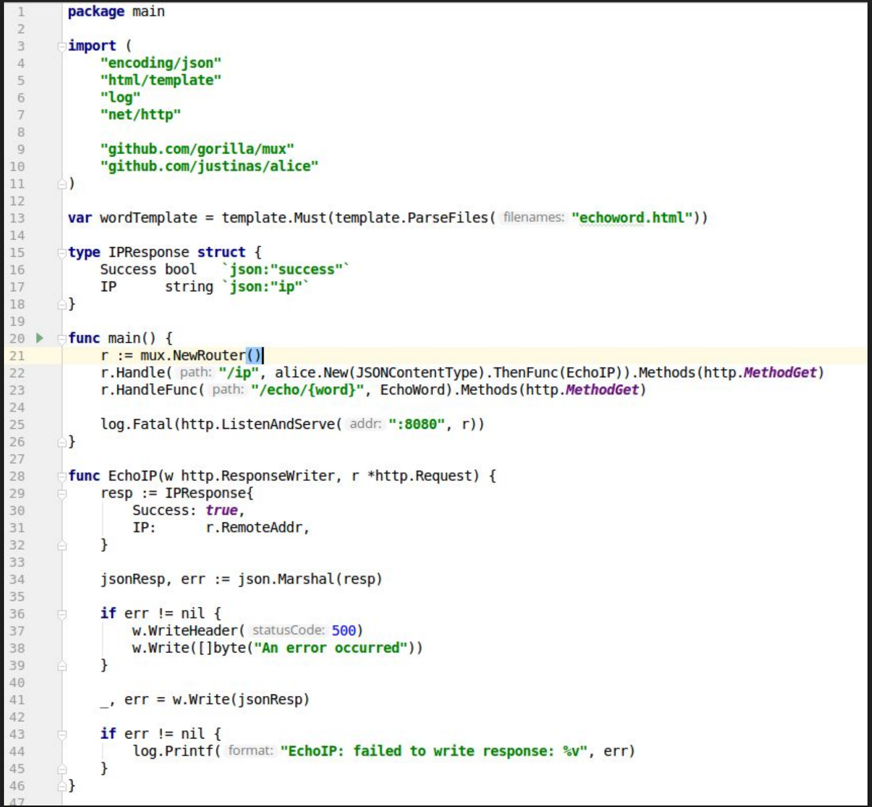
- github.com/apex/gateway
- github.com/haydenwoodhead/gateway
- github.com/awslabs/aws-lambda-go-api-proxy
- github.com/iamatypeofwalrus/shim

Convert

0. Move State

- Lambda executions are ephemeral
- We cannot store long lived data in memory
- Lambda executions scale horizontally
- Cannot rely on hitting the same container again
- We do get 512MB in /tmp
- Will mention db options later

1. Starting Point

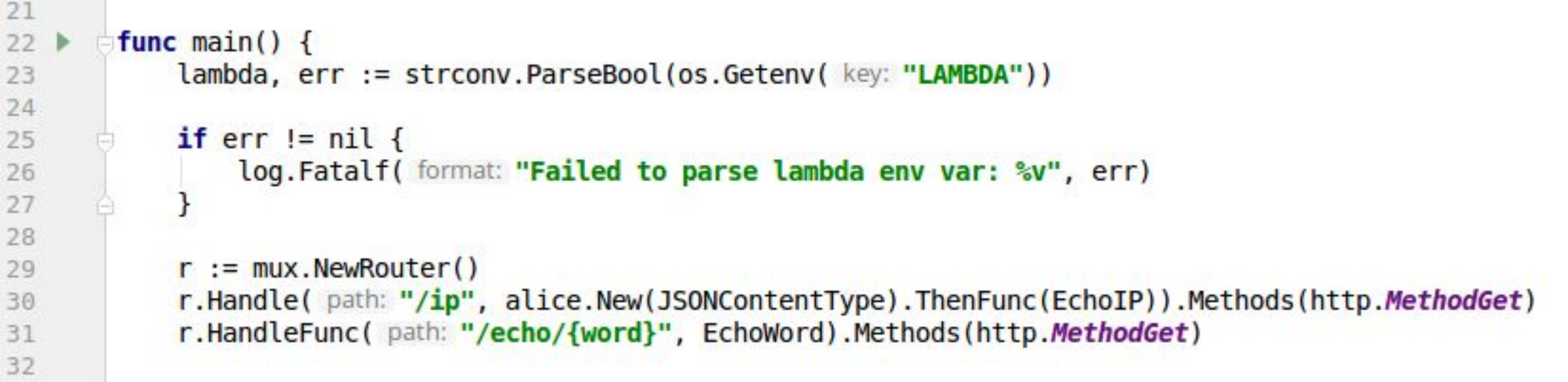


```go
package main

import (
    "encoding/json"
    "html/template"
    "log"
    "net/http"

    "github.com/gorilla/mux"
    "github.com/justinas/alice"
)

var wordTemplate = template.Must(template.ParseFiles( filenames: "echoword.html"))

type IPResponse struct {
    Success bool   `json:"success"`
    IP      string `json:"ip"`
}

func main() {
    r := mux.NewRouter()
    r.Handle( path: "/ip", alice.New(JSONContentType).ThenFunc(EchoIP)).Methods(http.MethodGet)
    r.HandleFunc( path: "/echo/{word}", EchoWord).Methods(http.MethodGet)

    log.Fatal(http.ListenAndServe( addr: ":8080", r))
}

func EchoIP(w http.ResponseWriter, r *http.Request) {
    resp := IPResponse{
        Success: true,
        IP:      r.RemoteAddr,
    }

    jsonResp, err := json.Marshal(resp)

    if err != nil {
        w.WriteHeader( statusCode: 500)
        w.Write([]byte("An error occurred"))
    }

    _, err = w.Write(jsonResp)

    if err != nil {
        log.Printf( format: "EchoIP: failed to write response: %v", err)
    }
}
```

Fooservice:

- Gorilla Mux for routing
- JSON endpoint returning IP address
- HTML endpoint echoing words
    - With Template

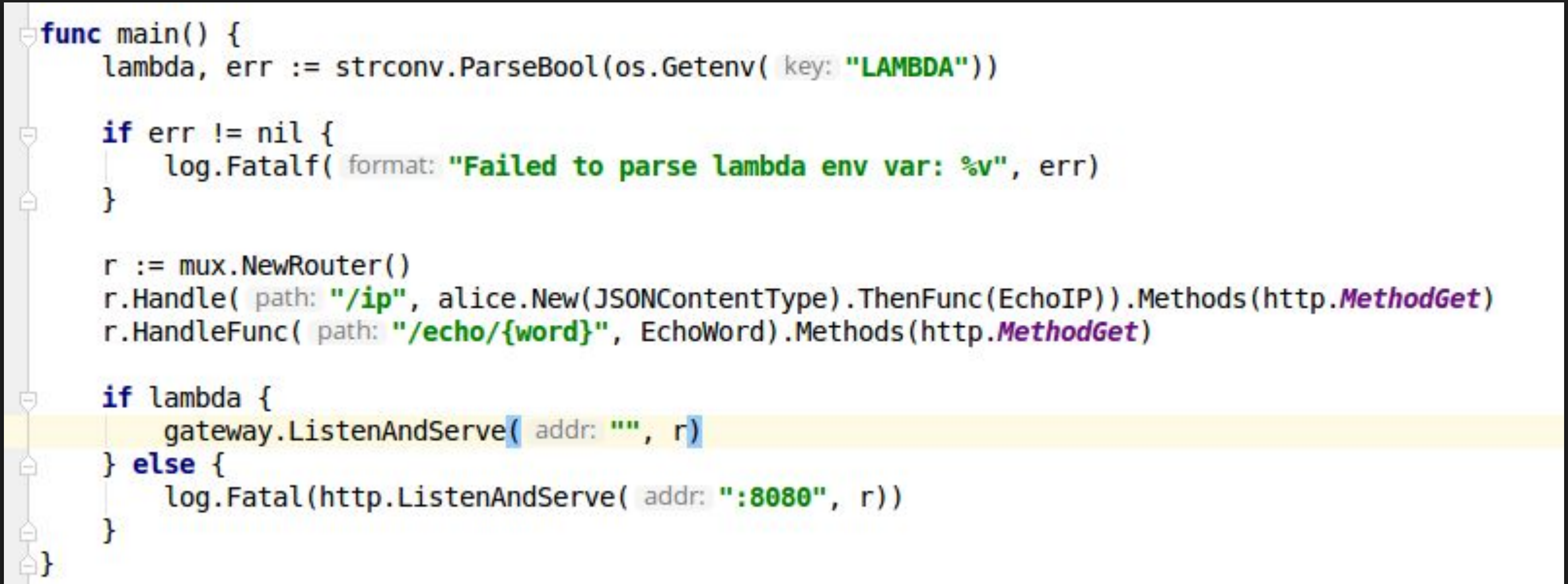
2. Add Environment Variable



```
func main() {
    lambda, err := strconv.ParseBool(os.Getenv( key: "LAMBDA"))

    if err != nil {
        log.Fatalf( format: "Failed to parse lambda env var: %v", err)
    }

    r := mux.NewRouter()
    r.Handle( path: "/ip", alice.New(JSONContentType).ThenFunc(EchoIP)).Methods(http.MethodGet)
    r.HandleFunc( path: "/echo/{word}", EchoWord).Methods(http.MethodGet)

```

I like to do this so we can run the standard HTTP server if we’re not using Lambda

3. Call gateway.ListenAndServe



```
func main() {
	lambda, err := strconv.ParseBool(os.Getenv( key: "LAMBDA"))

	if err != nil {
		log.Fatalf( format: "Failed to parse lambda env var: %v", err)
	}

	r := mux.NewRouter()
	r.Handle( path: "/ip", alice.New(JSONContentType).ThenFunc(EchoIP)).Methods(http.MethodGet)
	r.HandleFunc( path: "/echo/{word}", EchoWord).Methods(http.MethodGet)

	if lambda {
		gateway.ListenAndServe( addr: "", r)
	} else {
		log.Fatal(http.ListenAndServe( addr: ":8080", r))
	}
}
```

That is it. Simply call gateway if we’re running in Lambda or the standard library if we’re not.

Database

Database Options

- RDS
- DyanmoDB
- Aurora Serverless

DynamoDB

- AWS’ managed No SQL DB
- Key/value and document
- Single digit latency
- Scales for you
  - Caveat on scaling down. Must have consistent downwards trend to scale down
- Go library as part of AWS SDK

DynamoDB cont.

- Items have a
  - Partition Key
  - Sort Key
- Item limits
  - 400kb max size (including data in local secondary index)
  - 32 levels of nesting
- Partition keys should be well distributed
- AWS’ states that well designed applications use only one table

UserID (Partition)	Name (Sort)	Address	Phone
ce487bf1	Bobby Tables	123 Main St	555555555
	Homer Simpson	125 Fake Ave	555555556

Read and Write Capacity

Read Capacity unit:

- Strongly consistent: 1x 4kb items per second
- Eventually consistent: 2x 4kb item per second

Write capacity unit: 1kb item per second

Try not to have hot keys or partitions.

Go read:
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ProvisionedThroughput.html

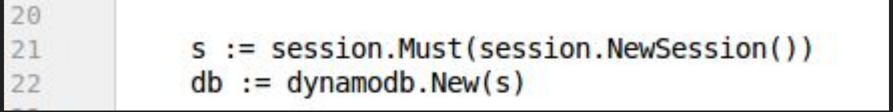
Secondary Index

- Allow querying on other attributes
- “Virtual table”
- Global Secondary Index
  - Partition and sort key seperate from main table
  - Can be created at any time
  - Define read and write units separately from main table
- Local Secondary Index
  - Partition key same but different sort key
  - Must be created at table creation
- Copy values into this new table
  - Uses duplicate storage
- Max of 5 GSI and 5 LSI

Example GSI - Query on Address

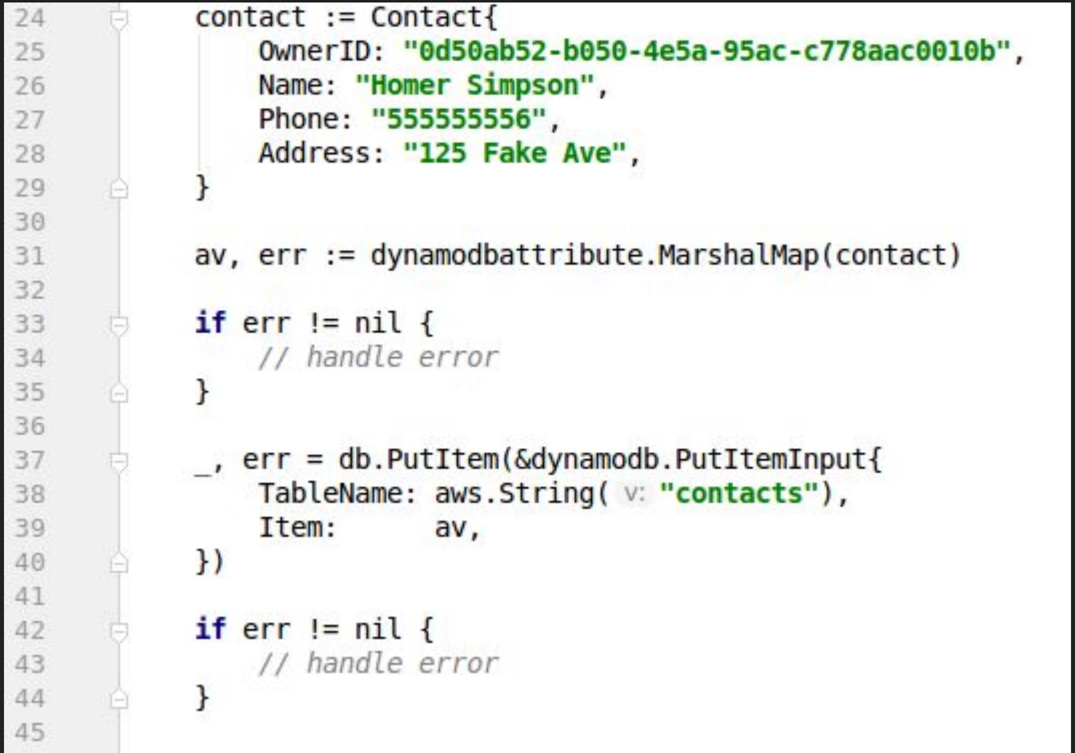
Address (Partition)	UserID	Name
123 Main St	ce487bf1	Bobby Tables
125 Fake Ave	ce487bf1	Homer Simpson

Starting a Session



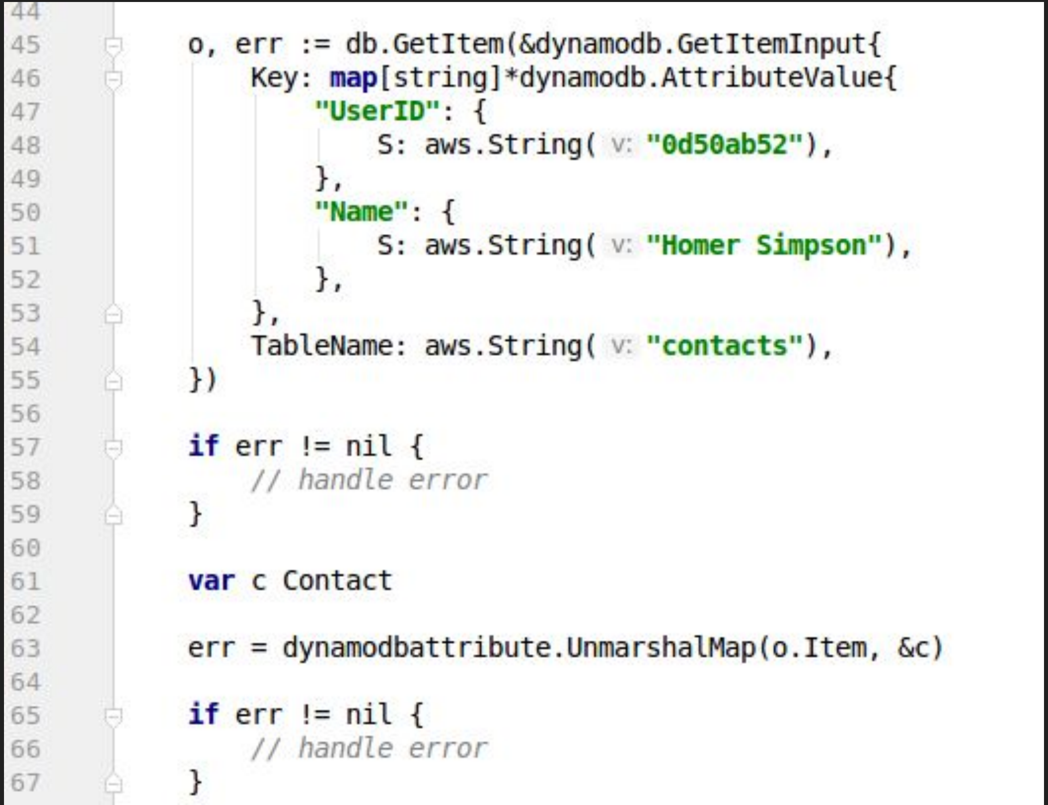
```
	s := session.Must(session.NewSession())
	db := dynamodb.New(s)
```

Save an Item



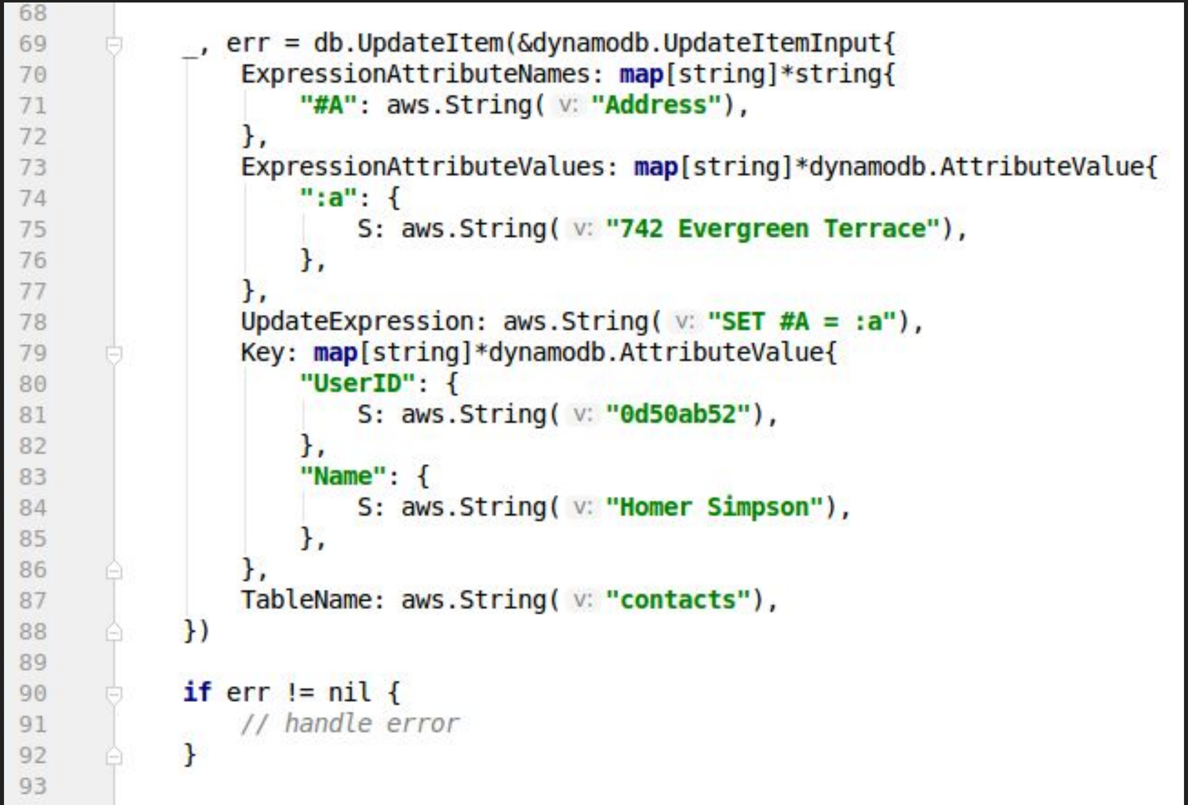
```go
contact := Contact{
    OwnerID: "0d50ab52-b050-4e5a-95ac-c778aac0010b",
    Name: "Homer Simpson",
    Phone: "555555556",
    Address: "125 Fake Ave",
}

av, err := dynamodbattribute.MarshalMap(contact)

if err != nil {
    // handle error
}

_, err = db.PutItem(&dynamodb.PutItemInput{
    TableName: aws.String( v: "contacts"),
    Item:      av,
})

if err != nil {
    // handle error
}
```

Get an Item



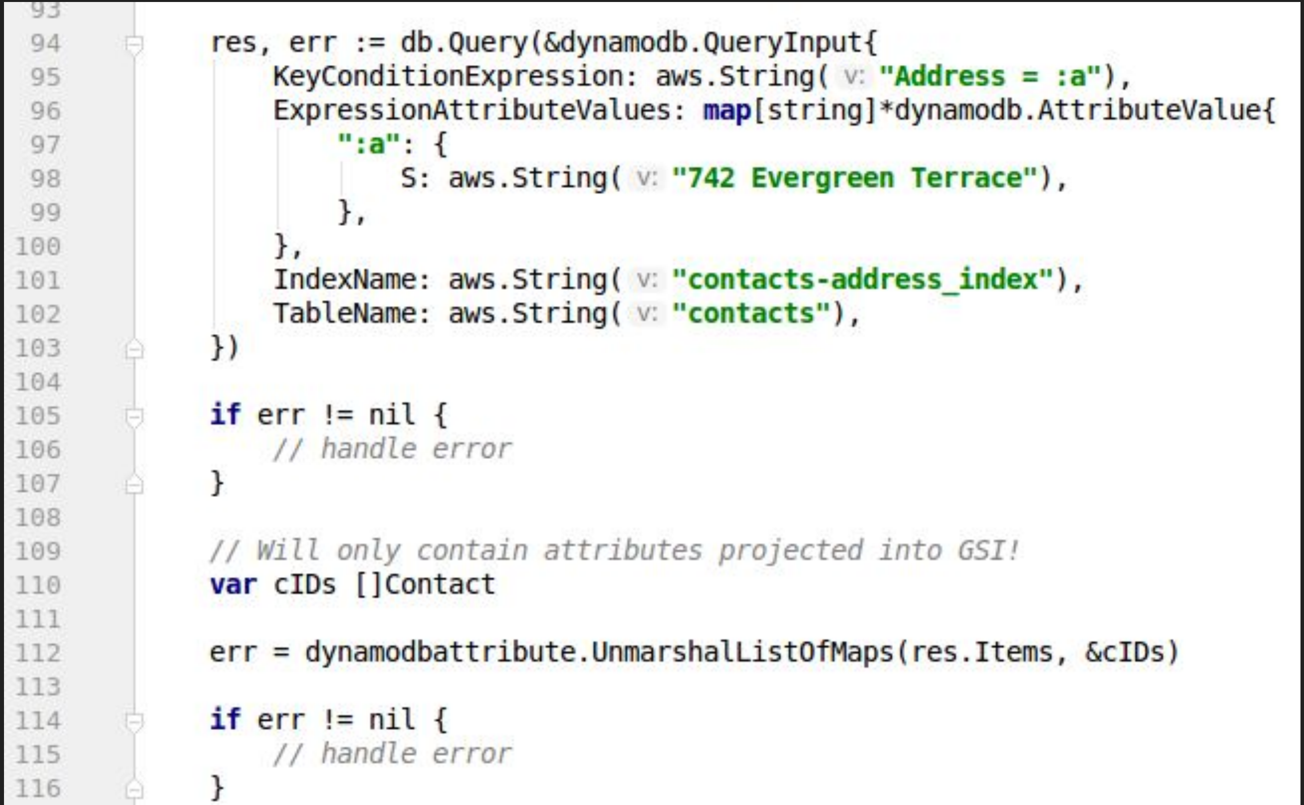
```go
o, err := db.GetItem(&dynamodb.GetItemInput{
    Key: map[string]*dynamodb.AttributeValue{
        "UserID": {
            S: aws.String( v: "0d50ab52"),
        },
        "Name": {
            S: aws.String( v: "Homer Simpson"),
        },
    },
    TableName: aws.String( v: "contacts"),
})

if err != nil {
    // handle error
}

var c Contact

err = dynamodbattribute.UnmarshalMap(o.Item, &c)

if err != nil {
    // handle error
}
```

Update an Item



```
    _, err = db.UpdateItem(&dynamodb.UpdateItemInput{
        ExpressionAttributeNames: map[string]*string{
            "#A": aws.String( v: "Address"),
        },
        ExpressionAttributeValues: map[string]*dynamodb.AttributeValue{
            ":a": {
                S: aws.String( v: "742 Evergreen Terrace"),
            },
        },
        UpdateExpression: aws.String( v: "SET #A = :a"),
        Key: map[string]*dynamodb.AttributeValue{
            "UserID": {
                S: aws.String( v: "0d50ab52"),
            },
            "Name": {
                S: aws.String( v: "Homer Simpson"),
            },
        },
        TableName: aws.String( v: "contacts"),
    })

    if err != nil {
        // handle error
    }
```

Query an Item on GSI



```go
res, err := db.Query(&dynamodb.QueryInput{
    KeyConditionExpression: aws.String( v: "Address = :a"),
    ExpressionAttributeValues: map[string]*dynamodb.AttributeValue{
        ":a": {
            S: aws.String( v: "742 Evergreen Terrace"),
        },
    },
    IndexName: aws.String( v: "contacts-address_index"),
    TableName: aws.String( v: "contacts"),
})

if err != nil {
    // handle error
}

// Will only contain attributes projected into GSI!
var cIDs []Contact

err = dynamodbattribute.UnmarshalListOfMaps(res.Items, &cIDs)

if err != nil {
    // handle error
}
```

Delete an Item

- Use TTL’s if possible. They don’t use read or write capacity!
- Basically same as the Get Item operation

Other notes

- DynamoDB allows for conditional writes
- AWS Go SDK: https://docs.aws.amazon.com/sdk-for-go/v1/developer-guide/using-dynamodb-with-go-sdk.html
- Go examples: https://docs.aws.amazon.com/sdk-for-go/v1/developer-guide/using-dynamodb-with-go-sdk.html
- Best Practices for DynamoDB https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-practices.html
- Advanced Design Patterns for DynamoDB https://www.youtube.com/watch?v=jzeKPKpucS0

Deploy

Deployment

- Setup repeatable deployment using AWS Serverless Application Model
  - Superset of Cloudformation. AWS's infrastructure as code service
  - Provides convenience types for lambda, apigateway and dynamodb
  - https://github.com/awslabs/serverless-application-model
- Deploy with AWS CLI and SAM CLI
  - https://aws.amazon.com/cli/
  - https://github.com/awslabs/aws-sam-cli
- Assume you already have these setup

1. Build and zip up our binary

- $GOOS=linux go build .
- $zip main.zip main echoword.html

Our html template

2. Create a template

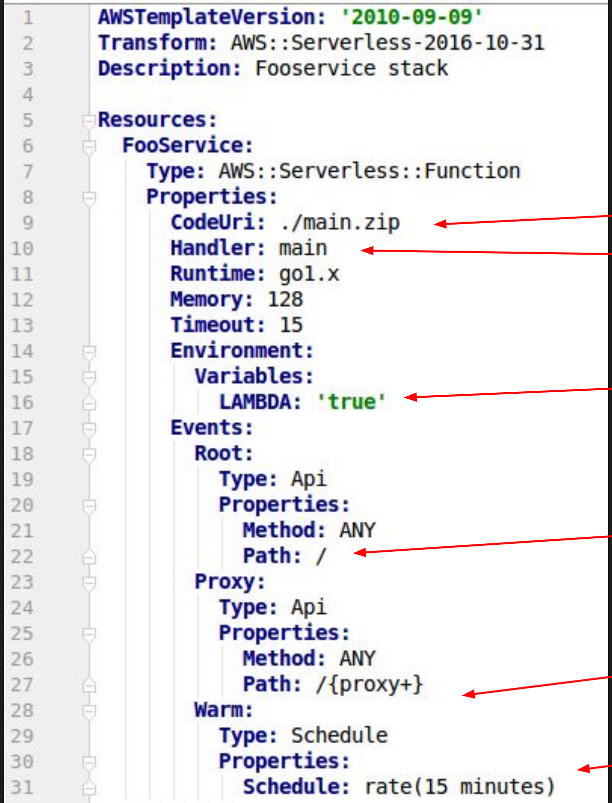
1. $touch template.yml
2. Edit this file

4. Create an S3 bucket

Need an s3 bucket to hold our deployment object

```
$aws s3api create-bucket --bucket somename --region ap-southeast-2
```

3. Write Cloudformation Template



```yaml
AWSTemplateVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: Fooservice stack

Resources:
  FooService:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./main.zip
      Handler: main
      Runtime: go1.x
      Memory: 128
      Timeout: 15
      Environment:
        Variables:
          LAMBDA: 'true'
      Events:
        Root:
          Type: Api
          Properties:
            Method: ANY
            Path: /
        Proxy:
          Type: Api
          Properties:
            Method: ANY
            Path: /{proxy+}
        Warm:
          Type: Schedule
          Properties:
            Schedule: rate(15 minutes)
```

- Location of our zip file
- Name of binary
- Need to use ‘’ so we parse the variable correctly
- Wildcard path doesn’t match /
- Wildcard path
- Calls our function every 15 to keep it warm

5. Package

Uploads file to s3 and creates a packaged template for you.

```
$sam package --template-file template.yml --output-template-file output.yml \
        --s3-bucket s3-bucket-name
```

6. Deploy

Actually deploy code to lamda. Capabilities flag acknowledges that this will create an IAM user.

```
$sam deploy --template-file output.yml --stack-name foostack \
        --capabilities CAPABILITY_IAM
```

SAM Links

https://github.com/awslabs/serverless-application-model

https://docs.aws.amazon.com/lambda/latest/dg/test-sam-cli.html

https://aws.amazon.com/documentation/cloudformation/

Deployed!!

Gotchas

- Goroutines don’t continue executing after we return a response to API Gateway

Questions?