



Conf42: Golang 2022

Go Serverless!



Savas Ziplies

<https://elipZis.com>

@insanitydesign

AHOY



Dipl. Inf. Savas Simon Ziplies

Managing Director @ elipZis  

Passionate Software Engineer, Technical Lead & Director for >15 years in research, project and product development, managing teams and server clouds alike.

Golang, Java, PHP, C#, AWS, Azure, Svelte etc.

Technical Lead @ Ubisoft Düsseldorf
Producer @ Bigpoint Hamburg
Researcher @ ATB-Bremen
and more...



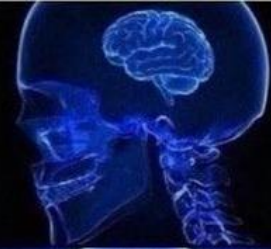
But I have a Server!

Why Serverless?

Defining Serverless...

- ...is hard!
 - SaaS such as Firebase, Auth0 etc.
 - Backend as a Service (BaaS)
- Not caring about Infrastructure
 - „The Cloud“: Just Servers of others!
 - But you don't own/know about Servers
 - No dedicated servers; No instances
- Functions as a Service (FaaS)
 - Think in App Functionalities
 - Simplicity first!

WEBHOSTING



**ROOT
SERVER**



**AUTO SCALING
EC2 INSTANCES**

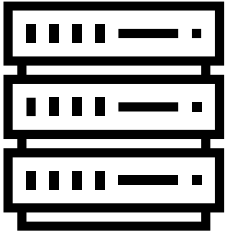


SERVERLESS



imgflip.com

What's Serverless?



Infrastructure
as a Service



High-availability
and (endless) scalable



Only Runs when
needed
\$\$\$



Focus on
Development
(only a bit of Ops)



The Market

Serverless Now!

Vendor Cloud Support & Frameworks & more ...



Azure Functions



AWS Lambda



Google Cloud Functions



IBM Cloud Functions



Fn Project



ΔPEX
serverless architecture



SERVERLESS NOW!

<https://www.netlify.com/press/>
<https://vercel.com/design>



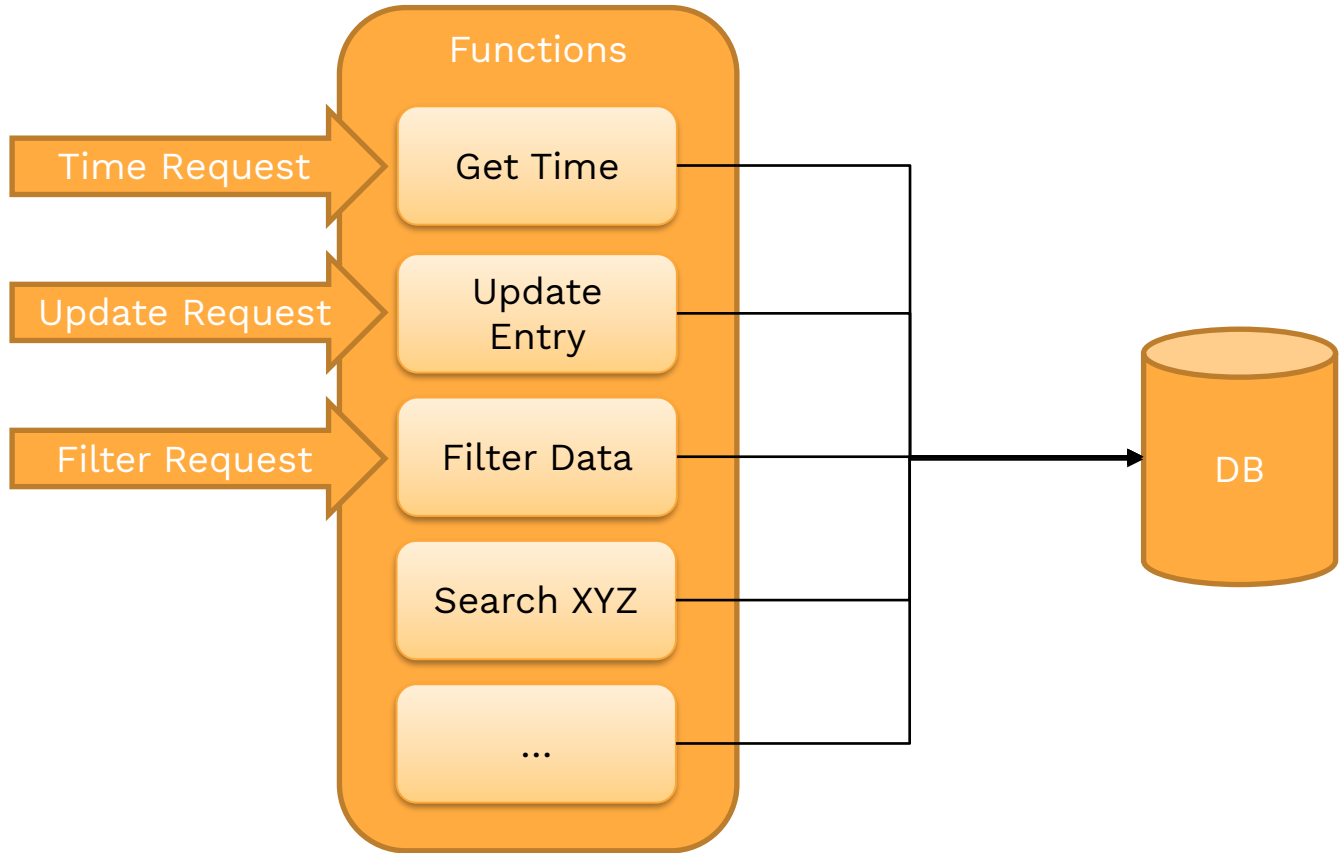


Going Serverless

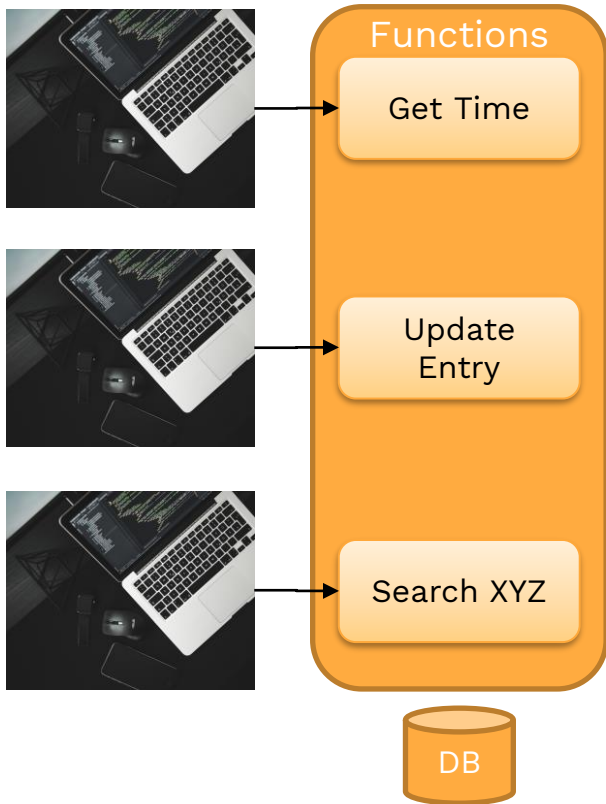
Let's Go



Simple Serverless Architecture



Develop & Deploy a Simple Serverless Architecture



./project

/time

/func.ts

/update

/func.go

/search

/func.py



<https://.../api/time>



<https://.../api/update>



<https://.../api/search>



Native

```
package main

import (
    "encoding/json"
    "fmt"
    "html"
    "net/http"
)

func HelloWorld(w http.ResponseWriter, r *http.Request) {
    var d struct {
        Name string `json:"name"`
    }
    if err := json.NewDecoder(r.Body).Decode(&d); err != nil {
        fmt.Fprint(w, "Hello World!")
        return
    }
    fmt.Fprint(w, html.EscapeString("Hello " + d.Name))
}
```

No main()
Only the
Function

Native

```
package main

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

type MyEvent struct {
    Name string `json:"name"`
}

func HandleRequest(ctx context.Context, name MyEvent)
(string, error) {
    return fmt.Sprintf("Hello %s!", name.Name), nil
}

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

Lambda Start to Handle Requests

- Create a (free) Account
 - Nearly all Providers offer free tiers
- Install AWS CLI, login and configure
 - You'll need it ;)
- Build, Zip, Deploy
 - Profit!
- Rinse & Repeat
 - Function by Function

```
// Build & Zip
:~$ go get github.com/aws/aws-lambda-go/lambda
:~$ GOOS=linux go build main
:~$ zip helloworld.zip main








// Execution Permission
:~$ aws iam create-role --role-name lambda-ex --assume-role-policy-
document file://trust-policy.json
:~$ aws iam attach-role-policy --role-name lambda-ex --policy-arn
arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole

// Create & Invoke
:~$ aws lambda create-function --function-name hello-world --zip-file
fileb://helloworld.zip --handler helloworld --runtime go1.x --role
arn:aws:iam::0123456789:role/lambda-ex
:~$ aws lambda invoke --function-name hello-world --cli-binary-format
raw-in-base64-out --payload '{"name": "World"}' helloworld.json

// Success!
:~$ cat helloworld.json
{„statusCode“:200, „body“:“\“Hello World!\“”}
```

Pros & Cons



- +  Quick to start
- +  Simple & easy development, Function by Function
- +  Single-purpose design pattern easy to follow
-  Easily Vendor locked
-  Single-purpose functions „too-micro-service“
-  Local development and testing can be complicated
-  Not „native“ to the known development structure, e.g. Routing

FaaS natively may fulfil specific Use Cases!

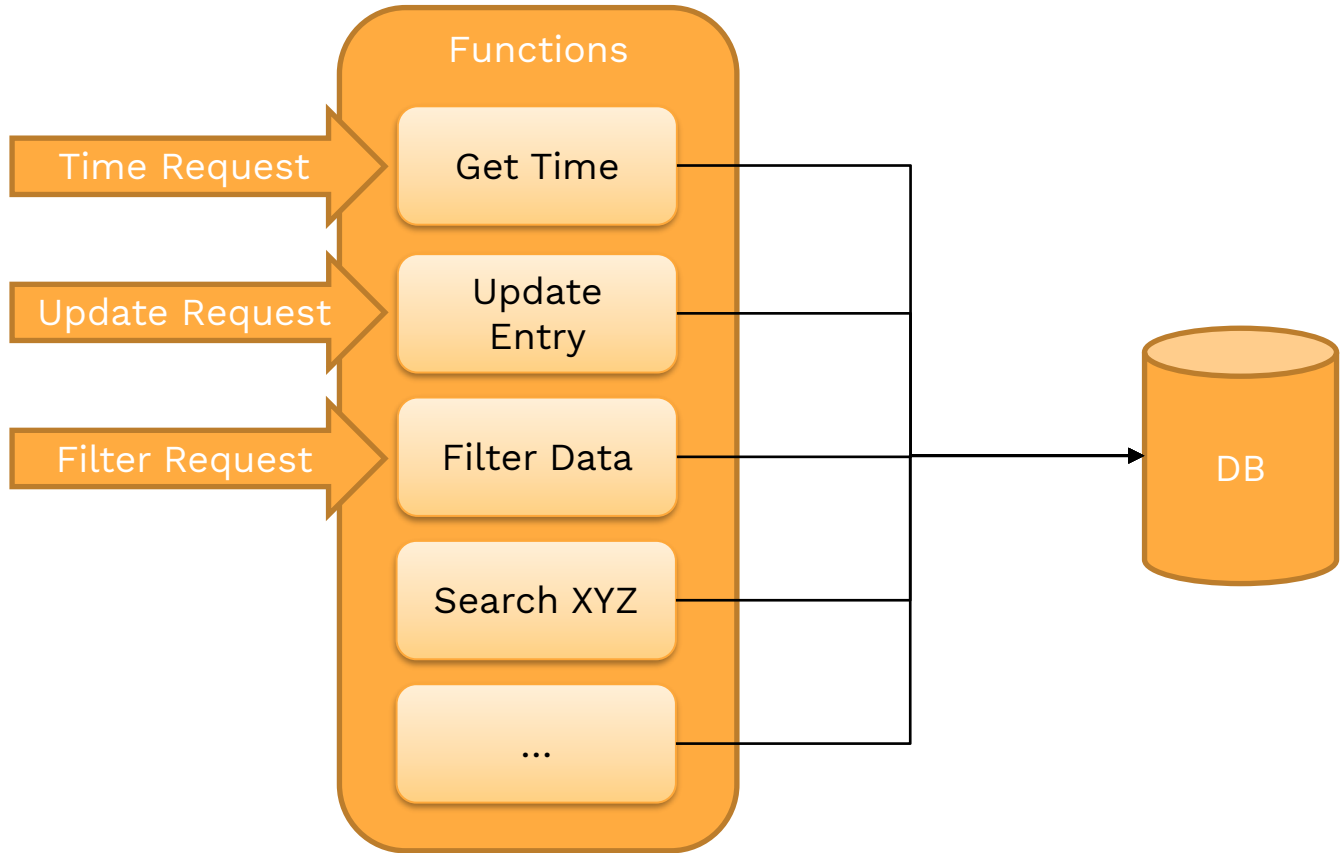


Serverless Next!

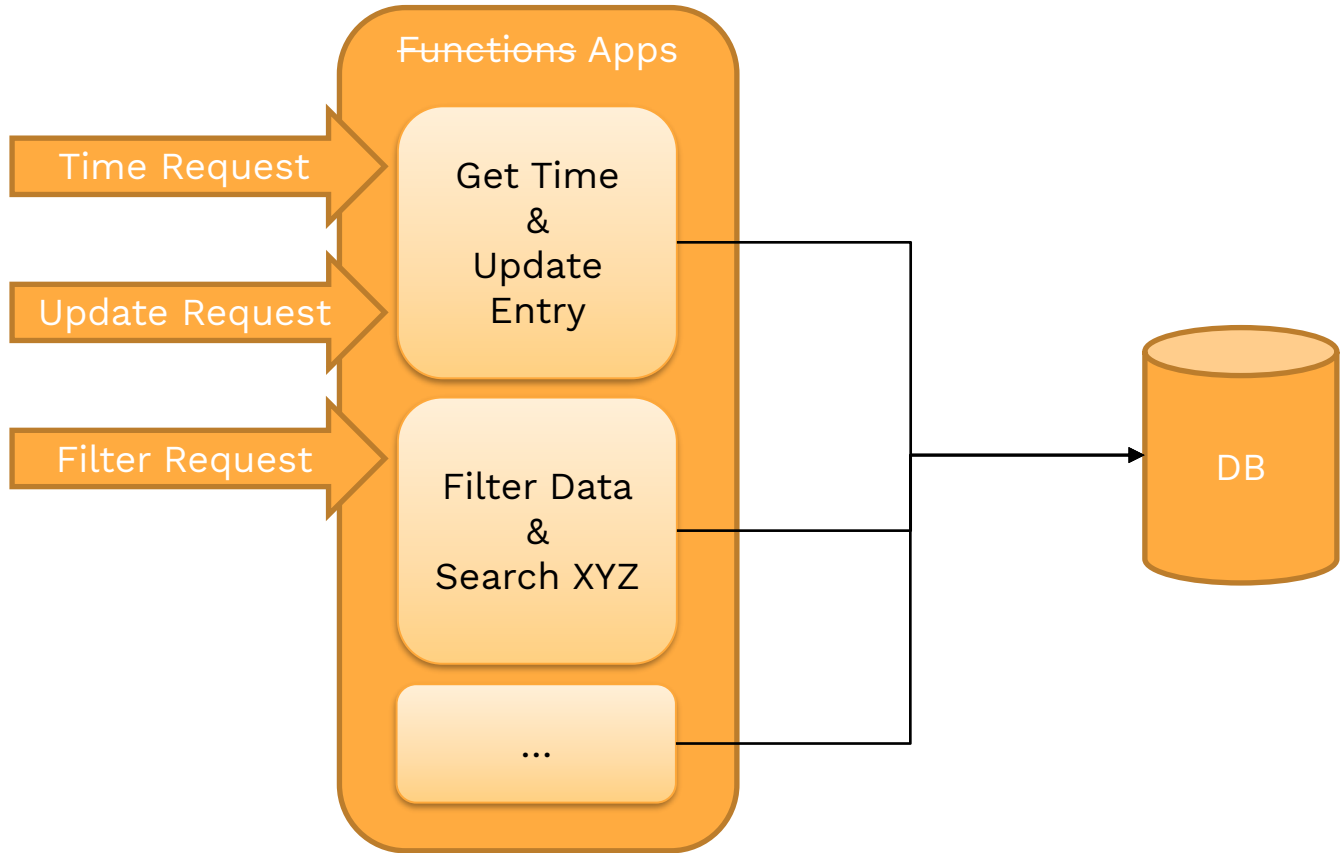
Let's Go further



Simple Serverless Architecture



“Simple” Serverless Architecture



GOals



Use our own Framework and Libs!



Start Simple! Stay Flexible!







Deploy to AWS (and others)



Don't care about Infrastructure!

A “Framework as a Service” approach

- +  Flexible and Extendable
 - Simple or use plugins/modules of the Framework
 - Routing handled by us, not the Ops part
- +  Local, Docker, On-Premise & SERVERLESS!
 - Dev-first! Ops-second
- +  Use common design patterns and structures
 - Keep It Simple, Serverless ;)
- +  Monolithic „Micro-“Services Architecture
 - Convention with Configuration





Express.js inspired
Web Framework

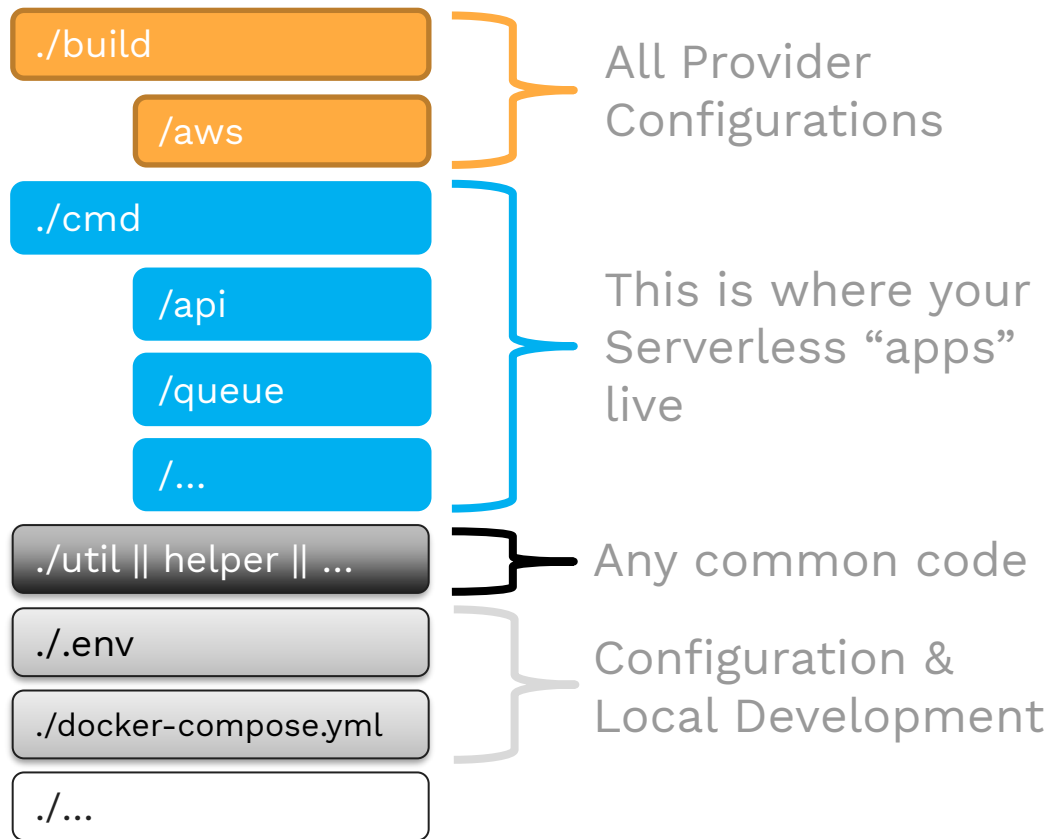


CLI &
Serverless Application
Model (SAM)



Build, Test & Deploy

Our Project Structure



Check it out at

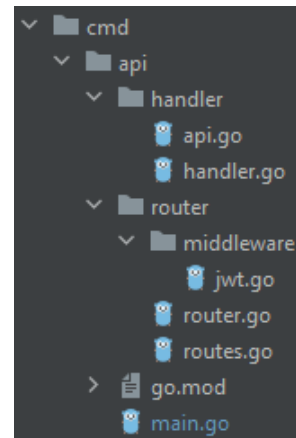


<https://github.com/elipZis/go-serverless>

- Creating fully-fledged Apps
 - go mod setup, standalone or intertwined
 - Use the libs and extensions you need!
- Combine with Vendor-specific apps
 - All is build together and deployed at once
- Start simple and extend later
 - For example split your App later if you need more flexibility over the configuration
 - Every cmd is one configurable Serverless „Function“

./cmd

/api



Single Point of Entry

```
func main() {
    // Branch your App by the environment, you can configure
    env := os.Getenv("SERVER_ENV")
    if env == "AWS" {
        // Hook your AWS handler
        lambda.Start(Handler)
    } else {
        // Or just start a local server
        app = fiber.New(fiber.Config{
            AppName: "Go Serverless!",
        })
        app.Listen(":1323")

        quit := make(chan os.Signal)
        signal.Notify(quit, os.Interrupt)
        <-quit
    }

    // Rest stays as-is!
}
```

Keep the
“special”
code limited
to a Single
Point of Code

Fiber → AWS Lambda Handler

```
import (  
    ""  
    fiberAdapter "github.com/aws-labs/aws-lambda-go-api-proxy/fiber"  
    ""  
)  
  
func init() {  
    // AWS Lambda needs a wrapper to proxy the requests  
    if os.Getenv("SERVER_ENV") = "AWS" {  
        fiberLambda = fiberAdapter.New(app)  
    }  
}  
  
// Handler to proxy AWS Lambda requests/responses  
func Handler(ctx context.Context, req events.APIGatewayProxyRequest)  
    (events.APIGatewayProxyResponse, error) {  
    return fiberLambda.ProxyWithContext(ctx, req)  
}  
  
// And we are back in the „normal“ Fiber context
```

AWS requires
a proxying of
requests!

The rest is
basic Fiber!

“Just” Develop!

The screenshot shows a Go IDE with a project structure on the left and code in the main editor. The project structure includes:

- go-serverless
- build
- aws
- layer
- chrome-aws-lambda
- gigamon
- env-example.json
- README.md
- template.yaml
- cmd
- api
- handler
- api.go
- handler.go
- router
- middleware
- jwt.go
- router.go
- routes.go
- go.mod
- main.go
- queue
- handler
- go.mod
- main.go
- web
- handler
- handler.go
- web.go
- router
- rice-box.go
- router.go
- routes.go
- view
- index.html
- go.mod
- main.go

The code in the main editor is for the `router` package:

```
1 package router
2
3 import (
4     "context"
5     "github.com/beertjohan/go.rice"
6     "github.com/eliqsis/go-serverless/util"
7     "github.com/gofiber/fiber/v2"
8     "github.com/gofiber/fiber/v2/middleware/compress"
9     "github.com/gofiber/fiber/v2/middleware/cors"
10    "github.com/gofiber/fiber/v2/middleware/logger"
11    "github.com/gofiber/template/html"
12    _ "github.com/joho/godotenv/autoload"
13    "log"
14 )
15
16 type Router struct {
17     App *fiber.App
18 }
19
20 // NewRouter Create a new router and configure some middlewares
21 func NewRouter() (*Router) {
22     this := new(Router)
23
24     //engine := html.New("./view", ".html")
25     engine := html.NewFileSystem(rice.MustFindBox("view").HTTPBox(), extension: ".html")
26     this.App = fiber.New(fiber.Config{
27         AppName: util.GetEnvOrDefault(key: "APP_NAME", defaultValue: "Go Serverless!"),
28         Views:   engine,
29     })
30
31     // Global middlewares
32     this.App.Use(cors.New())
33     this.App.Use(logger.New())
34     this.App.Use(compress.New())
35 }
36
37 Router
```

The terminal at the bottom shows the output of `go mod tidy`:

```
Run: Sync dependencies of github.com/eliqsis/go-serverless/...
go: downloading github.com/gofiber/fiber/v2 v2.39.0
go: downloading github.com/beertjohan/go.rice v1.0.7
go: downloading github.com/gofiber/template v1.6.25
go: downloading github.com/valyata/fasthttp v1.34.0
go: downloading golang.org/x/sys v0.0.0-2022022234510-4ee76aa19f9
go: downloading golang.org/x/net v0.0.0-20220225172249-27ad868420f
go: downloading github.com/fsnotify/fsnotify v1.5.1
go: downloading github.com/daaku/go.zipexe v1.0.0
go: downloading github.com/klauspost/compress v1.15.0
```

LET'S GO FURTHER





Bring it to the Cloud

Go Serverless!



./build

/aws

./cmd

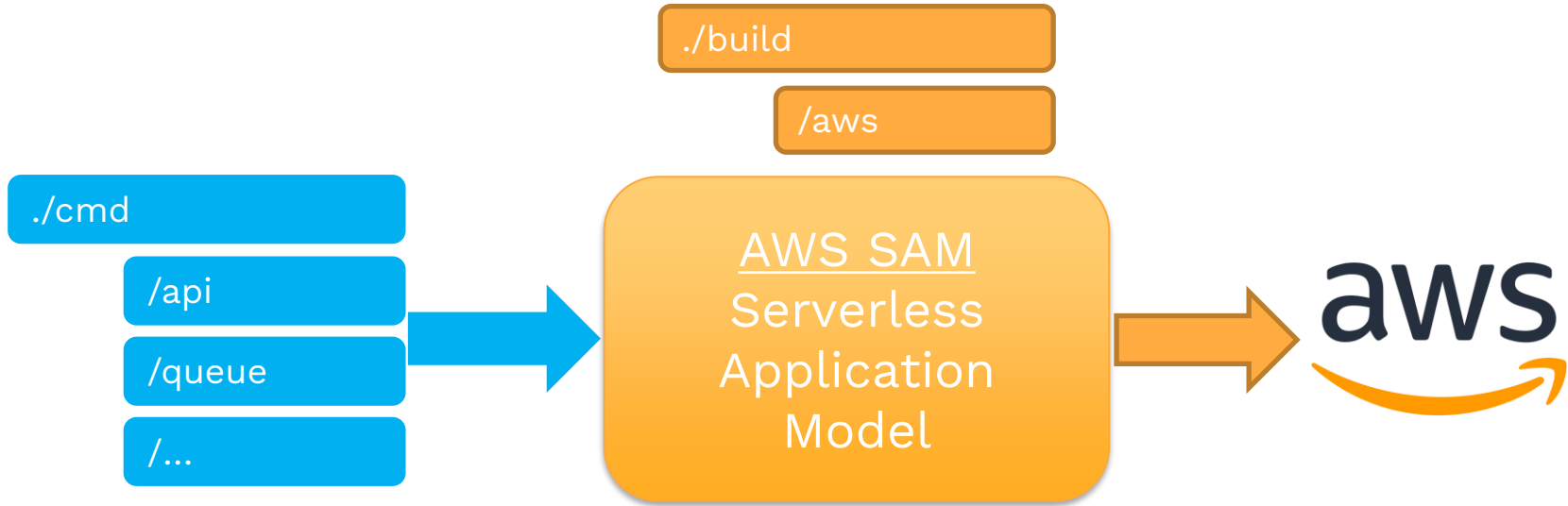
/api

/queue

/...



Infrastructure App as a Service



Ops-Layer between App and Serverless deployment

Starting from Scratch



- Init a project by „sam init“
 - Have SAM setup your structure
- Use the templates and deploy by zip
 - As we did before
- SAM builds your app and guides you through the deployment process
 - --guided only needed once!

```
:~$ sam init
Which template source would you like to use?
    1 - AWS Quick Start Templates ...
Choice: 1
What package type would you like to use?
    1 - Zip (artifact is a zip uploaded to S3) ...
Package type: 1
Which runtime would you like to use?
    1 - nodejs14.x
    ...
    4 - go1.x ...
Runtime: 4
Project name [sam-app]: go-serverless
Cloning from https://github.com/aws/aws-sam-cli-app-templates

AWS quick start application templates:
    1 - Hello World Example ...
Template selection: 1

:~$ sam build; sam deploy --guided
```

AWS SAM template.yml

```
ApiFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: ../../cmd/api/
    Handler: api
    Runtime: go1.x
    MemorySize: 128
    Timeout: 10
    ...
```

Define “apps”
as Serverless
Functions
and configure
your needs

When do your Apps trigger?



AWS SAM template.yml

```
ApiFunction:
  Type: AWS::Serverless::Function
  Properties:
    ...
    Events:
      GetResource:
        Type: Api
        Properties:
          Path: /{proxy+}
          Method: any
          RestApiId: !Ref ApiDeployment
    ...
  Environment:
    Variables:
      ENVIRONMENT: !Sub "${Environment}"
      SERVER_ENV: AWS
      RDS_HOST: !GetAtt RDS.Endpoint.Address
      RDS_USER:
        Ref: RDSUsername
```

Proxy Api
requests to
catch in
Framework
and pass the
Environment
dynamically

AWS SAM template.yml

```
YourFunction:
  ...
  Layers:
    - !Ref Chromium
  ...

ChromiumLayer:
  Type: AWS::Serverless::LayerVersion
  Name: !Sub "chromium-${Environment}"
  Properties:
    ContentUri: layer/chrome-aws-lambda/
```

Create
Execution
Environment
“layers” by
adding your
content

Configure your “Infrastructure”



AWS SAM template.yml

```
RDS:
  Type: AWS::RDS::DBInstance
  Properties:
    DBInstanceIdentifier: !Sub „mydb-${Environment}”
    DBName: "goserverless"
    DBInstanceClass: !Ref RDSInstanceClass
    AllocatedStorage: !Ref RDSAllocatedStorage
    Engine: "postgres"
    EngineVersion: "12"
    MasterUsername: !Ref RDSUsername
    MasterUserPassword: !Ref RDSPassword

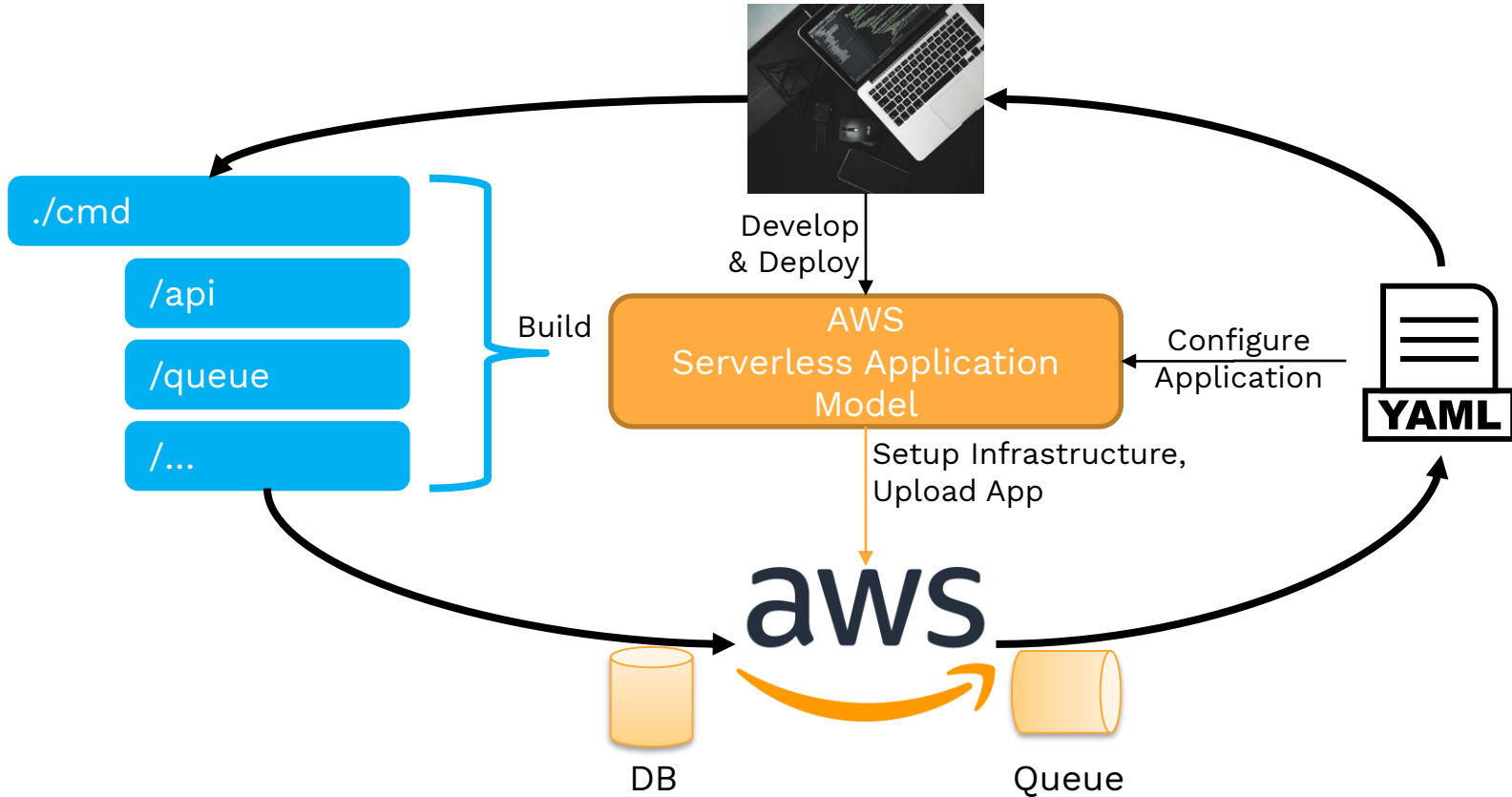
Queue:
  Type: AWS::SQS::Queue
  Properties:
    QueueName: !Sub „myqueue-${Environment}”
    VisibilityTimeout: 30
  ...
```

Define what
you need!

No manual
setup!

All “Resources”

Development to Deployment



- AWS SAM creates a reproducible configuration
 - `samconfig.toml`
- SAM Builds, Zips & Deploys
 - Based on the Config
 - No „cli-creation“ but reproducible
- One deployment
 - Per App, not Function by Function!

```
./build/aws/$ sam build
Building codeuri: /go-serverless/cmd/api runtime: go1.x metadata: {}
architecture: x86_64 functions: ['ApiFunction']
Running GoModulesBuilder:Build
Building codeuri: /go-serverless/cmd/web runtime: go1.x metadata: {}
architecture: x86_64 functions: ['WebFunction']
Running GoModulesBuilder:Build
Building codeuri: /go-serverless/cmd/queue runtime: go1.x metadata:
{} architecture: x86_64 functions: ['QueueFunction']
Running GoModulesBuilder:Build
Build Succeeded

./build/aws/$ sam deploy

./build/aws/$ sam deploy --profile=goserverless

./build/aws/$ sam deploy --config-env=prod

./build/aws/$ sam deploy --parameter-overrides Environment=stage
```

App Deployment



Deploying with following values

```
=====
Stack name           : elipzis-go-serverless
Region              : eu-central-1
Confirm changeset   : True
Disable rollback    : False
Deployment s3 bucket : aws-sam-cli-managed-default-samclisourcebucket-...
Capabilities         : ["CAPABILITY_IAM"]
Parameter overrides : {"Environment": "dev", "RDSInstanceClass": "db.t2.micro", ...}
Signing Profiles    : {}
```

Initiating deployment

=====

Uploading to elipzis-go-serverless/4eca16c2a58eac5d652a2307...template 7854 / 7854 (100.00%)

Waiting for changeset to be created..

CloudFormation stack changeset

```
-----
Operation                               LogicalResourceId
ResourceType                             Replacement
-----
+ Add                                     ApiDeploymentDeploymentbad58d6a2f
AWS::ApiGateway::Deployment              N/A
+ Add                                     ApiDeploymentStage
AWS::ApiGateway::Stage                   N/A
+ Add                                     ApiDeployment
AWS::ApiGateway::RestApi                 N/A
```

...



samconfig.toml

```
version = 0.1
[default]
[default.deploy]
[default.deploy.parameters]
stack_name = „elipzis-go-serverless"
s3_bucket = "aws-sam-cli-managed-default-samclisource..."
region = "eu-central-1"
parameter_overrides = "Environment=\\\"dev\\\""
RDSInstanceID=\\\"goserverless\\\" RDSEngine=\\\"postgres\\\""
RDSEngineVersion=\\\"12\\\" ...
profile = „goserverless"

[prod]
[prod.deploy]
[prod.deploy.parameters]
stack_name = " elipzis-go-serverless -prod"
s3_bucket = "aws-sam-cli-managed-default-samclisource..."
region = "eu-central-1"
profile = „goserverless"
parameter_overrides = "Environment=\\\"prod\\\" ..."
```

Infrastructure as Code
& App as Code
via **template.yml**

Deployment as Code
via **samconfig.toml**

-  **AWS SAM = Vendor lock!**
 - + Mediator to be exchanged with others e.g. GCP
 - + Maintaining „only an Ops-Layer“ between a single App and each Vendor
-  **Framework increases the bulk and image we deploy**
 - Boot & Execution times could increase
 - + Frameworks offer common optimizations, more frequently known

Why not

Everything Serverless?



Do I need Serverless?

NO!













Do I need Serverless?

BUT...

Know your Use Case!



- +  High-availability and scalable without a DevOps Team
- +  Easy to start & scale with your customers
- +  Availability when needed, not permanently paid
- +  Support Peaks without outages or slowdowns
- +  Much more to discover!
-  No insights into infrastructure
-  Costs can excel as Serverless requires optimization, too
-  „Permanent“ Apps might be better off going „classic“
-  Only as „scalable“ as the rest e.g. a slow DB
-  Limited computational resources

Thanks!

JUST GO!



goserverless@elipZis.com



Savas Ziplies

<https://elipZis.com>

@insanitydesign



<https://github.com/elipZis/go-serverless>