# Quickstart into Go

## Using Consul as a distributed message queue

Stephan Peijnik-Steinwender, Head of Research & Development

anexia

# OUTLINE

- Go
- Consul
- Quickstart
- Phase 0: project and basic code
- Phase 1: JSON
- Phase 2: Go and Consul
- Phase 3: Watching for changes
- Phase 4: Locking
- Phase 5: Putting it all together
- Use case: distributed Docker registry


GO & CONSUL

# Introduction

- General purpose
- Compiled
- Statically typed
- Concurrent
- Simple
- Productive

# History

- Google 2007 (Griesemer, Pike, Thompson)
- Open Source November 2009
- 1.0 released in March 2012
- 1.10.2 released 2018-05-01


# Introduction

- Service discovery
- Health Checking
- KV Store
- Distributed
- Highly available
- Written in Go


QUICKSTART

# QUICKSTART

## Go

- Text editor
- https://golang.org/doc/install


## Consul

- Official docker image
- Single node
- Ready to use

```
1 docker run --name=dev-consul -p 8500:8500 consul
```

```go
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("hello developers!")
7 }
```

# Go: Hello world

- Package: “main”
- Import statement: “fmt”
- “main” function

- “go run hello.go”

```go
package main

import (
    "fmt"
    "os"
)

func main() {
    name := "developers"
    if len(os.Args) > 1 {
        name = os.Args[1]
    }
    fmt.Printf("hello %s!\n", name)
}
```

# Go: Hello world extended

- “os” Package: CLI arguments
- “name” variable
- “if” clause
- fmt.Printf

- “go run hello_extended.go”

```go
package main

import (
    "fmt"
    "math/cmplx"
)

var (
    ToBe   bool       = false
    MaxInt uint64     = 1<<64 - 1
    z      complex128 = cmplx.Sqrt(-5 + 12i)
)

func main() {
    fmt.Printf("Type: %T Value: %v\n", ToBe, ToBe)
    fmt.Printf("Type: %T Value: %v\n", MaxInt, MaxInt)
    fmt.Printf("Type: %T Value: %v\n", z, z)
}
```

# Go: Basic data types

- bool
- string (Unicode)
- int, int8, int16, int32, int64
- uint, uint8, uint16, uint32, uint64
- byte (alias for uint8)
- rune (alias for int32, Unicode code point)
- float32, float64
- complex32, complex64

```go
package main

import (
    "fmt"
    "math/cmplx"
)

var (
    ToBe   bool       = false
    MaxInt uint64     = 1<<64 - 1
    z      complex128 = cmplx.Sqrt(-5 + 12i)
)

func main() {
    fmt.Printf("Type: %T Value: %v\n", ToBe, ToBe)
    fmt.Printf("Type: %T Value: %v\n", MaxInt, MaxInt)
    fmt.Printf("Type: %T Value: %v\n", z, z)
}
```

# Go: Zero value

- Numeric: 0
- bool: false
- string: “” (empty string)

```go
package main

import (
    "fmt"
)

func main() {
    var x string
    x = "a string"

    y := "another string"

    z := x
    fmt.Printf("x=%s,y=%s,z=%s\n", x, y, z)
}
```

# Go: Variables

- “var” keyword
- Type inference
- Unused variables are treated as errors

```go
package main

import (
        "fmt"
)

func main() {
        slice0 := []int{1, 2, 3}
        fmt.Printf("slice0: %v\n", slice0)
        slice0ref := slice0[0:2]

        // Length 3, capacity 5
        slice1 := make([]int, 3, 5)
        slice1[0] = 4
        slice1[2] = 5
        fmt.Printf("slice1: %v\n", slice1)

        slice0[0] = 99
        fmt.Printf("slice0: %v\n", slice0)
        fmt.Printf("slice0ref: %v\n", slice0ref)
}
```

# Go: Slices

- Like references to arrays
- Changing element modifies underlying memory
- Length and capacity

```go
package main

import (
        "fmt"
        "os"
)

func main() {
        userMap := make(map[string]string)
        userMap["stephan"] = "Stephan Peijnik-Steinwender"

        if len(os.Args) != 2 {
                fmt.Printf("usage: %s <username>\n", os.Args[0])
                os.Exit(1)
        }

        username := os.Args[1]

        realName, exists := userMap[username]
        if !exists {
                fmt.Printf("User %s not found.\n", username)
                os.Exit(2)
        }

        fmt.Printf("Real name of user %s: %s\n", username, realName)
}
```

# Go: Maps

- Maps map keys to values
- Initialization via make
- Iteration order non-deterministic

```go
package main

import (
        "fmt"
        "os"
)

func sayHello(name string) {
        fmt.Printf("hello %s!\n", name)
}

func main() {
        name := "developers"

        if len(os.Args) > 1 {
                name = os.Args[1]
        }

        sayHello(name)
}
```

# Go: Functions

- Parameter / Type order

```go
package main

import (
        "fmt"
)

type Person struct {
        Name string
}

func (p Person) SayHello() {
        fmt.Printf("hello %s!\n", p.Name)
}

func main() {
        p := Person{
                Name: "Stephan",
        }

        p.SayHello()
}
```

## Go: Structs

- Collection of fields
- Methods

```go
package main

import (
        "fmt"
)

func main() {
        slice := []int{8, 16, 32, 64}

        // 3-clause for
        for i := 0; i < len(slice); i++ {
                fmt.Printf("@%d: %d\n", i, slice[i])
        }

        // for range
        for idx, value := range slice {
                fmt.Printf("@%d: %d\n", idx, value)
        }
}
```

# Go: “for” loop

- Iteration over slices, maps, …
- 3-clause “for”
- “for range”

```go
package main

import (
    "fmt"
)

func main() {
    x := 1.5

    if x >= 1.5 {
        fmt.Println(">= 1.5")
    } else if x < 0.5 {
        fmt.Println("> 0.5")
    } else {
        fmt.Println("< 0.5")
    }
}
```

# Go: “if” conditions

- No parentheses required, but possible
- “if” – “else if” – “else”

```go
var ExportedVariable = 0xdeadbeef
var unexportedVariable = "unexported"

func ExportedFunc() {
    // exported function
}

type ExportedStruct struct {
    unexportedField string
    ExportedField   int
}
```

# Go: Scopes

- Uppercase first character: exported
- Lowercase first character: package-private

```
1 type Reader interface {
2     Read(p []byte) (n int, err error)
3 }
```

# Go: Interfaces

- Satisfied by implementing all methods
- Example: io.Reader
  - Provides “readability”
  - Implemented by readable types like files, sockets, buffers, …

## Go: A Tour of Go

- More information
- In-depth guide
- Interactive
- https://tour.golang.org

# PHASE 0

# PHASE 0 – PROJECT AND BASIC CODE

- GOPATH
  - “src/” directory holds code
  - namespaced (ie. github.com/anexia-it/go-quickstart)
- “go get”
  - retrieve packages and put them into GOPATH
- Vendoring
  - magic “vendor” folder
  - https://github.com/golang/dep

# PHASE 0 – PROJECT AND BASIC CODE

**github.com/anexia-it/go-quickstart**

- package “quickstart”
- “cmd/quickstart” folder – package “main”
- dep init

# PHASE 0 – PROJECT AND BASIC CODE

```go
package main

import (
        "github.com/spf13/cobra"

        "github.com/anexia-it/go-quickstart"
)

var cmdVersion = &cobra.Command{
        Use:   "version",
        Short: "print version information",
        Run: func(cmd *cobra.Command, _ []string) {
                cmd.Printf("quickstart v%s\n", quickstart.VersionString())
        },
}

func init() {
        cmdRoot.AddCommand(cmdVersion)
}
```

## Version command

- dependency: github.com/spf13/cobra
  - CLI framework

- Our first cobra command: “version”

# PHASE 0 – PROJECT AND BASIC CODE

```go
package quickstart

import "go.uber.org/zap"

// rootLogger holds the root logger instance and is a package-private variable
var rootLogger *zap.Logger

// GetLogger returns a named logger
func GetLogger(name string) *zap.Logger {
    return rootLogger.Named(name)
}

func init() {
    // init is called during program initialization

    // Create a new logger and ignore possible errors
    rootLogger, _ = zap.NewDevelopment()
}
```

## Logging

- dependency: go.uber.org/zap
  - Logging framework

- github.com/anexia-it/go-quickstart/log.go
  - Logging utility function GetLogger
  - “init” function
  - Package scope variable


PHASE 1

# PHASE 1 – WORKING WITH JSON

```go
package quickstart

// Request represents a worker queue request
type Request struct {
    URL string `json:"url"`
}
```

## request.go

- Struct holding request information
  - URL field
- Mapping via “tags”

# PHASE 1 – WORKING WITH JSON

```
1 type encoder interface {
2         Encode(o interface{}) error
3 }
```

## encoder interface

- Designed to be implemented by
  - encoding/json Encoder
  - github.com/anexia-it/go-human Encoder
- Simple “switching” between encodings in CLI


PHASE 2

# PHASE 2 – INTERACTING WITH CONSUL KV


## Consul KV interaction

- Client from github.com/hashicorp/consul
- Consul running in docker container
- CLI commands
  - get
  - put
  - delete


PHASE 3

# PHASE 3 – WATCHING FOR CHANGES IN CONSUL KV


## Consul KV watching

- CLI watch command
- Waits for changes to consul KV and prints them


PHASE 4

# PHASE 4 – CONSUL KV LOCKS


## Consul KV locks

- lock command
  - Command holds lock on given key until command is canceled
- locked-put command
  - Command obtains lock before writing key


PHASE 5

# PHASE 5 – PUTTING IT ALL TOGETHER


## Consul KV

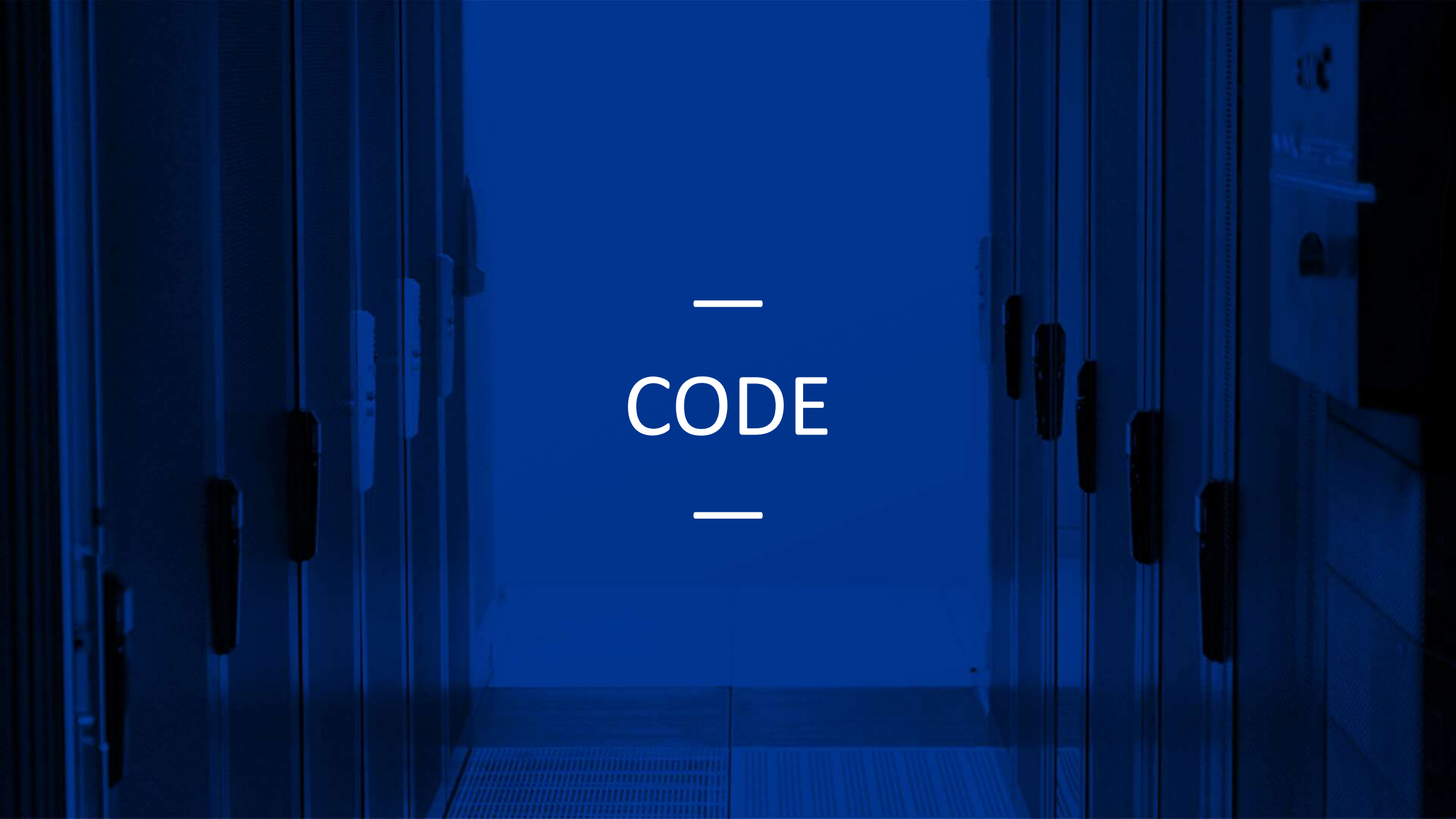
- “requests/” KV prefix
  - Requests
  - Key built using UUIDv4, generated by client
  - worker locks key during operation …
  - … and deletes the request key
  - workers watch prefix for changes

# PHASE 5 – PUTTING IT ALL TOGETHER


## Consul KV

- “results/” KV prefix
  - Written by worker
  - Request UUID identifies request
  - Client detects result by watching key

# DISTRIBUTED DOCKER REGISTRY


CODE


# Code at GitHub

- https://github.com/anexia-it/go-quickstart
- Steps are tagged
- Do not use in production ;-)