Golang a Go-Go

An Introduction to Go

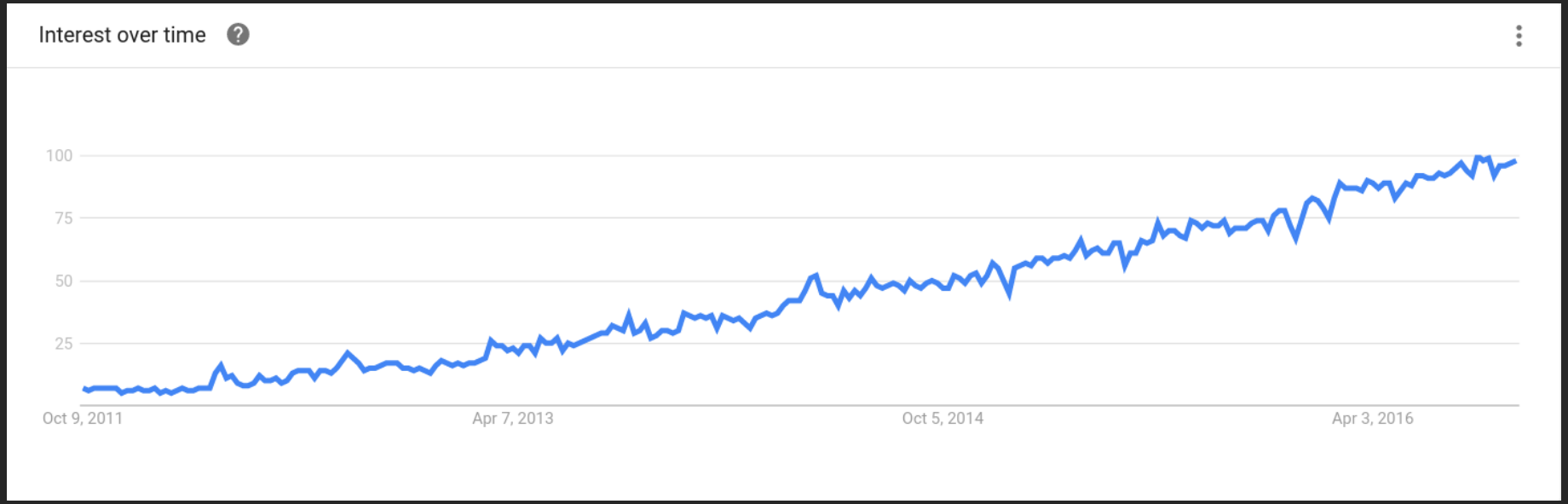
Scott Seighman
Solutions Architect, Red Hat

Why Golang?

- Designed for massive clusters
- Good tooling
- Modern, efficient language
- Fast compiles
- Easy learning curve
- The case for Go: http://bit.ly/1RGK0h1

Why Golang?

Google Trends: Interest over time



Go Language Overview

- Compiled ➡ statically-linked binary executables
- Statically typed
- Very opinionated
- General notes:
  - 25 keywords (like `func`, `struct`, `const`)
  - 36 pre-declared names (like `int` and `true`)
  - 4 major declarations:
    - `var`, `const`, `type` and `func`

Go Compared to Java

- No classes
- No constructors
- No inheritance
- No final
- No exceptions
- No annotations
- No user-defined generics

Go Language Basics

- Go programmers typically keep all code in a single workspace
- A workspace can contain many version control repositories (Git, for example)
- Each repository contains one or more packages
  - Each package consists of one or more Go source files in a single directory
- Path to a package's directory determines its import path

GOPATH Environment Variable

- **GOPATH** environment variable specifies location of your workspace
- Create a workspace directory and set **GOPATH** accordingly
- Your workspace can be located anywhere, but it's typically located in **$HOME/work**
  - Note: Must not be the same path as your Go installation

Go Workspaces

- A workspace is a directory hierarchy with three directories at its root:
  - `src` contains Go source files
  - `pkg` contains package objects
  - `bin` contains executable commands
- The `go` tool builds source packages and installs the resulting binaries to `bin`
- The `src` subdirectory typically contains multiple version control repos that track the dev source packages

Go Formatting

- Go takes an unusual approach and lets the machine take care of most formatting issues
- The `gofmt` program (also available as `go fmt`, operates at the package level rather than source file level) reads a Go program, emits the source in standard indentation, retaining and if necessary reformatting comments
- To handle a new layout situation, run `gofmt`
  - if the answer doesn't seem right, rearrange your program (or file a bug), don't work around it

Go Example

```
package main 1

import “fmt” 2

/* Print something */ 3

func main() { 4
   fmt.Println(“Hello World!”) 5
}
```

Go Commands

`go build`	Compile packages and dependencies
`go clean`	Remove object files
`go doc`	Show docs for package or symbol
`go env`	Print Go environment information
`go fix`	Run go tool fix on packages
`go fmt`	Run gofmt on package sources
`go generate`	Generate Go files by processing source

Go Commands (cont)

`go get`	Download & install packages & deps
`go install`	Compile & install packages & deps
`go list`	List packages
`go run`	Compile and run Go program
`go test`	Test packages
`go tool`	Run specified go tool
`go version`	Print Go version
`go vet`	Run go tool vet on packages

Go Dev/Build Process

- Create project in `src` directory
- Set **GOBIN** path (`bin` directory of your **$GOPATH**)
- In `src` directory containing `.go` file, run:

```
$ go build
$ go install
```

- **Executable will be installed in $GOPATH/bin**

```
$ ./helloworld
Hello World!
```

Dependency Management

- Out of the box, there is no ‘real’ package management
  - Currently no agreed upon package manager
- Point towards a namespace and that path is used to find your code during the build
- Less than ideal …

Dependency Management

- Can use “vendoring”
  - Copying the source from a dependency into your own source
- “Import rewriting”
  - Change import path to not point at the original source, but rather at a path in your tree
- “GOPATH rewriting”
  - Change the $GOPATH variable when switching projects

Dependency Management

- Most popular solution: `godep`
- Run `go get` to install a dependency
- Use the `godep save./...` command
  - Will copy all of the referenced code imported into the project from the current $GOPATH into the ./Godeps directory in your project
  - Will walk the graph of dependencies and create a ./Godeps/Godeps.json file
- To build, use: `godep go build`

Concurrency

- Remember, concurrency isn’t parallelism
- Concurrent programs in Go need just two components:
  - Goroutines
  - Channels

Concurrency

- Goroutines are:
  - Independently executing function within your program
  - Similar to lightweight thread
  - Run in the same address space
- Channel is:
  - Typed communication conduit to send/receive messages
  - Allows two goroutines to communicate

Goroutines

- A goroutine is a function that is capable of running concurrently with other functions
- Denoted by adding the word ‘go’ in front of the function call

Channels

- Channels provide a way for two goroutines to communicate with one another and synchronize their execution

Interfaces

- With Go, we have the convenience of “duck typing” with the safety of static checking
- If it has a set of methods that match an interface, then you can use it wherever that interface is needed without explicitly defining that your types implement that interface

Debugging Go

- `fmt.Printf()` is your friend
- `fmt.Fprintf()` can output to a file
- Go also has a built-in log and syslog library
- gdb
- godebug
- Delve

Cross Compiles

- Simply set two environment variables:
  - **$GOOS**, which is the target operating system
  - **$GOARCH**, which is the target processor
- Then we run **go build** as normal:

```
$ GOOS=windows GOARCH=386 go build arch.go
```

Go Capable Editors

Atom

Visual Studio Code

Sublime Text

Go Resources

- Books
  - **https://github.com/dariubs/GoBooks**
- Presentations
  - **https://talks.golang.org/**
- Tutorials
  - **https://tour.golang.org/**
  - **https://gobyexample.com/**
  - **https://miek.nl/go/**

Go Resources

- Courses
  - **https://golangschool.com/**
  - **https://www.udemy.com/learn-how-to-code/**
- Go frameworks, libraries and software
  - **http://awesome-go.com/**
- GoDoc (Go packages on Bitbucket, GitHub, etc)
  - **https://godoc.org/**
- Golang: The Good, the bad, the ugly
  - **https://www.youtube.com/watch?v=cMYhGNofHA4**

Summary

- Modern language
  - Efficient
  - Easy to learn
  - Good tooling
  - Many features

Thank You!

Scott Seighman
sseighma@redhat.com

Source files:
https://github.com/swseighman/linkstate16

How about some EXAMPLES?

Go Language Overview (cont)

- `Packages` in Go serve the same purpose as libraries or modules in other languages

Variables

- Variables declared without an explicit initial value are given their **zero value**
- When declaring a variable without specifying an explicit type, the variable's type is inferred from the value on the right

Variables (cont)

- Go is different from (most) other languages in that the type of a variable is specified *after* the variable name
  - Example: `var a int` (not `var int a`)

Go Keywords

break	default	func	interface	select
case	defer	go	map	struct
chan	else	goto	package	switch
const	fallthrough	if	range	type
continue	for	import	return	var