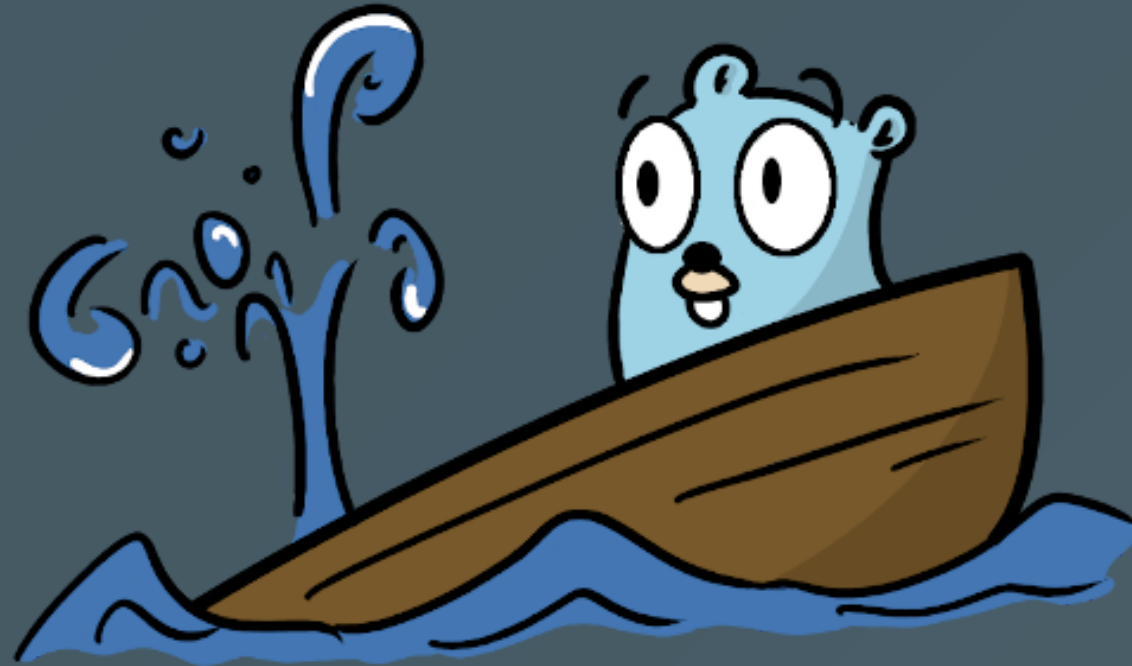


# Shipping Go Without Sinking



Go West Conference 2022

# Speaker Introduction

- **Name:** Sebastian Spaink
- Writing Go for 4 years
- Software Engineer at InfluxData



# What is Telegraf?

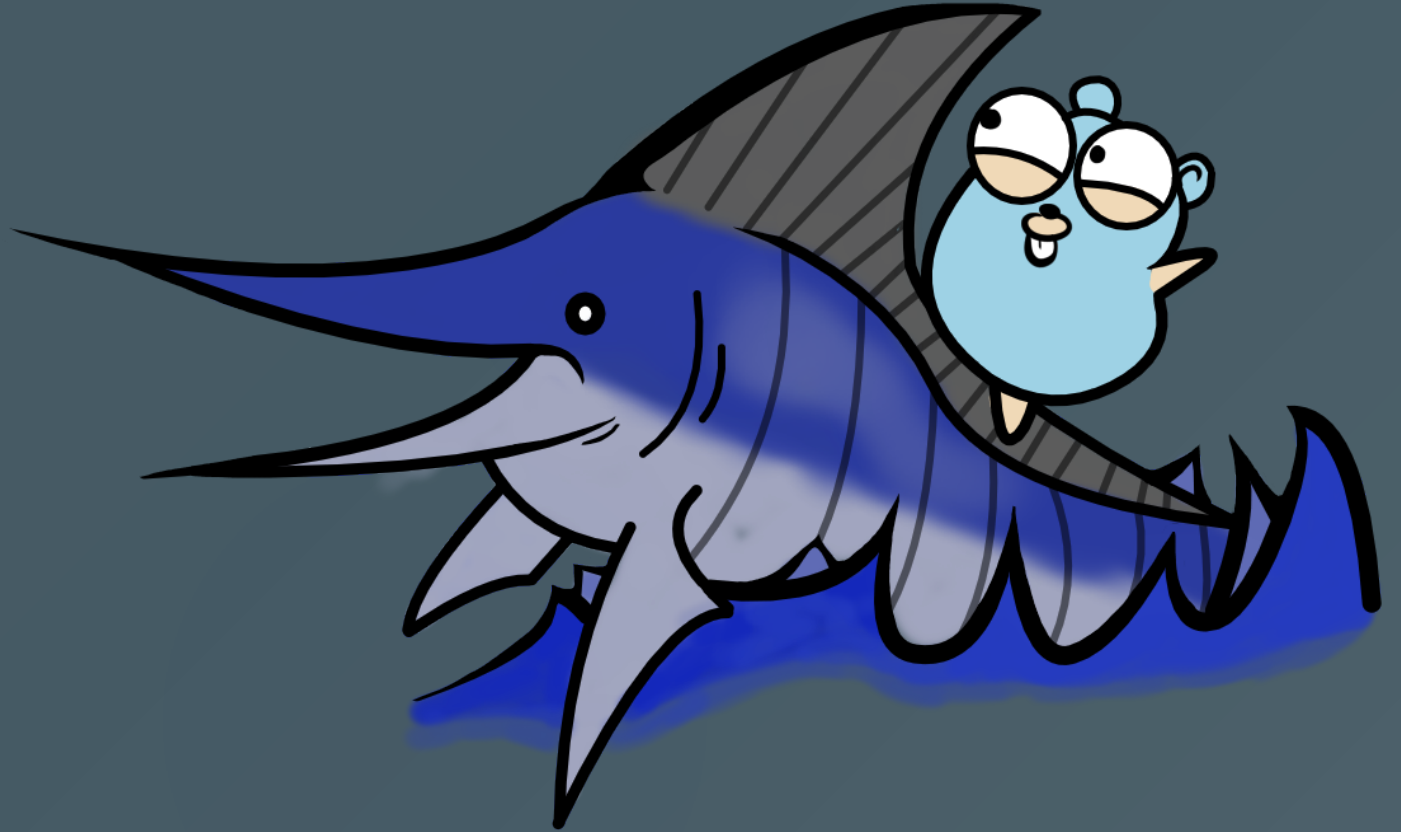
Collect data, organizes it, and push it where you want

- 40+ releases of Telegraf
- Open Source, MIT License
- 1000+ Unique Contributors



# A successful release of Telegraf

- Single Binary
- Cross Platform
- Github Release
- Website release
- Dockerfile



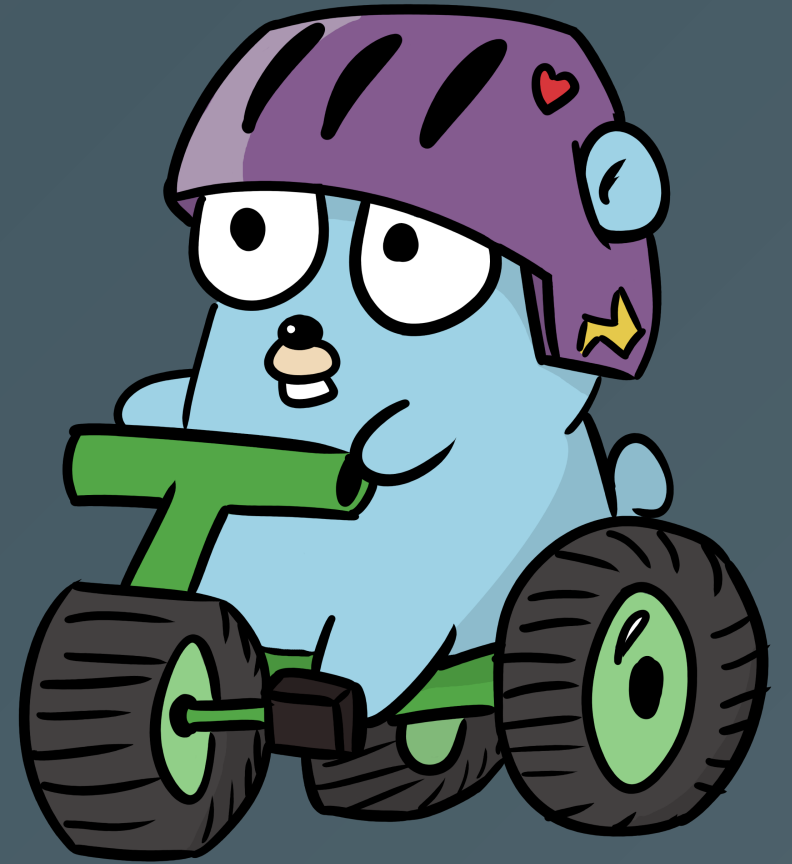
# Release Cycle

## Feature releases

- End of every quarter  
(March, June, September, December)

## Maintenance releases

- Every 3 weeks



# Semantic Versioning







example current version 1.19.0

| Type   | Version | Description                           |
|--------|---------|---------------------------------------|
| Patch: | 1.19.1  | maintenance release                   |
| Minor: | 1.20.0  | feature release                       |
| Major: | 2.0.0   | feature release with breaking changes |

# Starting the Release

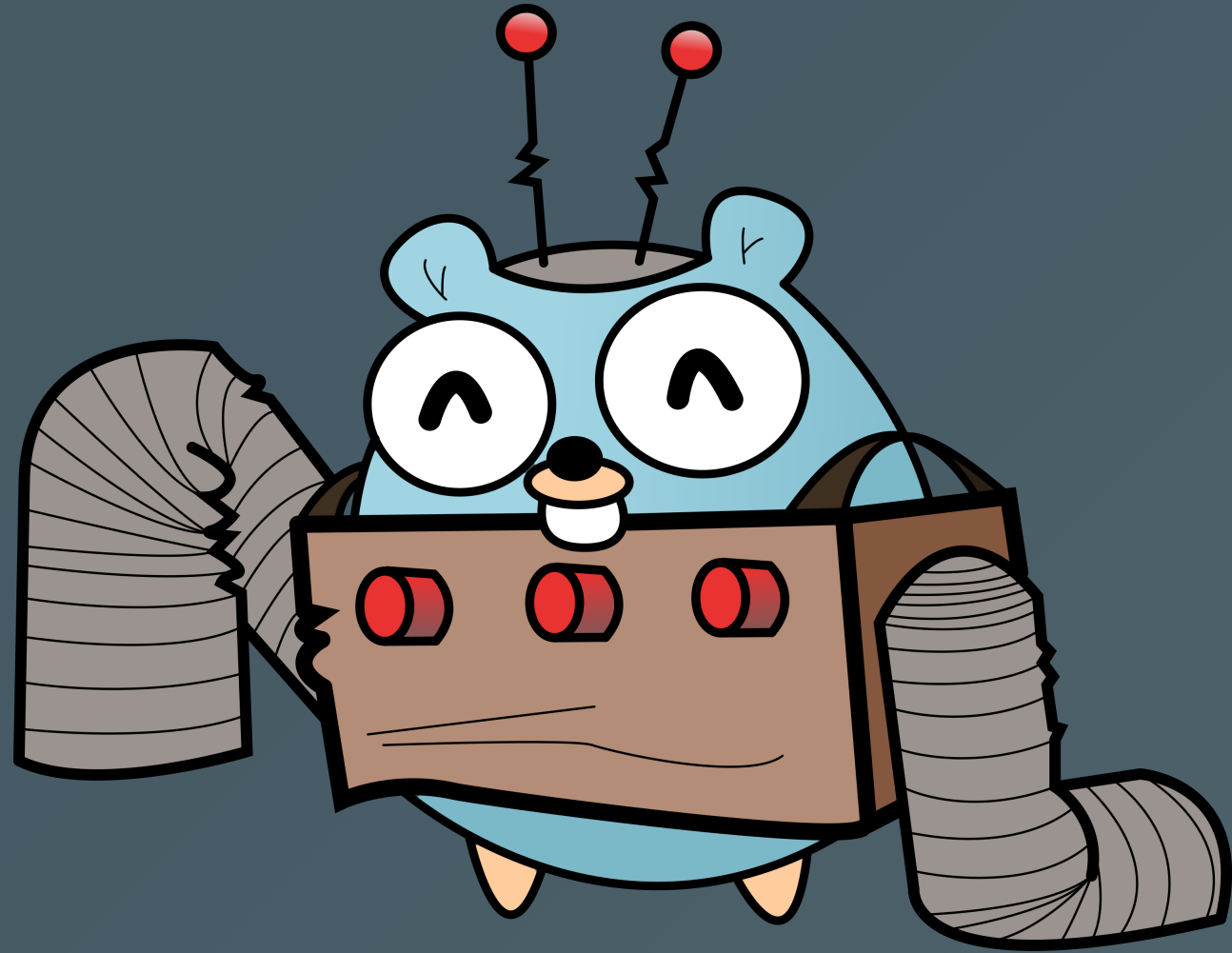
- Eeny, meeny, miny moe, Catch a tiger by the toe
- Track progress in slack

## Sebastian Spaink v1.23.0 release thread

-  Prepare release branch
-  Sign packages
-  Update the [repos.influxdata.com](https://repos.influxdata.com) repository
-  Update Docker images
-  Update downloads page
-  Update Helm charts

# Steps Overview

1. Preparing the git repo
2. Building and packaging
3. Distributing the binaries





# Preparing the git repo



# Conventional Commit Messages

Adding human and machine readable meaning to commit messages

**Example:** feat(inputs.directory\_monitor): Traverse sub-directories

| Type | Optional Scope           | Description              |
|------|--------------------------|--------------------------|
| feat | inputs.directory_monitor | Traverse sub-directories |

# Git branching strategy

- copy master branch
- branch for each release
- maintain release branch



# Executing external commands

```
func RunCommand(path, command string, arguments ...string) (string, error) {  
    var w io.Writer  
    var stdBuffer bytes.Buffer  
    w = io.MultiWriter(os.Stdout, &stdBuffer)  
  
    cmd := exec.Command(command, arguments...)  
    cmd.Dir = path  
    cmd.Stdout = w  
    cmd.Stderr = w  
    err := cmd.Run()  
    if err != nil {  
        return "", err  
    }  
    return stdBuffer.String(), nil  
}
```

# Adding some color

```
// Wrap color implements the io.Writer interface  
// Using fatih/color package it prints to stdout but with color  
type WrapColor struct{}
```

```
func (w *WrapColor) Write(b []byte) (int, error) {  
    color.Cyan(string(b))  
    return len(b), nil  
}
```

.....

```
var w io.Writer  
var stdBuffer bytes.Buffer  
var c WrapColor  
w = io.MultiWriter(&c, &stdBuffer)
```

# Running the commands

```
RunCommand(telegrafPath, "make", "tidy")
RunCommand(telegrafPath, "make", "test")

RunCommand(path, "git", "add", strings.Join(changes, " "))
RunCommand(path, "git", "commit", "-m", commitMsg)
RunCommand(path, "git", "push", "origin", branch)
hash, _ := RunCommand(path, "git", "rev-parse", "HEAD")
```

# Working with git commits

```
type Commit struct {  
    Hash          string  
    AuthorName    string  
    Type          string // (e.g. `feat`)  
    Scope         string // (e.g. `core`)  
    Subject       string // (e.g. `Add new feature`)  
    Title         string // (e.g. `feat(core): Add new feature`)  
}
```

# Collecting git commits

```
separator := "@@__CHGLOG__@"
delimiter := "@@__CHGLOG_DELIMITER__@"
logFormat := separator + strings.Join([]string{
    "HASH",
    "AUTHOR",
    "SUBJECT",
    "BODY",
}, delimiter)

arguments := []string{"log", fmt.Sprintf("--pretty=%s", logFormat), fromBranch + ".." + toBranch}
RunCommand(path, "git", arguments...)
```



# Changelog template

```
## {{ .Version }} [{{ .Date }}]  
{{ range .CommitGroups }}  
### {{ println .Title }}  
{{ range .Commits -}}  
- {{ .PullRequestLink }} {{ if .Scope }}`{{ .Scope }}` {{ end }}{{ println .Subject }}  
{{- end -}}  
{{ end -}}
```

# Example

**v1.24.1 [2022-09-19]**

## Bugfixes

- [#11787](#) Clear error message when provided config is not a text file

## Dependency Updates

- [#11788](#) `deps` Bump [cloud.google.com/go/pubsub](https://cloud.google.com/go/pubsub) from 1.24.0 to 1.25.1

# Preparing the code: lessons learned

- Organizing commits with conventional commit messages
- Write the release tools the team knows best
- Allow tools to be re-run

# Preparing the code: future improvements

- Cherry-pick changes into release branch continuously
- Update changelog continuously

# Building and Packaging



# The Makefile

- make test
- make build
- make package include\_packages="\$(make windows)"

```
LDFLAGS := $(LDLAGS) -X $(INTERNAL_PKG).Commit=$(commit) -X $(INTERNAL_PKG).Branch=$(branch)
ifneq ($(tag),)
    LDLAGS += -X $(INTERNAL_PKG).Version=$(version)
else
    LDLAGS += -X $(INTERNAL_PKG).Version=$(version)-$(commit)
endif

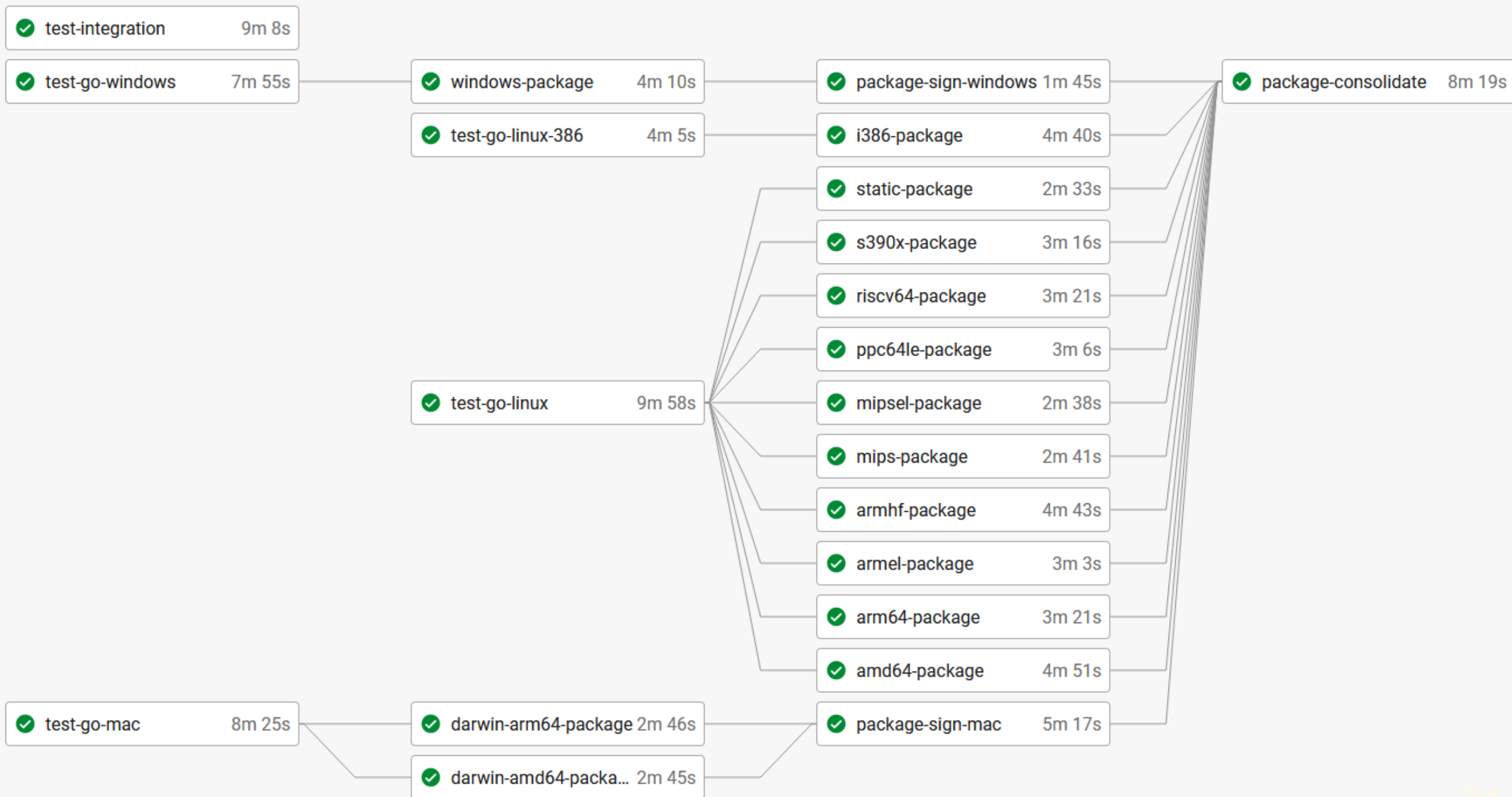
.PHONY: build
build:
    go build -tags "$(BUILDTAGS)" -ldflags "$(LDLAGS)" ./cmd/telegraf
```

# Mage

```
func build(p platform) (string, error) {
    env := map[string]string{"GOOS": p.OS, "GOARCH": p.ARCH}
    folderName := fmt.Sprintf("%s_%s", p.OS, p.ARCH)
    binName := productName + p.Extension
    filePath := filepath.Join("bin", folderName, binName)
    err := runCmd(env, "go", "build", "-o", filePath, "cmd/main.go")
    if err != nil {
        return "", err
    }

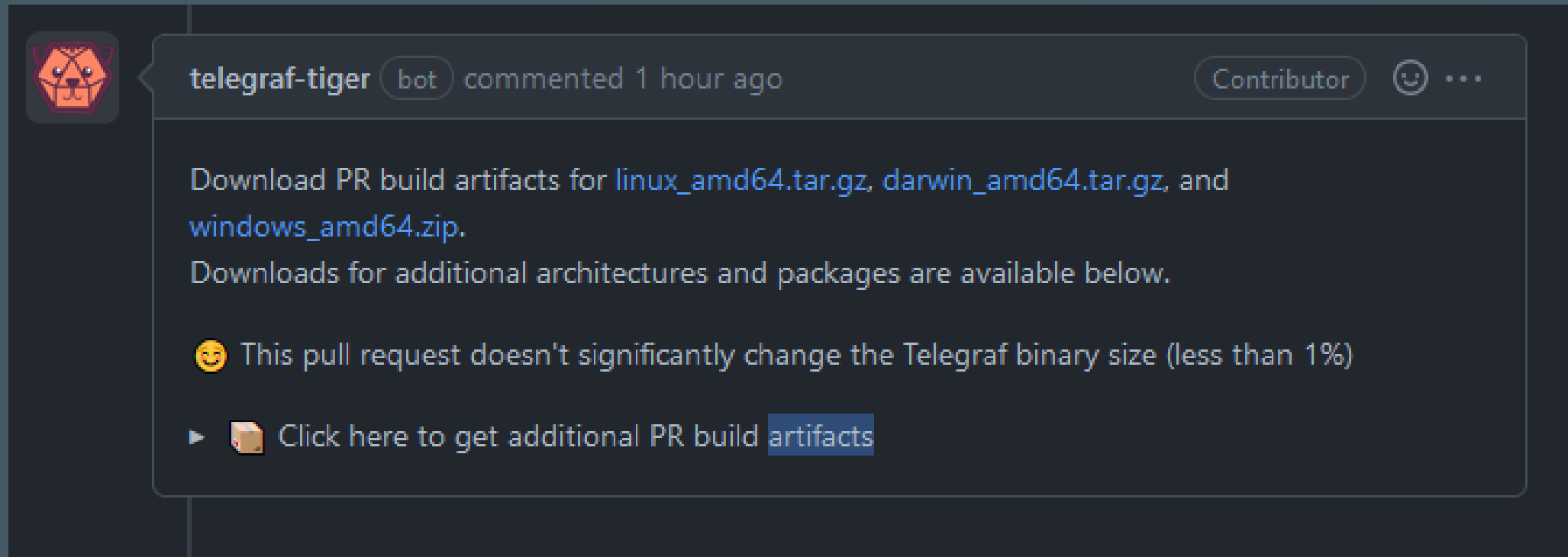
    return filePath, nil
}
```

Check it out: <https://github.com/magefile/mage>





# Automatic Alpha Builds



# Adding version and icon to Windows

Using package `josephspurrier/goversioninfo`

```
//go:generate goversioninfo -icon=../../assets/windows/tiger.ico
```

```
{  
  "StringFileInfo": {  
    "ProductName": "Telegraf",  
    "ProductVersion": "1.25.0-71b4a0af"  
  }  
}
```

# Generating versioninfo.json

```
version, _ := exec.Command("make", "version").Output()

v := VersionInfo{
    StringFileInfo: StringFileInfo{
        ProductName:    "Telegraf",
        ProductVersion: version,
    },
}

if err := ioutil.WriteFile("cmd/telegraf/versioninfo.json", file, 0644); err != nil {
    log.Fatalf("Failed to write versioninfo.json: %v", err)
}
```



telegraf.exe Properties

General Compatibility Digital Signatures  
Security Details Previous Versions

| Property           | Value             |
|--------------------|-------------------|
| <b>Description</b> |                   |
| File description   |                   |
| Type               | Application       |
| File version       | 0.0.0.0           |
| Product name       | Telegraf          |
| Product version    | 1.24.1-bd7d53fb   |
| Copyright          |                   |
| Size               | 143 MB            |
| Date modified      | 9/28/2022 1:48 PM |
| Language           | Language Neutral  |

[Remove Properties and Personal Information](#)

OK Cancel Apply

# Building and Packaging: lessons learned

- Don't run the CI on wednesdays
- Make artifacts readily available
- Sign mac nightly, replicate release pipeline

# Building and Packaging: future improvements


- Migrate more steps to continuous integration pipeline


# Distributing Binaries



Releases

Tags

 Choose a tag ▾










 Target: master ▾

Choose an existing tag, or create a new tag when you publish this release.

Release title

Write

Preview

Generate release notes

Describe this release



Attach binaries by dropping them here or selecting them.

☐ This is a pre-release

We'll point out that this release is identified as non-production ready.

Publish release

Save draft



# Updating the Website

```
b, err = sjson.SetBytes(b, "telegraf_stable.version", version)
if err != nil {
    return nil, fmt.Errorf("failed to set telegraf_stable.version: %w", err)
}
```

## Version

Telegraf v1.24.1



## Platform

Linux Binaries (64-bit)



SHA256: `f3a3d80909e54599fec d98dc fc7ae1e0202153d89f1f4ec d9c28a51658ac357d`

```
wget https://dl.influxdata.com/telegraf/releases/telegraf-1.24.1_linux_amd64.tar.gz
tar xf telegraf-1.24.1_linux_amd64.tar.gz
```

[Documentation](#) | [Release Notes](#)

# Testing Dockerfile

```
container, _ := testcontainers.GenericContainer(ctx, testcontainers.GenericContainerRequest{
    ContainerRequest: req,
    Started:          true,
})

name, _ := container.Name(ctx)

cmd := exec.Command("docker", "exec", name, "telegraf", "--version")
out, _ := cmd.CombinedOutput()

// Verify the correct version of Telegraf exists
if !strings.Contains(string(out), ver.String()) {
    return fmt.Errorf("Expected docker to give telegraf version %s but got %s", version, string(out))
}
```

# Distributing Binaries: lessons learned

- Document all permissions required
- If there is a problem increment patch, or the hash will change

# Distributing Binaries: future improvements

- Migrate more steps to continuous integration pipeline

# Conquering a Major Release

- Breaking changes
- Backwards compatibility important
- Deprecating configuration options slowly
- Getting into the habit

