Build Go projects with Github Actions

Posted on 12 Feb 2019

Github Actions is the hot new platform Github introduced in late 2018 that aims to be a generic workflow automation tool. Indeed an interesting move by Github, a company that stayed devoted to improvement of collaboration tools for developers, to expand into CI/CD space. Being a Github user for quite some time (10 years!) i got really excited about mysterious Actions project, which rolled out in Beta only.

Overview

Before we dig into Actions and Workflows, lets undestand why this thing even exist in the first place. To get any sort of automation for the repo, like running tests or creating artifacts (packages, binaries, etc), you'd have to use an external service or roll your own. Basically, your app would receive events from Github using webhooks and perform certain tasks. With Github Actions you could set up all these tasks without any third-party tools, all powered by Docker containers under the hood. Actions could also be shared using public Docker images or as part of a git repo.

Actions are not just simple tasks, they are part of the workflows that define the execution order and flow. One of the common use cases for Actions would be some kind of push -> analyze -> lint -> build -> release process, or sending notifications to third-party services. In case of this blog post - we want to cross compile Go binaries for all operating systems using a single action, and upload the artifacts to S3.

Workflow Setup

For the sake of simplicity we'll use a dummy Go project with a single file like this:

```
package main

import(
  "fmt"
)

func main() {
  fmt.Println("Hello World!")
}
```

Save it under your GOPATH, like `~/go/src/github.com/username/dummy/main.go`. That way we can easily run `go build` and `go install`. In the end we'll have a binary that just prints out `Hello World!`, nothing else.

Next, we'd want to cross compile binaries for OSX/Linux/Windows (386/x64). Pretty straightforward process (on OSX/Linux at least) and could be done with a command:

```
GOOS=linux GOARCH=amd64 go build -o dummy_linux_amd64
```

Let's setup a Github Workflow file `.github/main.workflow`. Our new worflow will only include a single task for cross-compilation:

```
workflow "Build Project" {
  // We want to run the workflow on each push
  on = "push"

  // Specify which actions should be triggered
  resolves = ["build"]
```

```
}

action "build" {
  // My public action
  uses = "sosedoff/actions/golang-build@master"

  // Optional args for specific architechtures
  args = "linux/amd64 darwin/amd64"
}
```

When you commit and push out your local changes, Github will pick up the workflow and trigger a new run using the action repository specified with `uses` keyword.

Action Script

I've created an action that uses the standard golang Docker image with a few tweaks:

```
FROM golang:1.11

RUN \
  apt-get update && \
  apt-get install -y ca-certificates openssl zip && \
  update-ca-certificates && \
  rm -rf /var/lib/apt

COPY entrypoint.sh /entrypoint.sh

ENTRYPOINT ["/entrypoint.sh"]
```

As for the actual `entrypoint.sh` script that cross-compiles the binaries, let's have a look:

```
#!/bin/bash

set -e

if [[ -z "$GITHUB_WORKSPACE" ]]; then
  echo "Set the GITHUB_WORKSPACE env variable."
  exit 1
fi

if [[ -z "$GITHUB_REPOSITORY" ]]; then
  echo "Set the GITHUB_REPOSITORY env variable."
  exit 1
fi

root_path="/go/src/github.com/$GITHUB_REPOSITORY"
release_path="$GITHUB_WORKSPACE/.release"
repo_name="$(echo $GITHUB_REPOSITORY | cut -d '/' -f2)"
targets=${@-"darwin/amd64 darwin/386 linux/amd64 linux/386 windows/amd64 windows/386"}

echo "----> Setting up Go repository"
mkdir -p $release_path
mkdir -p $root_path
cp -a $GITHUB_WORKSPACE/* $root_path/
cd $root_path

for target in $targets; do
  os="$(echo $target | cut -d '/' -f1)"
  arch="$(echo $target | cut -d '/' -f2)"
  output="${release_path}/${repo_name}_${os}_${arch}"

  echo "----> Building project for: $target"
  GOOS=$os GOARCH=$arch CGO_ENABLED=0 go build -o $output
  zip -j $output.zip $output > /dev/null
done

echo "----> Build is complete. List of files at $release_path:"
```

```
cd $release_path
ls -al
```

Once the action run is complete you'll see something similar in your logs:

```
----> Setting up Go repository
----> Building project for: darwin/amd64
----> Building project for: darwin/386
----> Building project for: linux/amd64
----> Building project for: linux/386
----> Building project for: windows/amd64
----> Building project for: windows/386
----> Build is complete. List of files at /github/workspace/.release:
total 16436
drwxr-xr-x 2 root root    4096 Feb  5 00:03 .
drwxr-xr-x 5 root root    4096 Feb  5 00:02 ..
-rwxr-xr-x 1 root root 1764764 Feb  5 00:02 test-go-action_darwin_386
-rw-r--r-- 1 root root  978566 Feb  5 00:02 test-go-action_darwin_386.zip
-rwxr-xr-x 1 root root 2003480 Feb  5 00:02 test-go-action_darwin_amd64
-rw-r--r-- 1 root root 1008819 Feb  5 00:02 test-go-action_darwin_amd64.zip
-rwxr-xr-x 1 root root 1676585 Feb  5 00:02 test-go-action_linux_386
-rw-r--r-- 1 root root  918555 Feb  5 00:02 test-go-action_linux_386.zip
-rwxr-xr-x 1 root root 1906945 Feb  5 00:02 test-go-action_linux_amd64
-rw-r--r-- 1 root root  952985 Feb  5 00:02 test-go-action_linux_amd64.zip
-rwxr-xr-x 1 root root 1728000 Feb  5 00:03 test-go-action_windows_386
-rw-r--r-- 1 root root  930942 Feb  5 00:03 test-go-action_windows_386.zip
-rwxr-xr-x 1 root root 1957376 Feb  5 00:02 test-go-action_windows_amd64
-rw-r--r-- 1 root root  972286 Feb  5 00:02 test-go-action_windows_amd64.zip
```

All compiled and compressed binares are saved under `$GITHUB_WORKFLOW/.release` and thus could be used with any further actions as the files under `$GITHUB_WORKSPACE` directory are persisted for the duration of the run.

Upload

Now that we've produced the binaries, the next logical step in the chain of actions is to upload them somewhere. That could be either to Amazon S3, Github Releases or any other service of your choice.

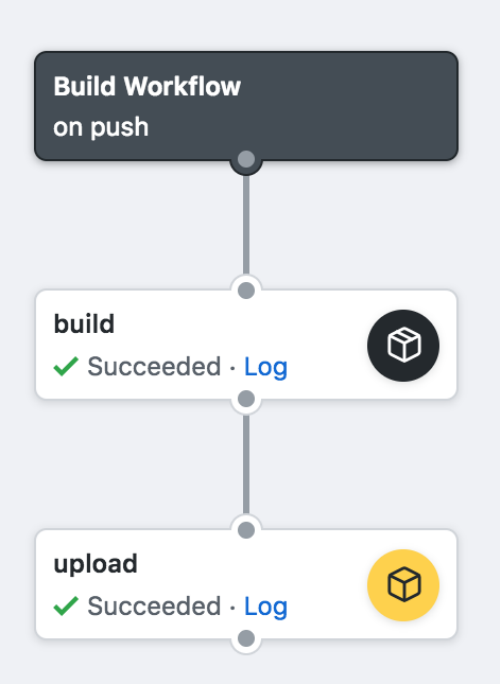
```
workflow "Build Project" {
  on = "push"
  resolves = ["upload"]
}

action "build" {
  uses = "sosedoff/actions/golang-build@master"
}

action "upload" {
  needs = "build"
  uses = "actions/aws/cli@master"
  args = "s3 sync .release s3://my-bucket-name"

  // Make sure to add the secrets in the repo settings page
  // AWS_REGION is set to us-east-1 by default
  secrets = [
    "AWS_ACCESS_KEY_ID",
    "AWS_SECRET_ACCESS_KEY"
  ]
}
```

If everything in the workflow setup correctly you should be able to see a successfull run screen like this:



Extras

Developing actions using live environment is very slow and tends to be error prone. I would recommend looking into nektos/act, a tool to test the actions locally. While it does not 100% replicate the real environment it provides just enough to get the scripts flushed out. Also check out Actions Marketplace, there's a high chance that someone else have already come up with an Action of your interest.