Sebastian Choren
Posted on Mar 22, 2020

Create Container images with Bazel

#bazel #containers #buildtools #goodpractices

If you know bazel, you know how great it is: it is fast and reliable. When you work in projects that uses multiple services, maybe even in different languages, having a build system that is fast and reliable, and more importantly, produces deterministic builds, is key.

You might not be aware, however, how easy it is to use `bazel` to build your container images. You will gain all the benefits from using `bazel` applied to your image build process. Plus, you don't have to deal with ugly `Dockerfiles`.

If you want to see how to implement bazel to build your docker images, keep reading.

Example Project

> You can see the final code and all it's commits in GitHub:
> https://github.com/schoren/example-bazel-containers-hasher

Our project is a password hashing and verification API. It will have two endpoints:

POST /hash

Body:

```
{"plain": "string to hash"}
```

Returns

```
{"hashed": "hashed string"}
```

POST /hash

Body

```
{"hashed": "hashed string", "compare_to": "plaintext string"}
```

Returns

- `200 Ok` if `compare_to` is equivalent to `hashed`
- `406 Not Acceptable` otherwise

Create a new bazel project

For this guide, we asume that you have Bazel and Git installed and configured. Our project files will live on `$GOPATH/src/github.com/<username>/examples-bazel-containers-hasher` (remember to replace `<username>` with your actual GitHub username). Let's start by creating the project folder and initiating a Git repo:

```
mkdir -p $GOPATH/src/github.com/schoren/examples-bazel-containers-hasher
cd $GOPATH/src/github.com/schoren/examples-bazel-containers-hasher
git init .
```

Now, let's setup Bazel so it can build a simple Hello World program in go. For that, we have to create a `WORKSPACE` file in the project root, and load rules_go, including Gazelle:

```
## General rules
load("@bazel_tools//tools/build_defs/repo:http.bzl", "http_archive")

## rules_go
http_archive(
    name = "io_bazel_rules_go",
    sha256 = "142dd33e38b563605f0d20e89d9ef9eda0fc3cb539a14be1bdb1350de2eda659",
```

```
    urls = [
        "https://mirror.bazel.build/github.com/bazelbuild/rules_go/releases/downl
        "https://github.com/bazelbuild/rules_go/releases/download/v0.22.2/rules_g
    ],
)

load("@io_bazel_rules_go//go:deps.bzl", "go_register_toolchains", "go_rules_depen

go_rules_dependencies()

go_register_toolchains()

## Gazelle
http_archive(
    name = "bazel_gazelle",
    sha256 = "d8c45ee70ec39a57e7a05e5027c32b1576cc7f16d9dd37135b0eddde45cf1b10",
    urls = [
        "https://storage.googleapis.com/bazel-mirror/github.com/bazelbuild/bazel-
        "https://github.com/bazelbuild/bazel-gazelle/releases/download/v0.20.0/ba
    ],
)

load("@bazel_gazelle//:deps.bzl", "gazelle_dependencies")

gazelle_dependencies()
```

Gazelle needs you to setup a `BUILD.bazel` files at the project root to define the `//:gazelle` target, and also to set the base package name:

```
load("@bazel_gazelle//:def.bzl", "gazelle")

## This is a gazelle anotation, change the package
# gazelle:prefix github.com/schoren/example-bazel-containers-hasher
gazelle(name = "gazelle")
```

To fetch all the newly added dependencies, just run Gazelle:

```
bazel run //:gazelle
```

```
 ➜ bazel run //:gazelle

INFO: Analyzed target //:gazelle (54 packages loaded, 6990 targets configured).
INFO: Found 1 target...
Target //:gazelle up-to-date:
  bazel-bin/gazelle-runner.bash
  bazel-bin/gazelle
INFO: Elapsed time: 151.773s, Critical Path: 6.48s
INFO: 34 processes: 34 darwin-sandbox.
INFO: Build completed successfully, 47 total actions
INFO: Build completed successfully, 47 total actions
```

Bazel generates and manages a few directories in the workspace root, and those should not be commited into version control, so let's create a `.gitignore` file:

```
/bazel-*
```

Commit your changes:

```
git add .
git commit -m "Setup Bazel with rules_go and Gazelle"
```

Add a hello word example code

Now, let's create the basic structure for our program. Since the main point of this article is the docker part, we will not go too much over the go code.

We'll use Gorilla Mux to handle URL matching, and go's `net/http` package for the actual server.

First, let's initialize `go mod` for this package, so go will handle our dependencies:

```
go mod init
```

And now, let's create a `main` function in `cmd/api`:

```
package main

import (
    "encoding/json"
    "log"
    "net/http"
    "time"
```

```
    "github.com/gorilla/mux"
    "golang.org/x/crypto/bcrypt"
)

func main() {
    r := mux.NewRouter()
    r.HandleFunc("/hash", hashHandler).Methods(http.MethodPost)
    r.HandleFunc("/compare", compareHandler).Methods(http.MethodPost)

    srv := &http.Server{
        Handler:      r,
        Addr:         ":8000",
        WriteTimeout: 1 * time.Second,
        ReadTimeout:  1 * time.Second,
    }

    log.Println("Start serving...")
    log.Fatal(srv.ListenAndServe())
}

type hashRequest struct {
    Plain string `json:"plain"`
}

type hashResponse struct {
    Hashed string `json:"hashed"`
}

func hashHandler(w http.ResponseWriter, r *http.Request) {
    req := hashRequest{}
    err := json.NewDecoder(r.Body).Decode(&req)
    if err != nil {
        log.Printf("Cannot decode hashRequest: %s", err.Error())
        w.WriteHeader(http.StatusBadRequest)
        return
    }

    hashedBytes, err := bcrypt.GenerateFromPassword([]byte(req.Plain), bcrypt.Def
    if err != nil {
        log.Printf("Cannot encrypt password: %s", err.Error())
        w.WriteHeader(http.StatusInternalServerError)
        return
    }

    resp, err := json.Marshal(hashResponse{Hashed: string(hashedBytes)})
```

```
    if err != nil {
        log.Printf("Cannot marshal response json: %s", err.Error())
        w.WriteHeader(http.StatusInternalServerError)
        return
    }

    w.Header().Set("Content-Type", "application/json")
    w.WriteHeader(http.StatusOK)
    w.Write(resp)
}

type compareRequest struct {
    Hashed    string `json:"hashed"`
    CompareTo string `json:"compare_to"`
}

func compareHandler(w http.ResponseWriter, r *http.Request) {
    req := compareRequest{}
    err := json.NewDecoder(r.Body).Decode(&req)
    if err != nil {
        log.Printf("Cannot decode compareRequest: %s", err.Error())
        w.WriteHeader(http.StatusBadRequest)
        return
    }

    err = bcrypt.CompareHashAndPassword([]byte(req.Hashed), []byte(req.CompareTo)
    // the only error we can have here is if there's not a match
    if err != nil {
        w.WriteHeader(http.StatusUnauthorized)
        return
    }

    w.WriteHeader(http.StatusOK)
}
```

Initialize `go mod` and build to configure deps. This is needed so the package is usable with standard go tooling. We will then use `gazelle` to sync bazel dependencies.

```
go build -o /dev/null ./...
bazel run //:gazelle -- update-repos -from_file=go.mod
bazel run //:gazelle
```

The `go build` output is going to `/dev/null` because we are not interested in the built binary, only in updating `go.mod` and `go.sum` dependency files.

We then use `gazelle` to impo the `go.mod` dependencies and insert them in the `WORKSPACE` file.

Finally, we run `gazelle` without any parameters to create or update all the required `BUILD.bazel` files.

Now, we should be able to use `bazel` to build and run the project:

```
bazel build //...
bazel run //cmd/api
```

To test that everything is working as expected, we can use `curl` in a different terminal:

```
$ curl -i localhost:8000/hash -d '{"plain":"text"}'
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 73

{"hashed":"$2a$10$ZqRE.vvvpjHYHvp8HFHO7eGg6RRUS//ctlYPU5sqMYKYzjhAsJIsu"}

$ curl -i localhost:8000/compare -d '{"hashed":"$2a$10$ZqRE.vvvpjHYHvp8HFHO7eGg6R
HTTP/1.1 200 OK
Content-Length: 0

$ curl -i localhost:8000/compare -d '{"hashed":"$2a$10$ZqRE.vvvpjHYHvp8HFHO7eGg6R
401 Unauthorized
Content-Length: 0
```

Nice! We can commit our code:

```
git add .
git commit -m "Add go api code"
```

Now let's see how we can build a docker container for this app.

Add docker support

We will use rules_docker to create the container image. This package offers rules for building generic images, as well as language specific images. We could use the

`go_image` but as it's stated on the docs, it doesn't work in Mac, and we don't want to force developers to use any specific OS, so we have to use the more generic container_image rule.

First, we have to load the rules in our `WORKSPACE` file:

```
## General rules
load("@bazel_tools//tools/build_defs/repo:http.bzl", "http_archive")

## rules_docker
http_archive(
    name = "io_bazel_rules_docker",
    sha256 = "dc97fccceacd4c6be14e800b2a00693d5e8d07f69ee187babfd04a80a9f8e250",
    strip_prefix = "rules_docker-0.14.1",
    urls = ["https://github.com/bazelbuild/rules_docker/releases/download/v0.14.1
)

load(
    "@io_bazel_rules_docker//repositories:repositories.bzl",
    container_repositories = "repositories",
)

container_repositories()

load("@io_bazel_rules_docker//repositories:deps.bzl", container_deps = "deps")

container_deps()

load("@io_bazel_rules_docker//container:pull.bzl", "container_pull")

container_pull(
    name = "alpine_linux_amd64",
    registry = "index.docker.io",
    repository = "library/alpine",
    tag = "3.8",
)

## rules_go
http_archive(
    name = "io_bazel_rules_go",
    sha256 = "142dd33e38b563605f0d20e89d9ef9eda0fc3cb539a14be1bdb1350de2eda659",
    urls = [
        "https://mirror.bazel.build/github.com/bazelbuild/rules_go/releases/downl
        "https://github.com/bazelbuild/rules_go/releases/download/v0.22.2/rules_g
```

```
    ],
)

# ...
```

Note that the order of bazel rules loading is not important, but we prefer to leave the go rules at the bottom, because `gazelle` adds dependencies at the bottom of the file.

Build images

Now we have to declare a new target that will create a docker image. Update the `cmd/api/BUILD.bazel` file so it looks like this:

```
load("@io_bazel_rules_go//go:def.bzl", "go_binary", "go_library")
load("@io_bazel_rules_docker//container:container.bzl", "container_image")

go_library(
    name = "go_default_library",
    srcs = ["main.go"],
    importpath = "github.com/schoren/example-bazel-containers-hasher/cmd/api",
    visibility = ["//visibility:private"],
    deps = [
        "@com_github_gorilla_mux//:go_default_library",
        "@org_golang_x_crypto//bcrypt:go_default_library",
    ],
)

go_binary(
    name = "api",
    embed = [":go_default_library"],
    visibility = ["//visibility:public"],
)

container_image(
    name = "image",
    base = "@alpine_linux_amd64//image",
    entrypoint = ["/api"],
    files = [":api"],
)
```

Now, test the new target:

```
bazel build //cmd/api:image
```

This command will build a `tar` file that can be imported to docker. You can `docker load` the file manually, or use baze to do that:

```
bazel run //cmd/api:image
```

Now, if you run `docker images` you should see this:

```
REPOSITORY      TAG    IMAGE ID            CREATED             SIZE
bazel/cmd/api   image  e793d723ef4f        50 years ago        10.8MB
```

Now you can run the image using docker:

```
docker run --rm -it -p8000:8000 bazel/cmd/api:image
```

You can again use `curl` to test it's working:

```
$ curl -i localhost:8000/hash -d '{"plain":"text"}'
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 73

{"hashed":"$2a$10$ZqRE.vvvpjHYHvp8HFHO7eGg6RRUS//ctlYPU5sqMYKYzjhAsJIsu"}
```

Let's commit the changes:

```
git add .
git commit -m "Add docker support"
```

> **Note for mac users**
>
> By default, bazel will compile binaries for the platform it's running on. So, when you run these commands, you will end up with a binary compiled for MacOS. This binary won't be compatible with Linux.
>
> However, the docker image is Linux, so the file won't run. It will show an error like this:

```
$ docker run --rm -it -p8000 bazel/cmd/api:image
standard_init_linux.go:211: exec user process caused "exec format error"
```

> To solve this, you have to add a `--platform` flag to the command:

```
azel run --platforms=@io_bazel_rules_go//go/toolchain:linux_amd64 //cmd/api:image
ocker run --rm -it -p8000 bazel/cmd/api:image
0/03/21 20:57:17 Start serving...
```

> If you plan to run all your binaries within docker (which you should), you can automate this by using bazel RC files. Run

```
build --platforms=@io_bazel_rules_go//go/toolchain:linux_amd64
run --platforms=@io_bazel_rules_go//go/toolchain:linux_amd64
```

Publish images to DockerHub

You probably noticed that your container name starts with `bazel/`. This is not only ugly, but it also makes it impossible to push:

```
$ docker push bazel/cmd/api
The push refers to repository [docker.io/bazel/cmd/api]
e90f26cebdee: Preparing
7444ea29e45e: Preparing
denied: requested access to the resource is denied
```

Also, the tag is the name of the target, `image` in our case. That is also not very useful when deploying this image.

To solve the first problem, we can use the `repository` attribute in the `container_image` rule in `cmd/api/BUILD.bazel`. Replace `<username>` with your DockerHub ID, or any repository ID:

```
container_image(
    name = "image",
    base = "@alpine_linux_amd64//image",
    entrypoint = ["/api"],
    files = [":api"],
    repository = "<username>"
)
```

Now when you run `bazel run //cmd/api:image` it will save the image as `<username>/cmd/api:image`

Again we could manually call `docker push <username>/cmd/api` to push our image, but we can also use the `docker_push` rule to automate this for us. Add this to `cmd/api/BUILD.bazel`:

```
container_push(
    name = "image-push",
    format = "Docker",
    image = ":image",
    registry = "index.docker.io",
    repository = "<username>/cmd-api",
)
```

> Depending on what repository you are using, it might support nested repositories (i.e. ECR). In that case, you can make `repository` something like `"<username>/cmd/api"` which looks a bit nicer

Now bazel can push images for you:

```
$ bazel run //cmd/api:image-push
INFO: Analyzed target //cmd/api:image-push (0 packages loaded, 0 targets configur
INFO: Found 1 target...
Target //cmd/api:image-push up-to-date:
  bazel-bin/cmd/api/image-push.digest
  bazel-bin/cmd/api/image-push
INFO: Elapsed time: 0.241s, Critical Path: 0.00s
INFO: 0 processes.
INFO: Build completed successfully, 1 total action
INFO: Build completed successfully, 1 total action
2020/03/21 18:43:59 Successfully pushed Docker image to index.docker.io/schoren/e
```

Commit:

```
git add .
git commit -m "Add push support to bazel"
```

Conclusion

You can now use bazel to manage your container images development lifecycle: it can build and push images to repositories, with all the benefits of bazel: fast and reproducible builds.

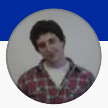
In a small example like this you might not see the benefits right away, but in a more complex project, composed of several microservices (running inside containers), it is a great way to reduce build and CI times.

Discussion (1)

Per Halvor Tryggeseth • Aug 21 '20

It seems that the repository that is referred to (github.com/schoren/example-bazel-c...) is available on github.com/speak2jc/examples-bazel... with a few changes ...

Code of Conduct • Report abuse



Sebastian Choren

LOCATION
Buenos Aires, Argentina

JOINED
Feb 28, 2018

Trending on DEV Community 🔥

The Pomodoro Technique: Productivity hacks
#productivity

Emmet Series1🎉- Boost your productivity
#webdev #html #css #productivity

How many meetings do you have per week?
#discuss #career