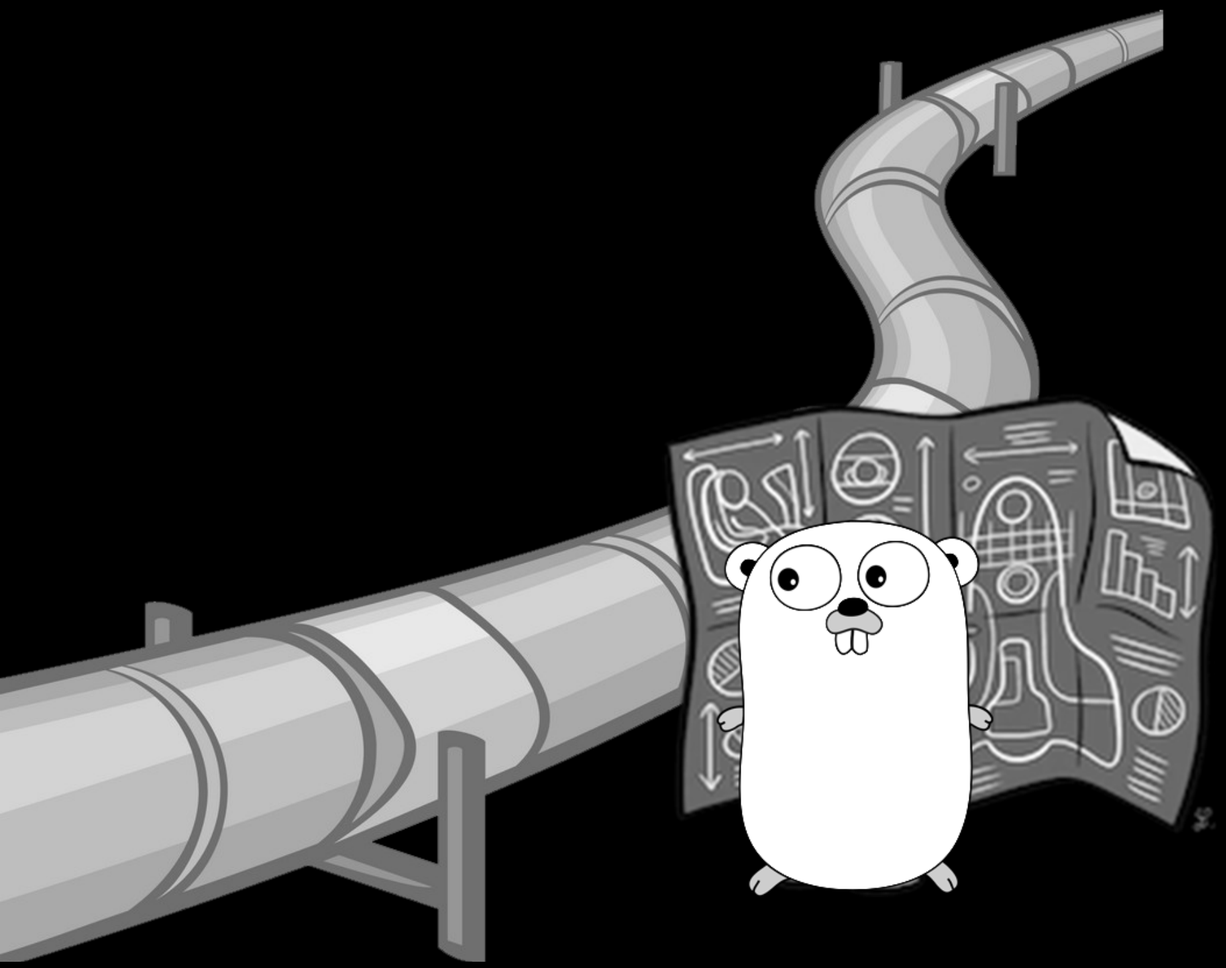
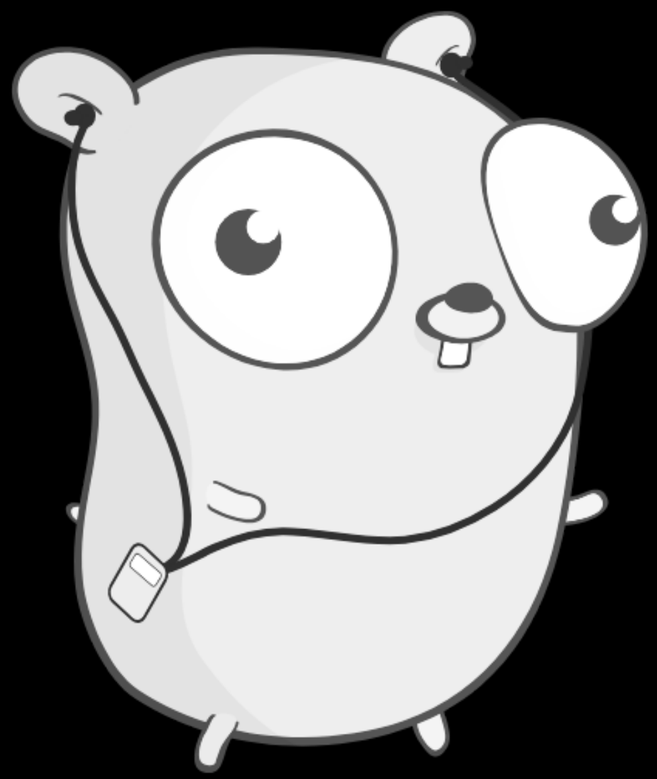


CI Pipeline as Code

Go Meetup Bern - 25. Juli 2023

acend
experience knowledge

PUZZLE ITC



About Me

Christian Schlatter

Working as:

- CICD engineer at Puzzle ITC, Bern
- Trainer at Acend for cloud-native technologies

Also well known as:

- Only Windows user in the whole company

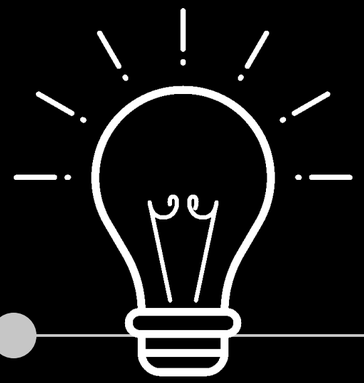
acend
experience knowledge

PUZZLE ITC

Continuous Integration

Continuous Integration is the idea of integrate, test, and build code as frequent as possible

History of CI



1994

The Concept of Continuous Integration

2001
Cruise Control

First open source Continuous Integration tool

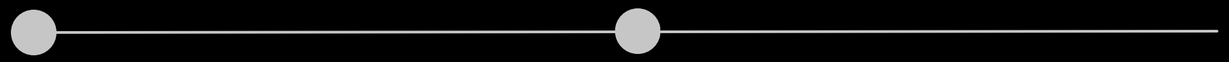
2006
Jenkins CI

The "Plug-in Hell Machine"

2014
GitLab

All in one DevOps system

History of CI



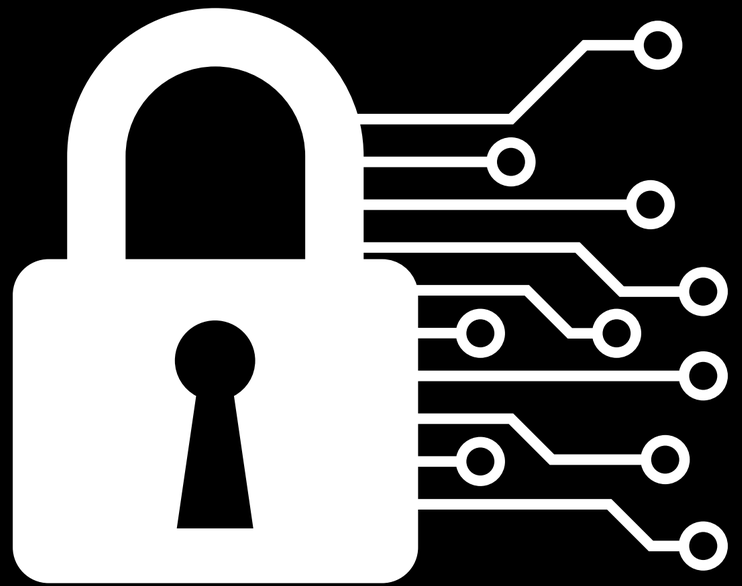
2018
GitHub Actions

2018
Tekton

Kubernetes cloud-native
container based CI tool

Vendor Lock-In

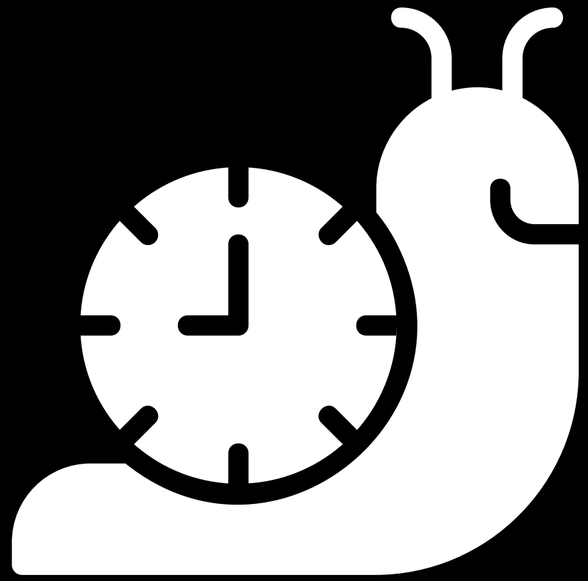
- Different CI tool - different world
- Switch to different CI tool is expensive
- No reuse across different CI tools



Slow Feedback

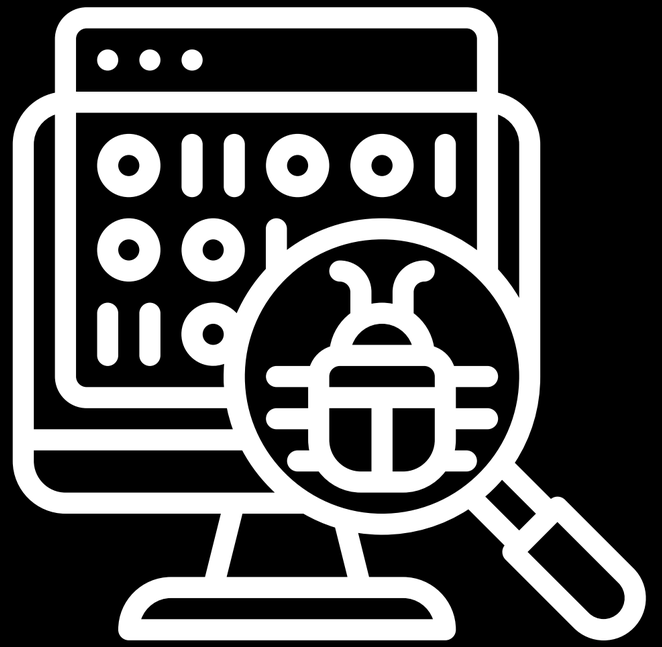
Change, Commit, Wait, Validate, Repeat

- Tight system coupling
- Complicated flow control
- Testing is hard



Bad Observability

- Bottlenecks are hard to track
- No monitoring & tracing standards
- No debugging capabilities



Future of CI?

2018
GitHub Actions

2018
Tekton

Kubernetes cloud-native container based CI tool

2022
Dagger

Container based CI tool

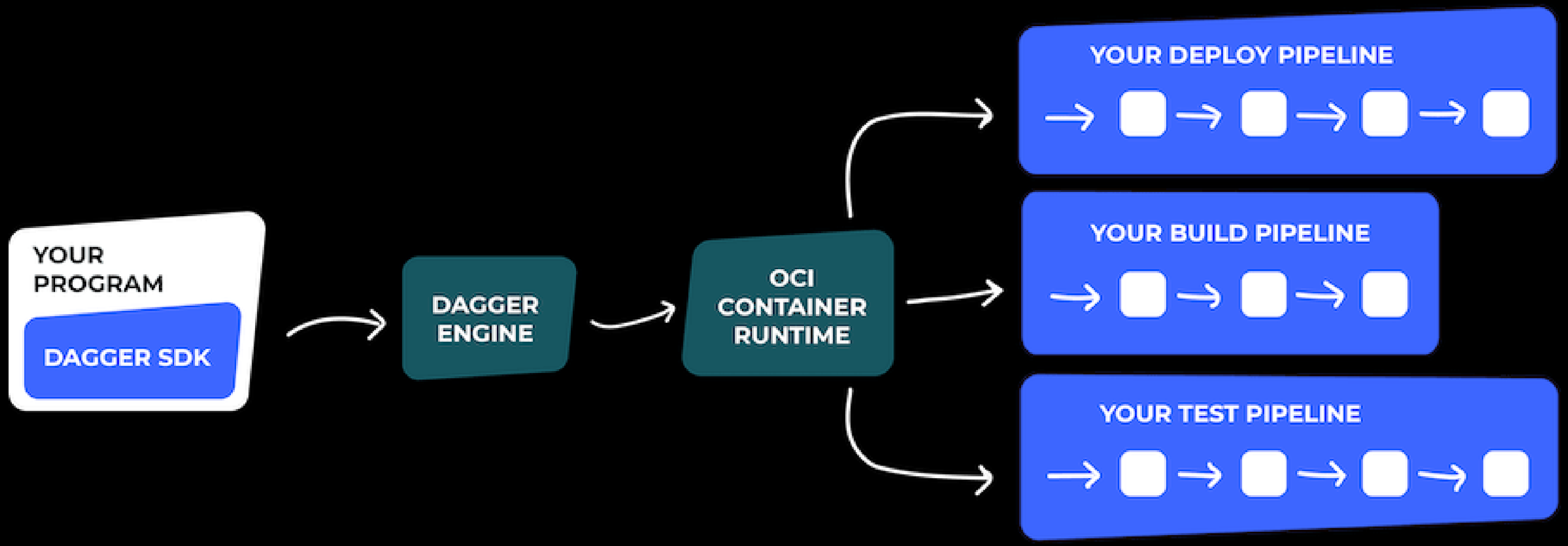
2023

Dagger

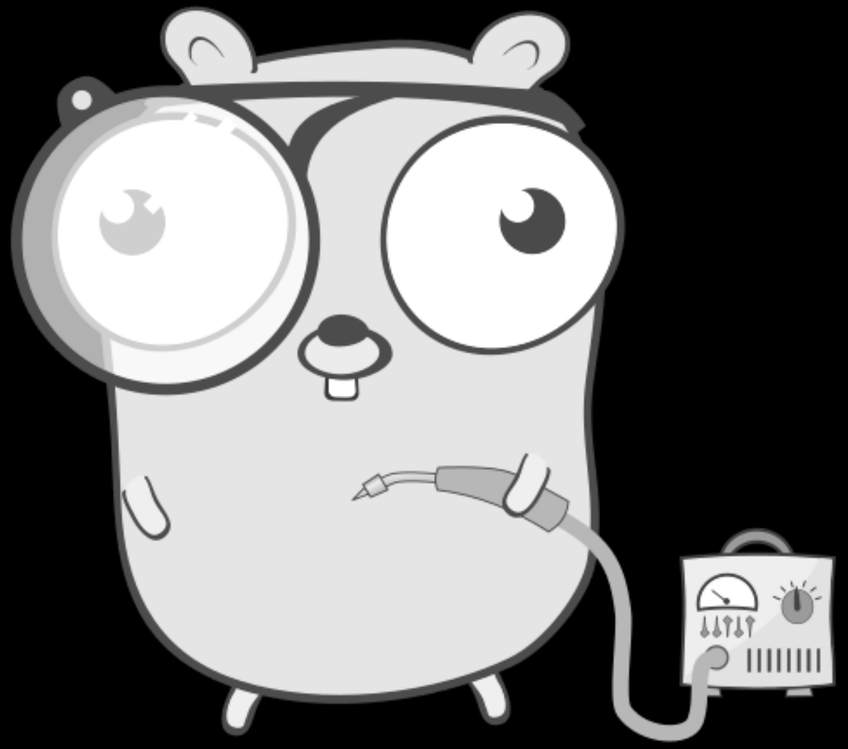
« Dagger is a programmable CI/CD engine that runs your pipelines in containers »

www.dagger.io

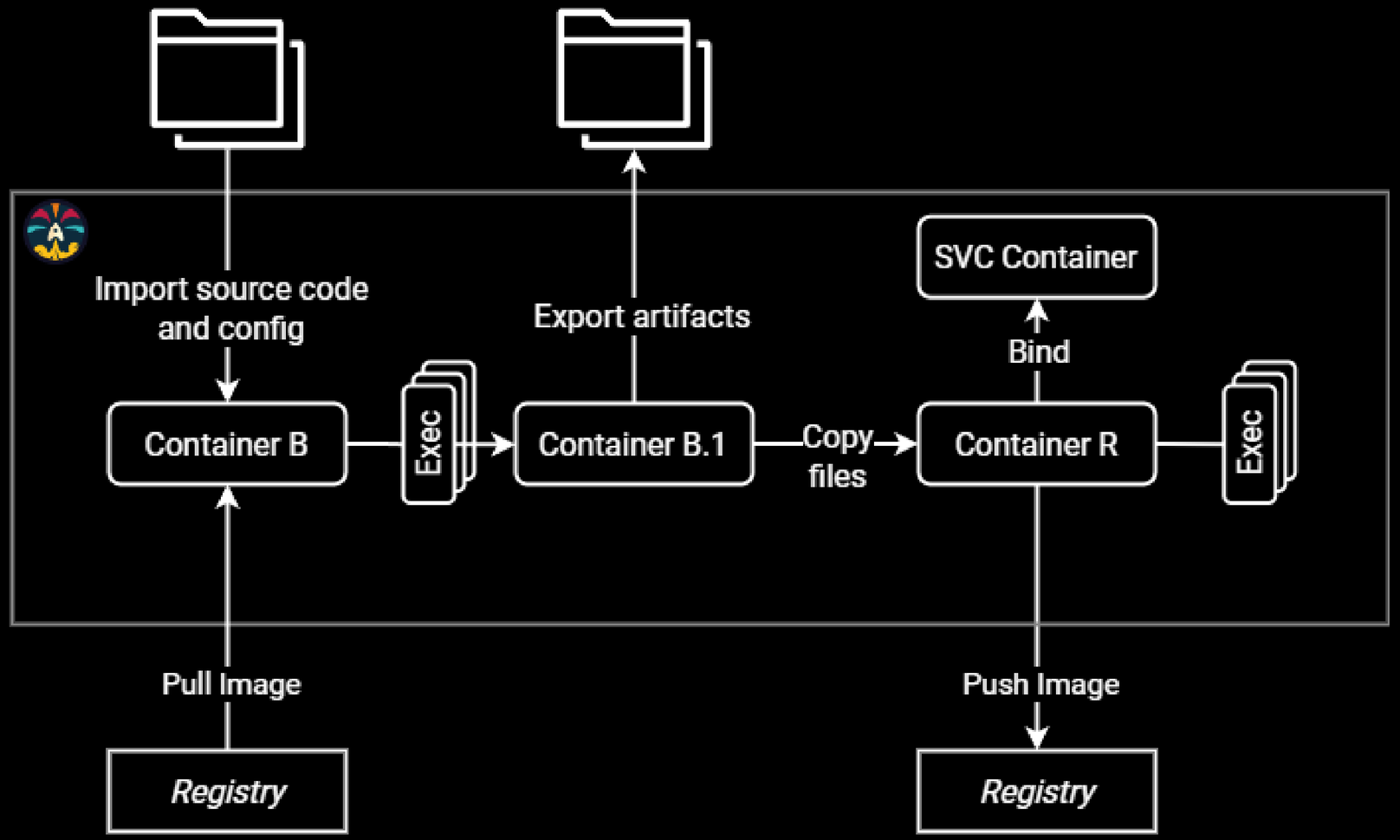
How it works


YOUR
PROGRAM
DAGGER SDK
DAGGER
ENGINE
OCI
CONTAINER
RUNTIME
YOUR DEPLOY PIPELINE
YOUR BUILD PIPELINE
YOUR TEST PIPELINE

Showcase

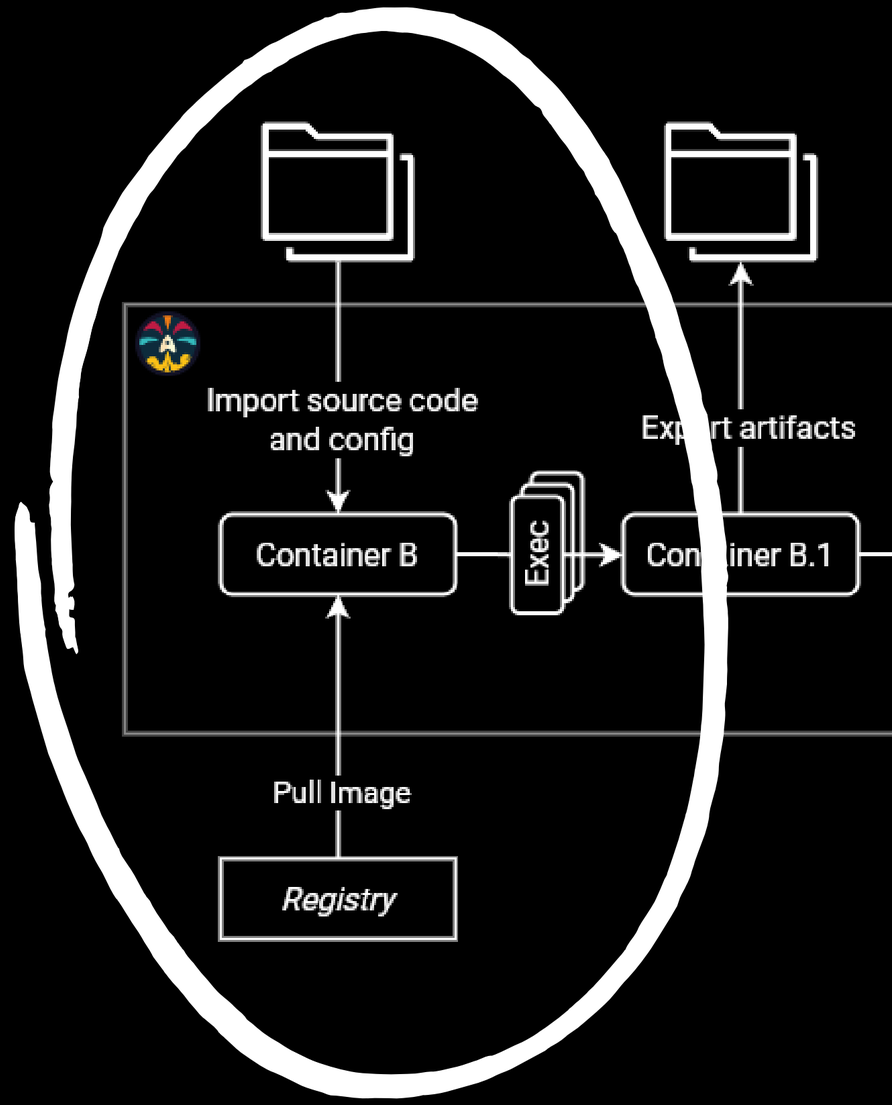


Build Go apps with Dagger



Build Go app

```go
func build(ctx context.Context) error {
    // initialize Dagger client
    client, err := dagger.Connect(ctx, dagger.WithLogOutput(os.Stdout))
    if err != nil {
        return err
    }
    defer client.Close()

    // get reference to the local project
    src := client.Host().Directory(".")

    // get `golang` image
    golang := client.Container().From("golang:latest")

    // mount cloned repository into `golang` image
    golang = golang.WithDirectory("/src", src).WithWorkdir("/src")

    // define the application build command
    path := "build/"
    golang = golang.WithExec([]string{"go", "build", "-o", path})

```



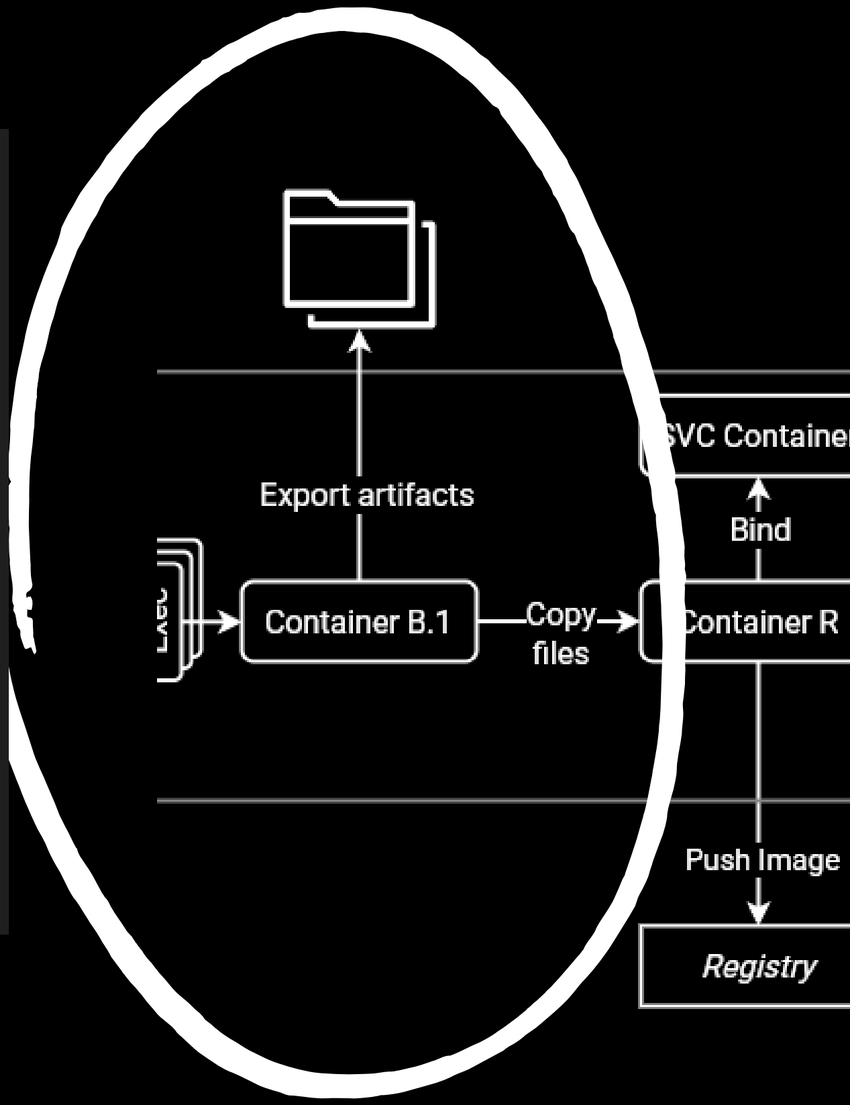
Build Go app

```go
func build(ctx context.Context) error {
    // initialize Dagger client
    client, err := dagger.Connect(ctx, dagger.WithLogOutput(os.Stdout))
    if err != nil {
        return err
    }
    defer client.Close()

    // get reference to the local project
    src := client.Host().Directory(".")

    // get `golang` image
    golang := client.Container().From("golang:latest")

    // mount cloned repository into `golang` image
    golang = golang.WithDirectory("/src", src).WithWorkdir("/src

    // define the application build command
    path := "build/"
    golang = golang.WithExec([]string{"go", "build", "-o", path})

```

```
FROM golang:latest

COPY . /src

WORKDIR /src

RUN go build -o build/
```

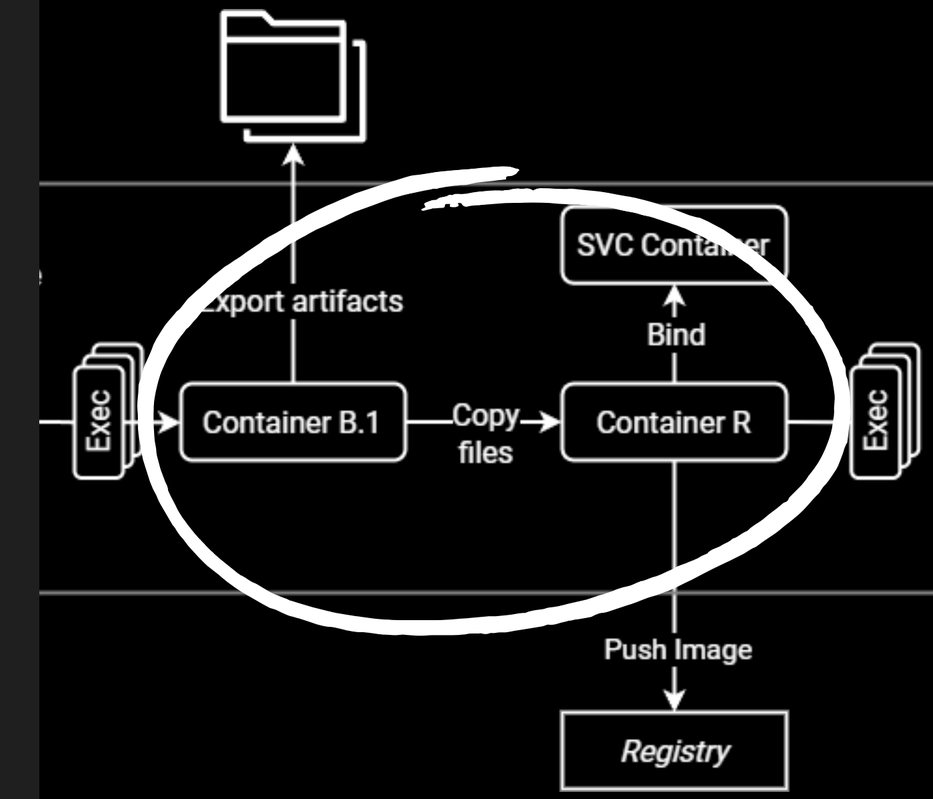
Export files to host

```go
// build app
builder := client.Container().
    From("golang:latest").
    WithDirectory("/src", project).
    WithWorkdir("/src").
    WithEnvVariable("CGO_ENABLED", "0").
    WithExec([]string{"go", "build", "-o", "myapp"})

// export binary to host
_, err = builder.File("/src/myapp").Export(ctx, "./build")
if err != nil {
    panic(err)
}
```



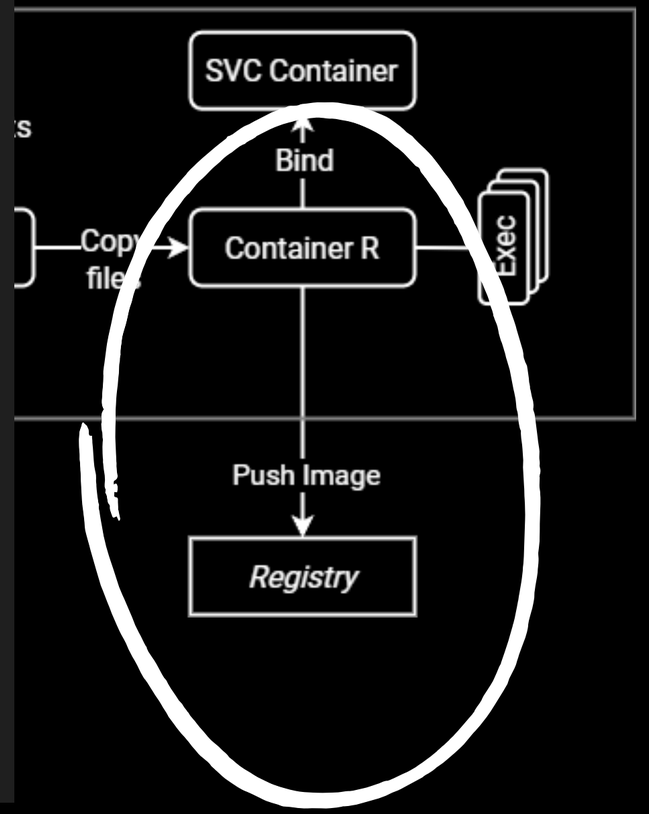
Multistage build with Dagger

```
project := client.Host().Directory(".")

// build app
builder := client.Container().
    From("golang:latest").
    WithDirectory("/src", project).
    WithWorkdir("/src").
    WithEnvVariable("CGO_ENABLED", "0").
    WithExec([]string{"go", "build", "-o", "myapp"})

// publish binary on alpine base
prodImage := client.Container().
    From("alpine").
    WithFile("/bin/myapp", builder.File("/src/myapp")).
    WithEntrypoint([]string{"/bin/myapp"})
```



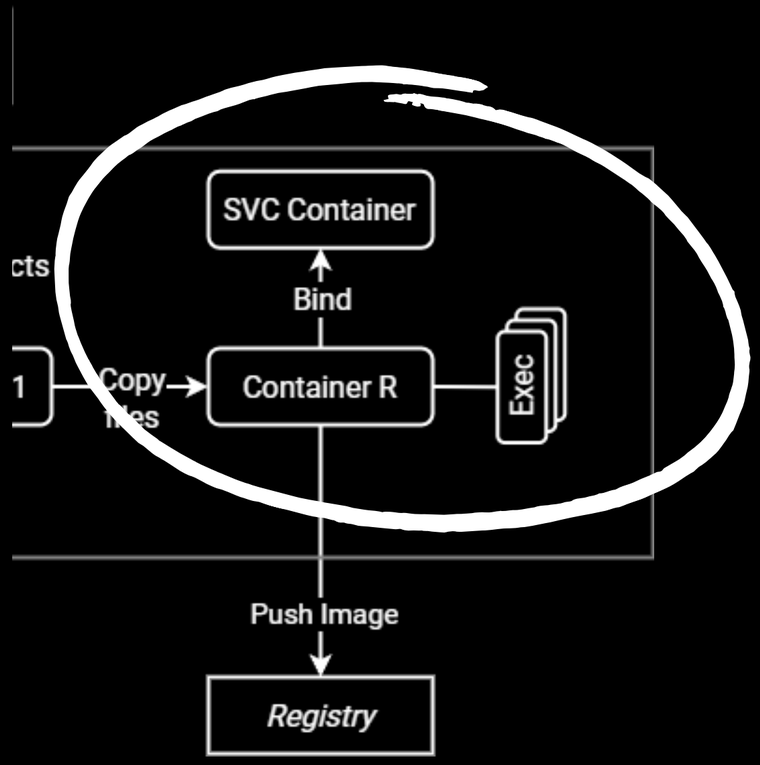
Push image

```go
// get host directory
project := client.Host().Directory(".")

// build app
builder := client.Container().
    From("golang:latest").
    WithDirectory("/src", project).
    WithWorkdir("/src").
    WithEnvVariable("CGO_ENABLED", "0").
    WithExec([]string{"go", "build", "-o", "myapp"})

// publish binary on alpine base
prodImage := client.Container().
    From("alpine").
    WithFile("/bin/myapp", builder.File("/src/myapp")).
    WithEntrypoint([]string{"/bin/myapp"})

addr, err := prodImage.Publish(ctx, "localhost:5000/multistage")
if err != nil {
    panic(err)
}
```

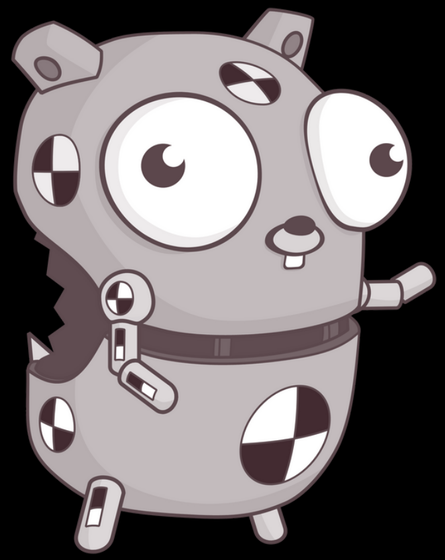


Service bindings

```
redis := daggerClient.Container().From("redis").
    WithExposedPort(6379)

_, err := golang.
    WithServiceBinding("redis", redis).
    WithExec([]string{"go", "test", "./...", "-v"}).Sync(ctx)

return err
```

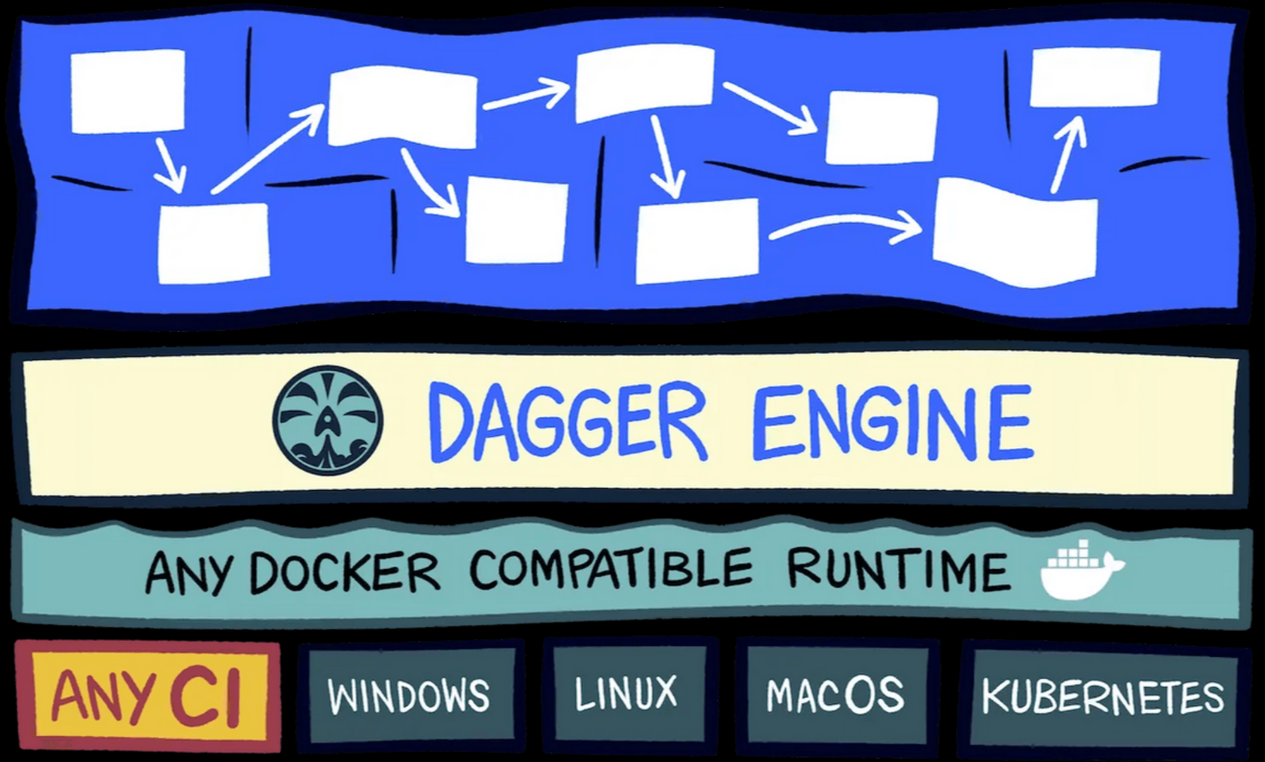


Complex Pipeline with Dagger



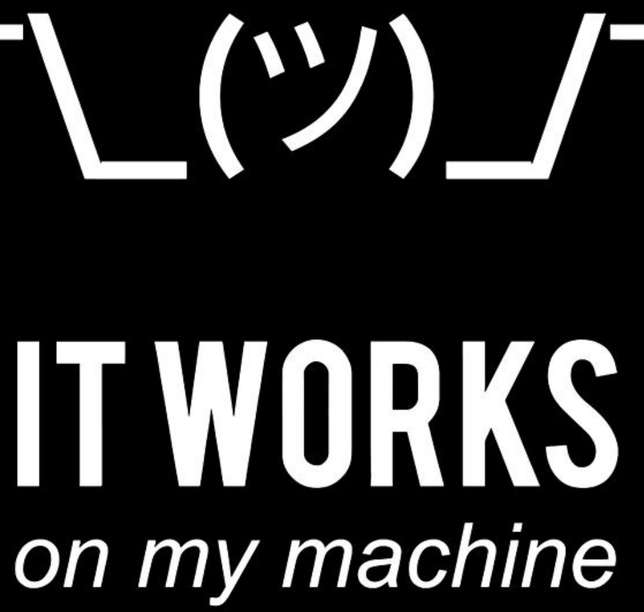
https://github.com/puzzle/goff/blob/main/ci/main.go

No vendor lock-in



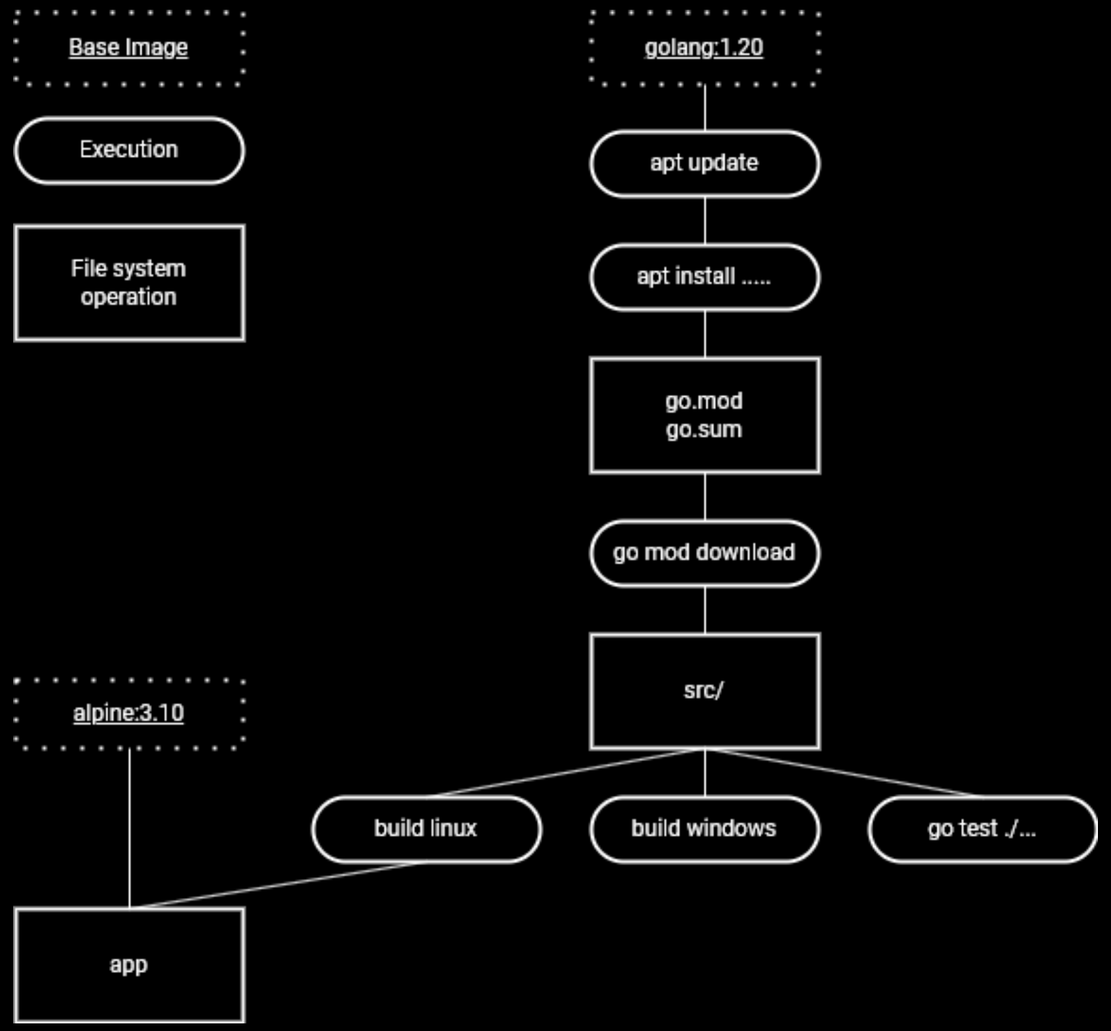
Fast Feedback

- **Instant local testing**
- Decoupled from CI tooling and Git



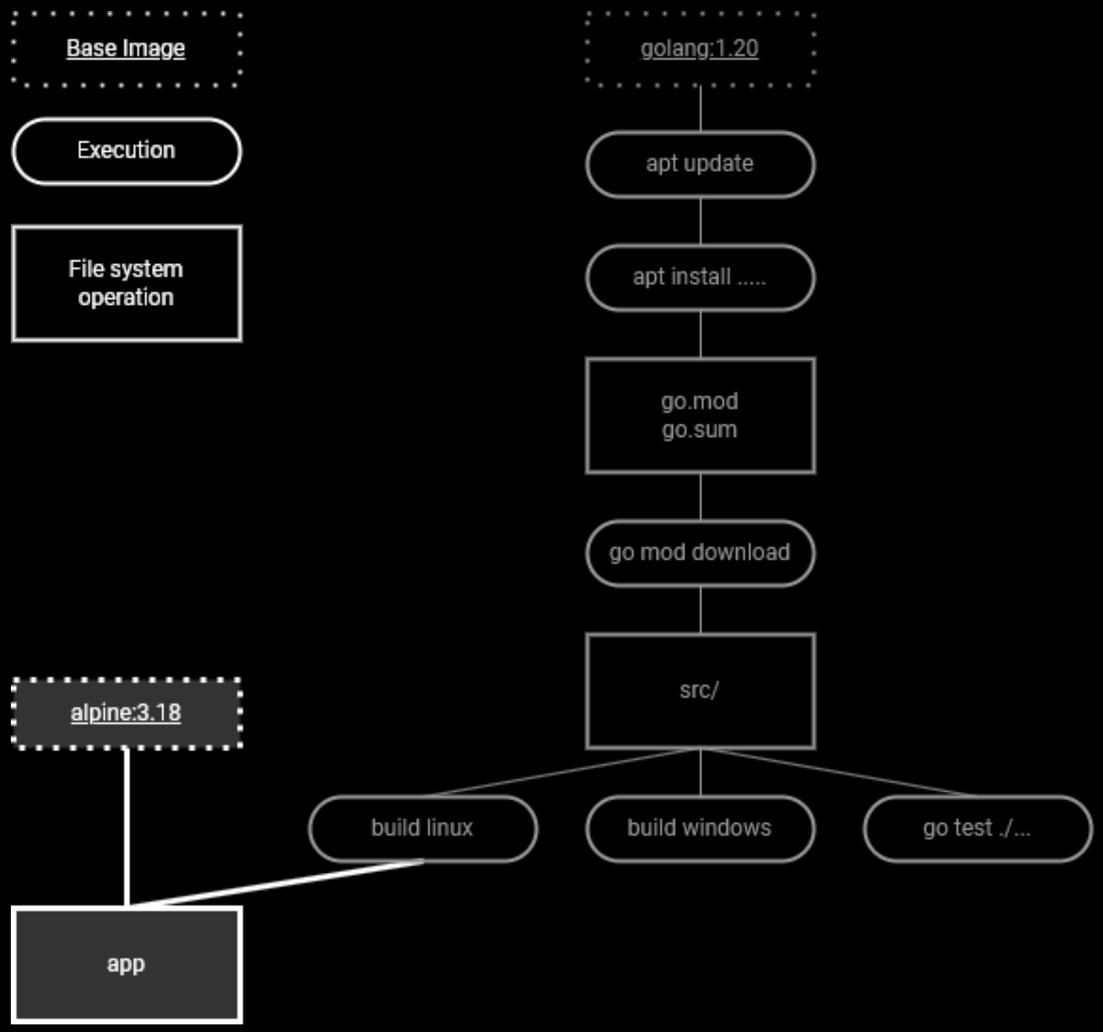
Fast Feedback

- Instant local testing
- Decoupled from CI tooling and Git
- Superior caching based on Buildkits LLB-Graph



Fast Feedback

- Instant local testing
- Decoupled from CI tooling and Git
- Superior caching based on Buildkits LLB-Graph
- "Jobs" only execute when necessary



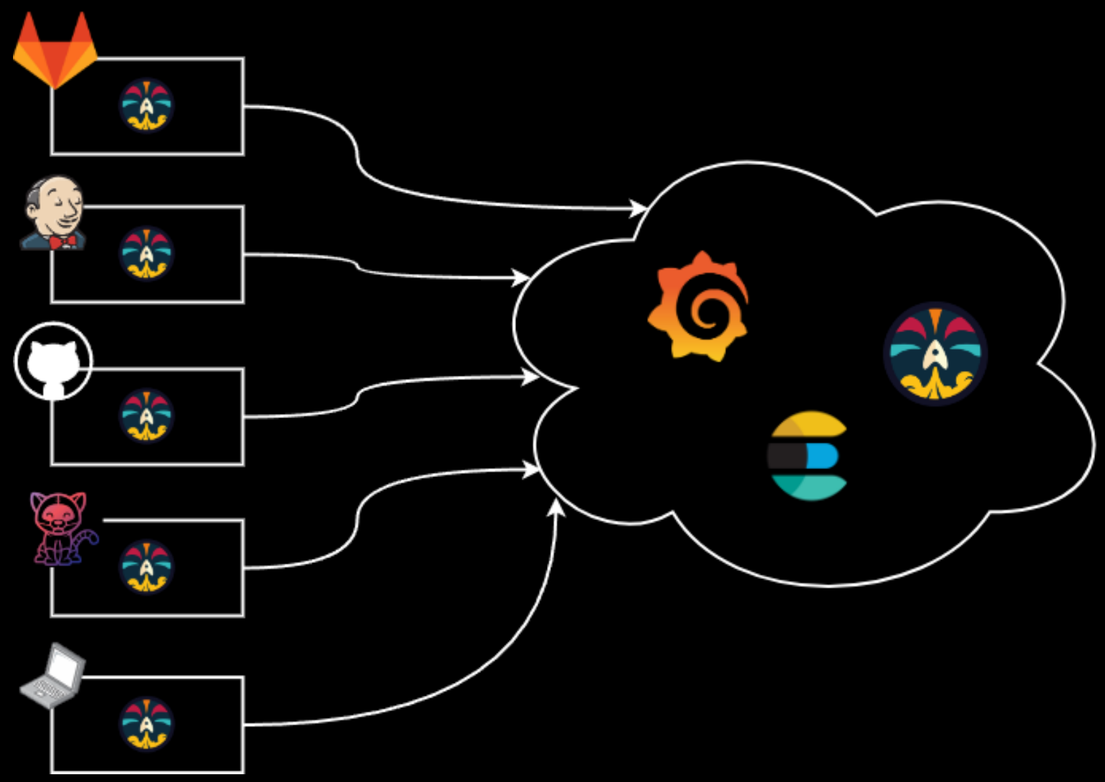
Observability

Open Telemetry integration

- Send metrics from anywhere

Integrate into

- Grafana
- Elasticsearch
- Dagger cloud



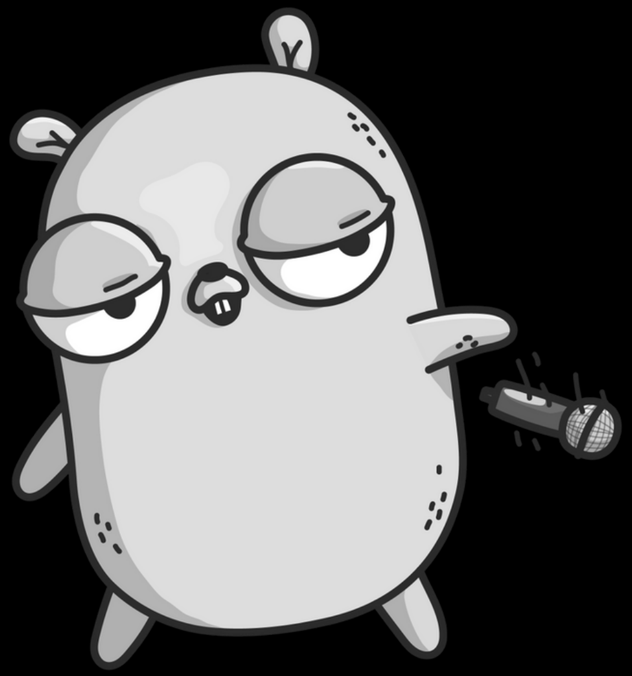
Why using Dagger? 1/2

- Instant local testing! -> Fast Feedback
- Portability -> No more vendor lock-in
- Superior caching -> Fast Feedback
- Compatibility with the Docker ecosystem
- Cross-platform tracing & monitoring -> Better observability

Why using Dagger? 2/2

- Make use of Go language features
  - Static typed
  - Concurrency features
  - Go test
  - Go modules
  - .....
- Native flow and execution control

Questions?



acend
experience knowledge

PUZZLE ITC