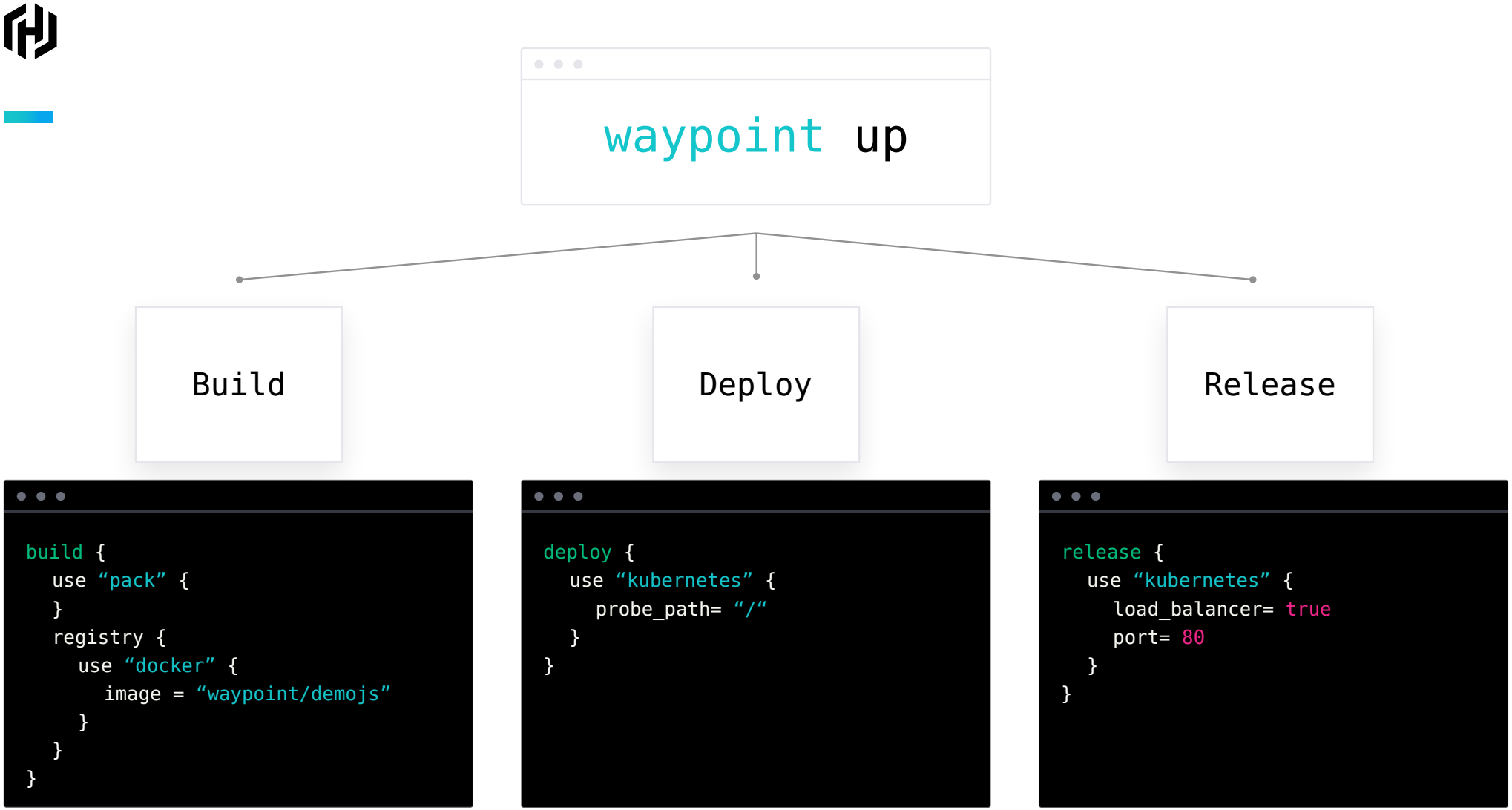
Waypoint Deep-dive

waypoint up

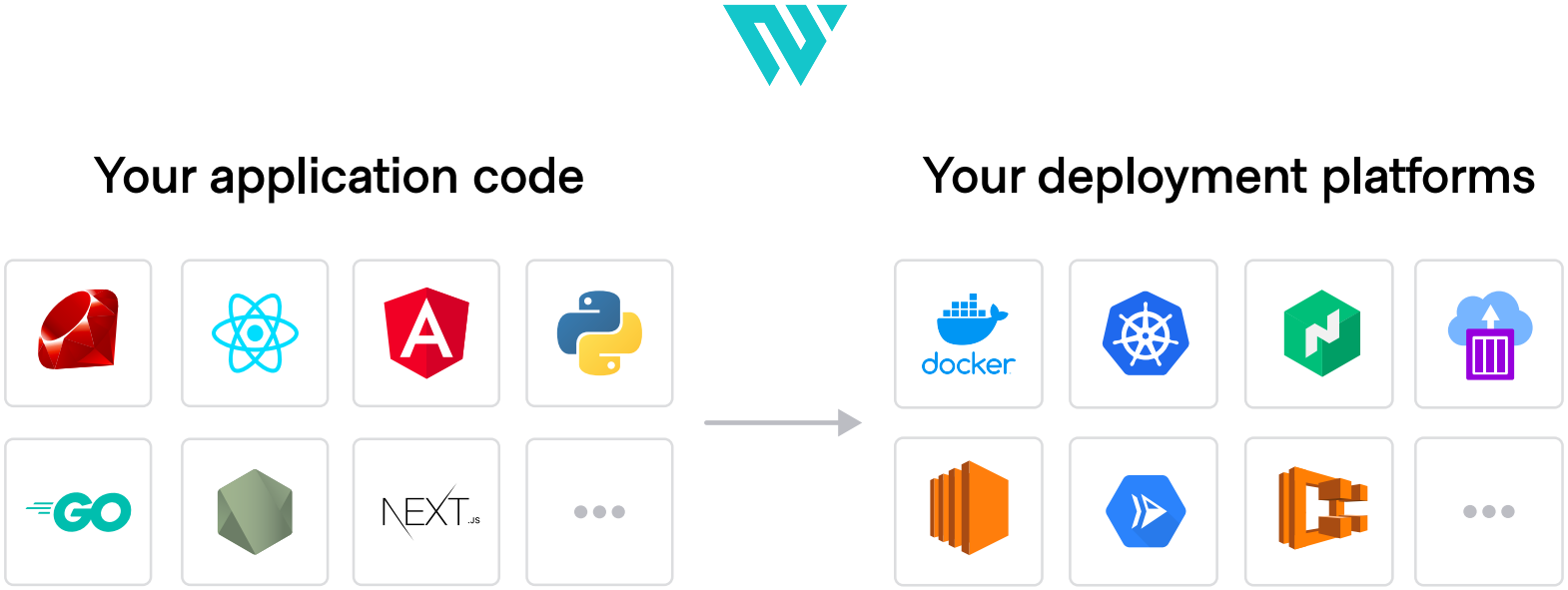
```
$ waypoint up

» Building...
✓ Initializing Docker client...
✓ Building image...
✓ Injecting Waypoint Entrypoint...

Tagging Docker image: waypoint.local/web:latest => hashicorp/wp-demo:latest

Docker image pushed: hashicorp/wp-demo:latest

» Deploying...
✓ Deployment successfully rolled out!

» Releasing...
✓ Service successfully configured!

The deploy was successful! A Waypoint deployment URL is shown below. This can be used internally to check
your deployment and is not meant for external traffic. You can manage this hostname using "waypoint hostname."

Release URL: http://192.168.1.79
Deployment URL: https://implicitly-new-whale--v1.alpha.waypoint.run
```


waypoint up
Build
Deploy
Release
build {
use “pack” {
}
registry {
use “docker” {
image = “waypoint/demojs”
}
}
}
deploy {
use “kubernetes” {
probe_path= “/“
}
}
release {
use “kubernetes” {
load_balancer= true
port= 80
}
}

Waypoint configuration file

- One configuration file, one common language, as code
- End-to-End build, deploy, and release for applications
- An interface for developers

```
# waypoint.hcl

project = “my-project”

app “wp-react” {

    build {
        use “pack” {
        }
        registry {
            use “docker” {
                image = “waypoint/demojs”
            }
        }
    }

    deploy {
        use “kubernetes” {
                probe_path=“/“
        }
    }

    release {
        use “kubernetes” {
            load_balancer=true
            port=80
        }
    }
}
```


Your application code
Your deployment platforms
docker
GO
NEXT.JS

Common workflow integrating various components

Waypoint Deep-dive

TOC

Configuration language

Client / Server architecture

Waypoint entrypoint

Plugin architecture

Configuration language

Config file

HCL.

As expected, this an HCL formatted file with all the benefits, including also supporting JSON.

Project based.

Multiple applications can be specified for projects using a distribute architecture.

Functions.

One big benefit to HCL is the ability to execute functions in configuration.

Config file

Functions

```
# waypoint.hcl

project = “my-project”

app “wp-react” {

    build {
        use “pack” {
        }
        registry {
            use “docker” {
                image = “waypoint/demojs”
            }   tag = gitrefpretty()
        }   }
    }   }
    }
    deploy {
        use “kubernetes” {
                probe_path=“/“
        }
    }

    release {
        use “kubernetes” {
            load_balancer=true
            port=80
        }
    }
}
```

Config file

Workflow

Full picture of application.

The entire lifecycle of production can be seen at once.

Common interface.

No matter how different the platforms may be, they are available in the same format.

Multiple applications.

Extending the workflow across application boundaries has big value.

Config file

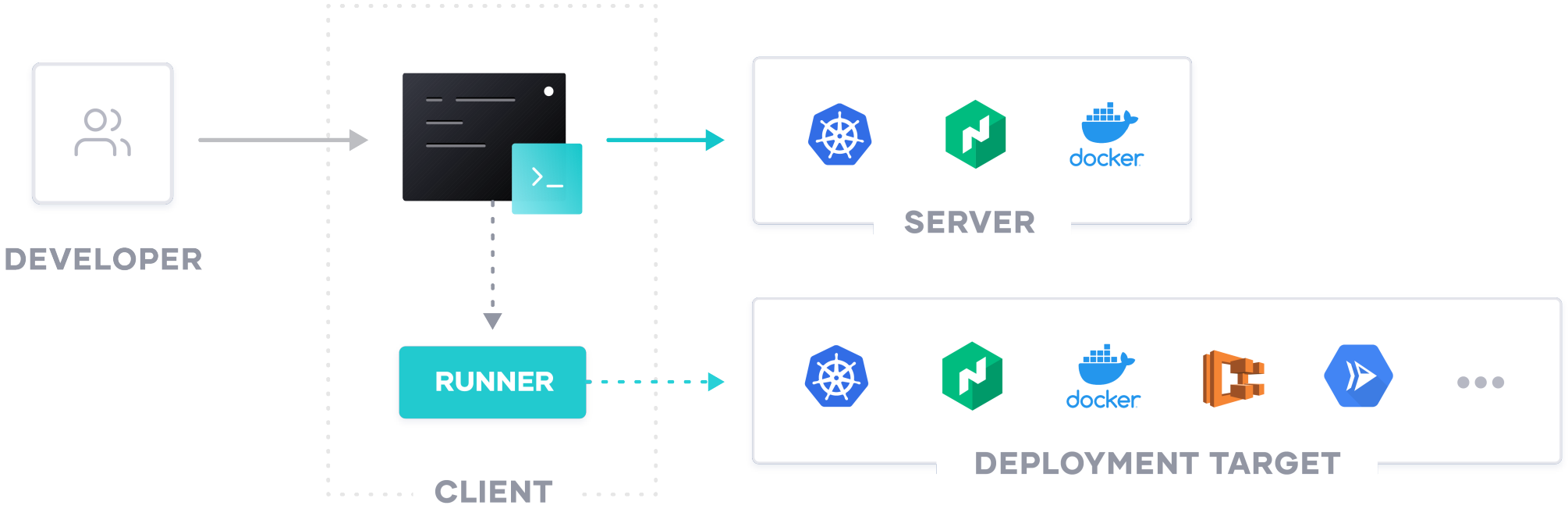
Functions

```
# waypoint.hcl

project = “my-project”

app “wp-react” {

    build {
        use “pack” {
        }
        registry {
            use “docker” {
                image = “waypoint/demojs”
            }
        }
    }

    deploy {
        use “kubernetes” {
                probe_path=“/“
        }
    }

    release {
        use “kubernetes” {
            load_balancer=true
            port=80
        }
    }
}
```

```
app “login” {

    build {
        use “pack” {
        }
        registry {
            use “docker” {
                image = “waypoint/login”
            }
        }
    }

    deploy {
        use “google-cloud-run” {
                probe_path=“/“
        }
    }
}
```

Client / Server architecture

Server architecture



Server

Which tracks and coordinates actions.

Provides a catalog.

All artifacts, deploys, and releases are tracked.

Provides instance functionality.

Instances connect back to the server, providing features such as exec and logs.

UI.

Establishes a place for multiple users to interact with their applications.

Client

Who requests all actions.

gRPC.

A standard gRPC client is used, allowing easy integration with the server.

CLI.

Builtin CLI is implemented via public APIs.

Queueing.

Thin clients can still trigger actions such as builds and deploys to happen remotely.

Runner

Where all actions take place.

Driven by server.

Runners connect to a server and execute jobs.

Context aware.

Runners can execute in the context of security environments to give plugins credentials.

Automatic.

CLIs register themselves as a runner, allowing jobs to run where users expect them.

Waypoint entrypoint

Unlocking runtime functionality

Waypoint entrypoint

Runtime

Functionality

Runs inside each deployment.

Automatically injected by pack and docker, can be included manually as well.

Connects back to server.

Using deployment configuration, connects back to the server on start.

Executes application.

Entrypoint is run first to provide the runtime functionality, then executes the application.

Waypoint entrypoint

Security

Makes only outbound connections.

Does not listen on the network, only connects to the server which provides an RPC channel.

Not in release URL path.

Application access via the Release URL is direct, not via the entrypoint.

Capability based tokens.

Tokens used only have access to specific Entrypoint APIs, protecting server data.

Waypoint entrypoint

Security

Fully open source.

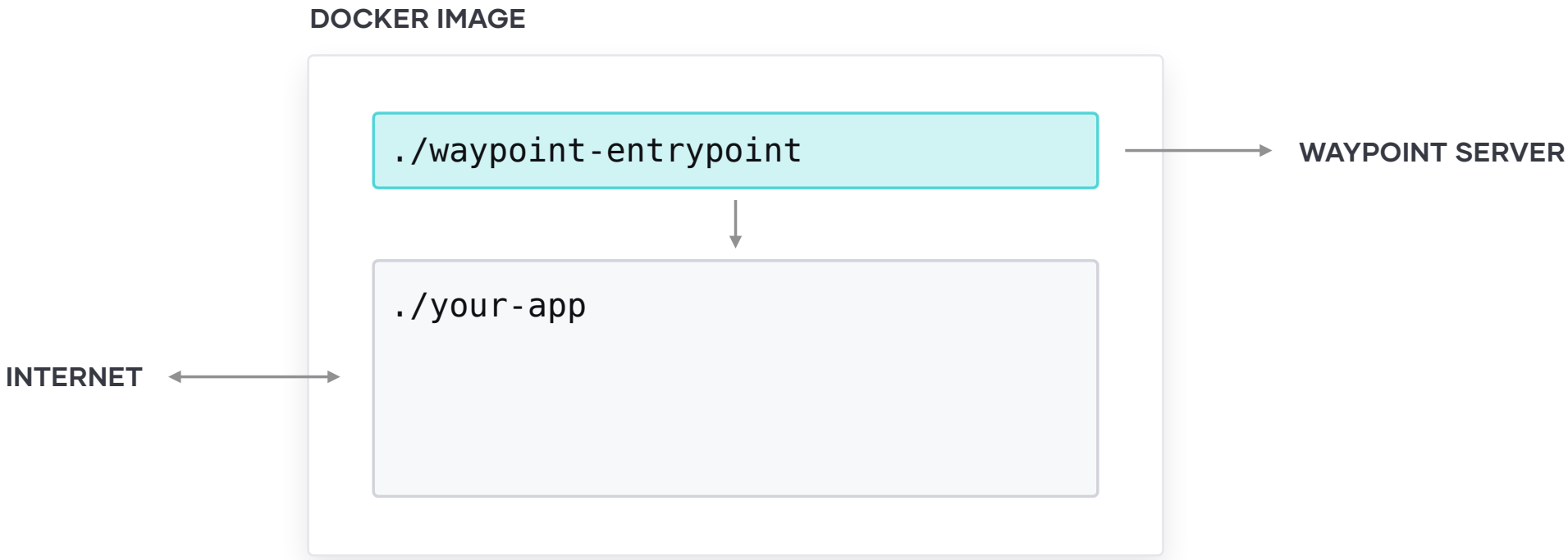
As with all of Waypoint, this component is fully open for your inspection.

Optional.

Waypoint can be used without the entrypoint, which means forgoing its provided features.

Inside Docker

Injected as the Image Entrypoint, executes your app after initialization.



Automatic URLs

Courtesy of the Waypoint URL service

URLs everywhere

Get deployment URLs
no matter the platform.

Powered by HashiCorp services.

Ingress operated by HashiCorp for use by Waypoint users.

Automatic.

Registration on setup for zero configuration.

URLs everywhere

Get deployment URLs no matter the platform.

HTTP backend only.

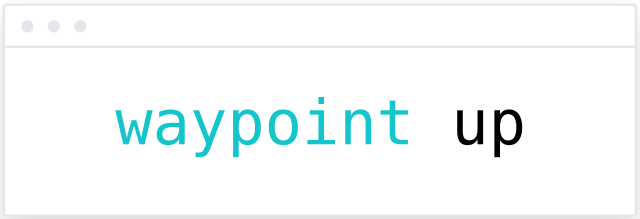
Currently only supports HTTP services.

Usage limits.

Request per second and bandwidth limits in effect to prevent abuse.

Self-hosted available.

All code to run your own URL service available today.




```
$ waypoint up

» Building...
✓ Initializing Docker client...
✓ Building image...
✓ Injecting Waypoint Entrypoint...

Tagging Docker image: waypoint.local/web:latest => hashicorp/wp-demo:latest

Docker image pushed: hashicorp/wp-demo:latest

» Deploying...
✓ Deployment successfully rolled out!

» Releasing...
✓ Service successfully configured!

The deploy was successful! A Waypoint deployment URL is shown below. This can be used internally to check
your deployment and is not meant for external traffic. You can manage this hostname using "waypoint hostname."

Release URL: http://192.168.1.79
Deployment URL: https://implicitly-new-whale--v1.alpha.waypoint.run
```

Waypoint exec

Access live environment.

Vastly improves live debugging.

Runs in same context of application.

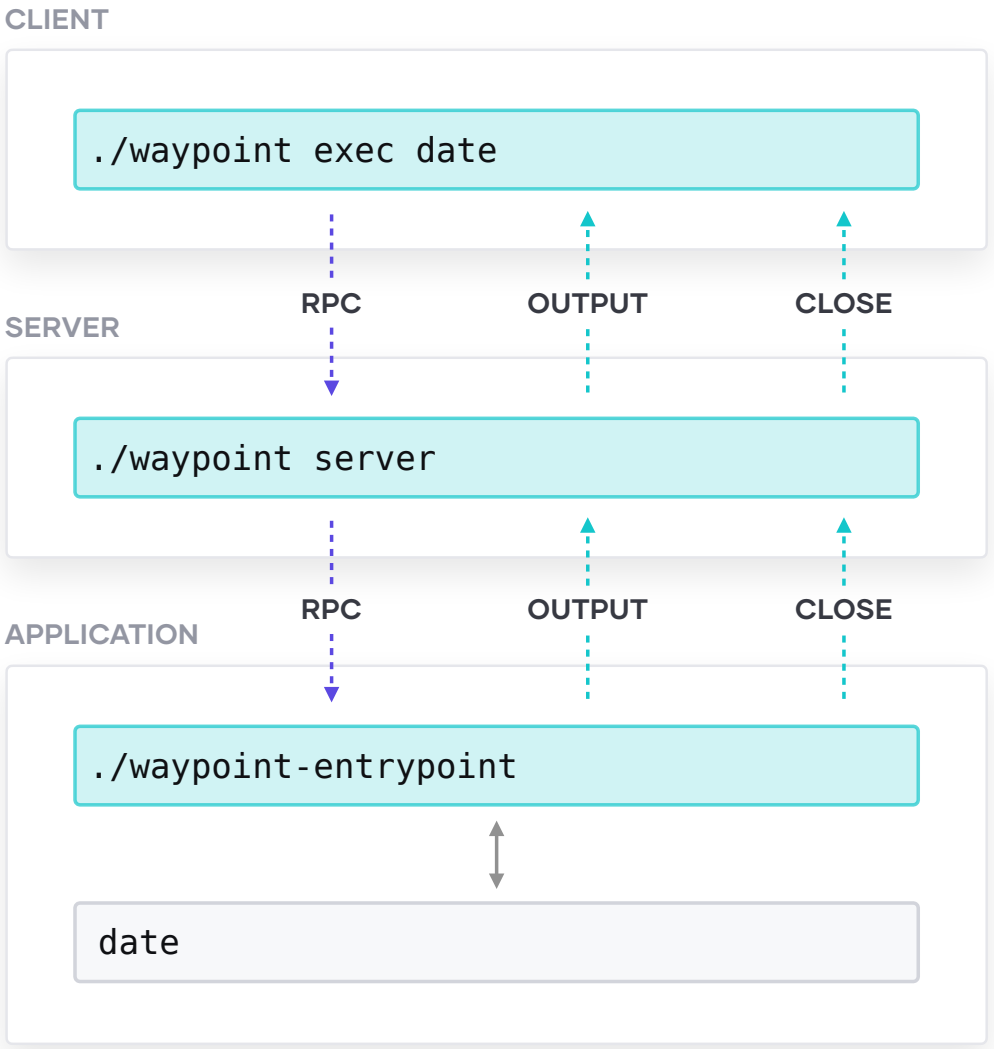
Useful for one off tasks such as database migrations.

Secured through server.

The server manages exec requests which makes them quite available and secure.

Waypoint entrypoint

Exec architecture



Waypoint entrypoint

Application configuration

Stored on server.

Application configuration served on start to provide common experience across platforms.

Per project and application.

Both scopes are available to remove the need to duplicate configuration across applications.

Provides as environment variables.

Applications need only read their environment variables to find the configuration.

Waypoint entrypoint

Logs

Stored on server.

Server stores a rolling window of the latest logs.

Developer focused.

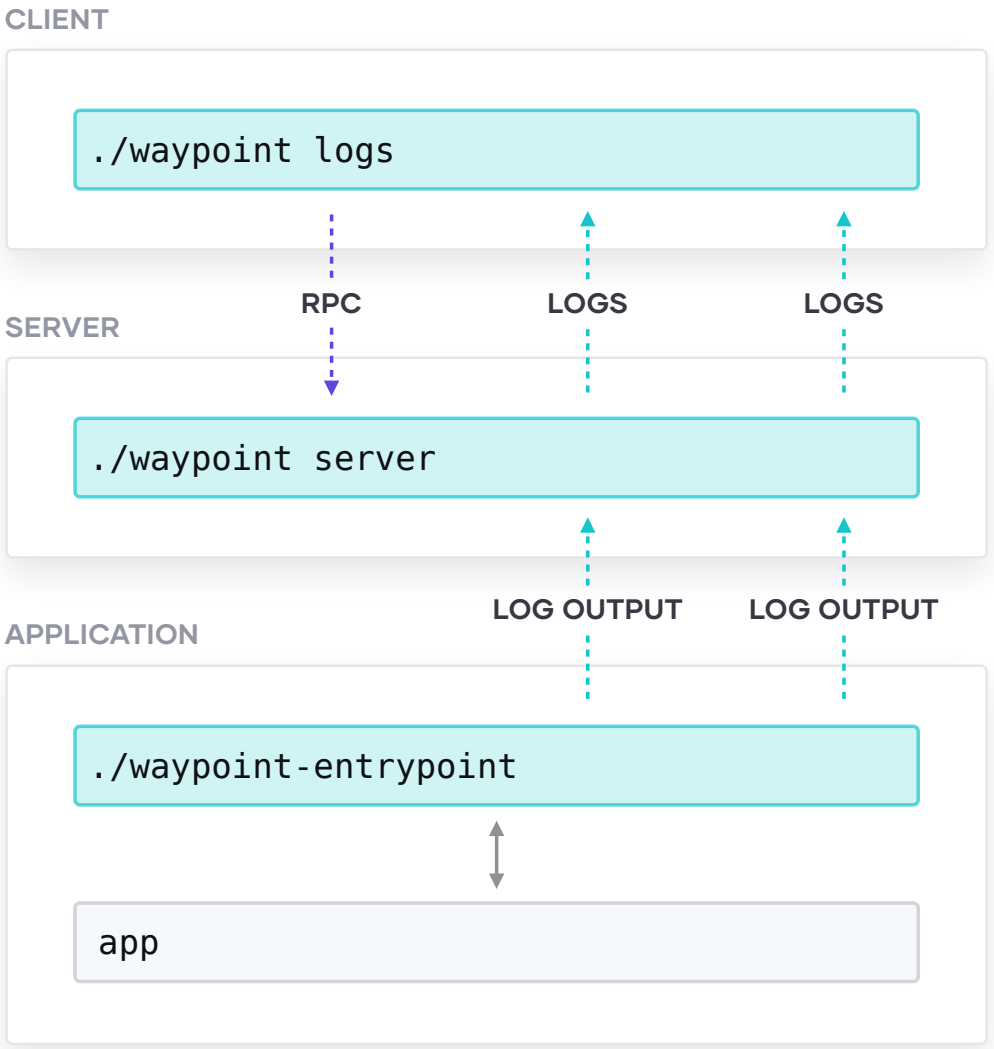
Usage optimized for observing activity of live deployments.

Compatible with other loggers.

Captures application stdio and produces it as entrypoint stdio, allowing other tools to log as well.

Waypoint entrypoint

Logs architecture



Waypoint entrypoint

Runtime functionality

Extends workflow to runtime.

Allows developers to use the same debugging and setup techniques across platforms.

Focused on developer.

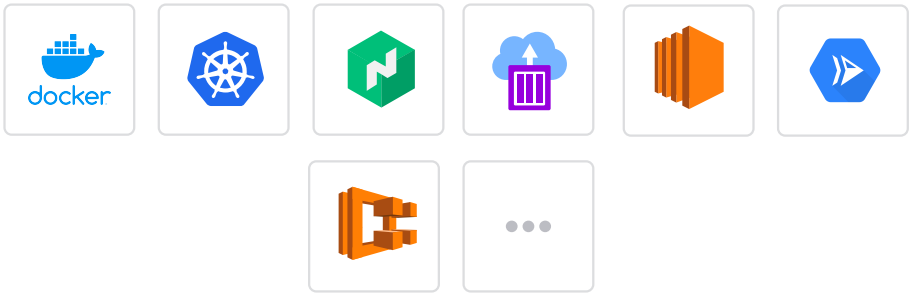
The features are focused on making the day to day experience of developers easier.

Compatible with other solutions.

The entrypoint doesn't block similar functionality by existing tools.

Plugin architecture

Plugin architecture



Integration between plugins

- Plugins communicate by outputting values that are read by later plugins.
- Values are introduced by declaration in plugin code itself.
- Framework can map between compatible values automatically.

```
# waypoint.hcl

project = “my-project”

app “wp-react” {

    build {
        use “pack” {
        }
        registry {
            use “docker” {
                image = “waypoint/demojs”
            }
        }
    }

    deploy {
        use “kubernetes” {
                probe_path=“/“
        }
    }

    release {
        use “kubernetes” {
            load_balancer=true
            port=80
        }
    }
}
```

Builders

Create an artifact.

```
build {
    use “pack” {
    }
    registry {
        use “docker” {
            image = “waypoint/demojs”
        }
    }
}
```

Access to application code.

Take the application code and transform it into an artifact to be used by later stages.

Docker images.

0.1 uses Docker images for this mostly, but the artifacts can be anything a platform plugin understands.

Examples:

Buildpacks (pack), Docker Image (docker), AMI Search (aws-ami)

Registry

Ship an artifact

(optional)

```
build {
    use “pack” {
    }
    registry {
        use “docker” {
            image = “waypoint/demojs”
        }
    }
}
```

Decoupled from build.

Being a separate plugin type allows the logic of how and where to put an artifact to be handled separately.

Captures complexity.

Many infrastructure has complexity around how and where to store artifacts.

Examples:

Docker Image (docker)

Platform

Deploy an artifact.

```
deploy {
    use “kubernetes” {
            probe_path=“/“
    }
}
```

Deploy the latest artifact.

Create whatever infrastructure is needed to run the given artifact.

Standalone Deployments.

Typically this will create deployments that do not effect existing deployments or endpoints.

Examples:

Kubernetes (kubernetes), AWS ECS (aws-ecs), Google Cloud Run (google-cloud-run)

Release

Direct traffic to a deployment (optional)

```
release {
    use “kubernetes” {
        load_balancer=true
        port=80
    }
}
```

Make a deployment accessible.

Mutate endpoint configuration to send traffic to a specific deployment.

Reversible.

By default given the latest deployment, but can redirect traffic to older, still running deployments.

Examples:

Kubernetes (kubernetes), AWS ALB (aws-alb)

Today

Builder

- Docker Image
- Buildpacks
- AWS AMI
- Files

Platform

- AWS EC2
- AWS ECS
- Azure Cloud Instances
- Docker
- Google Cloud Run
- Kubernetes
- Netlify
- Nomad

Release

- AWS ALB
- AWS ECS
- AWS AMI
- Google Cloud Run
- Kubernetes

Registry

- AWS ECR
- Docker
- Files

Plugin framework

Uses go-plugin.

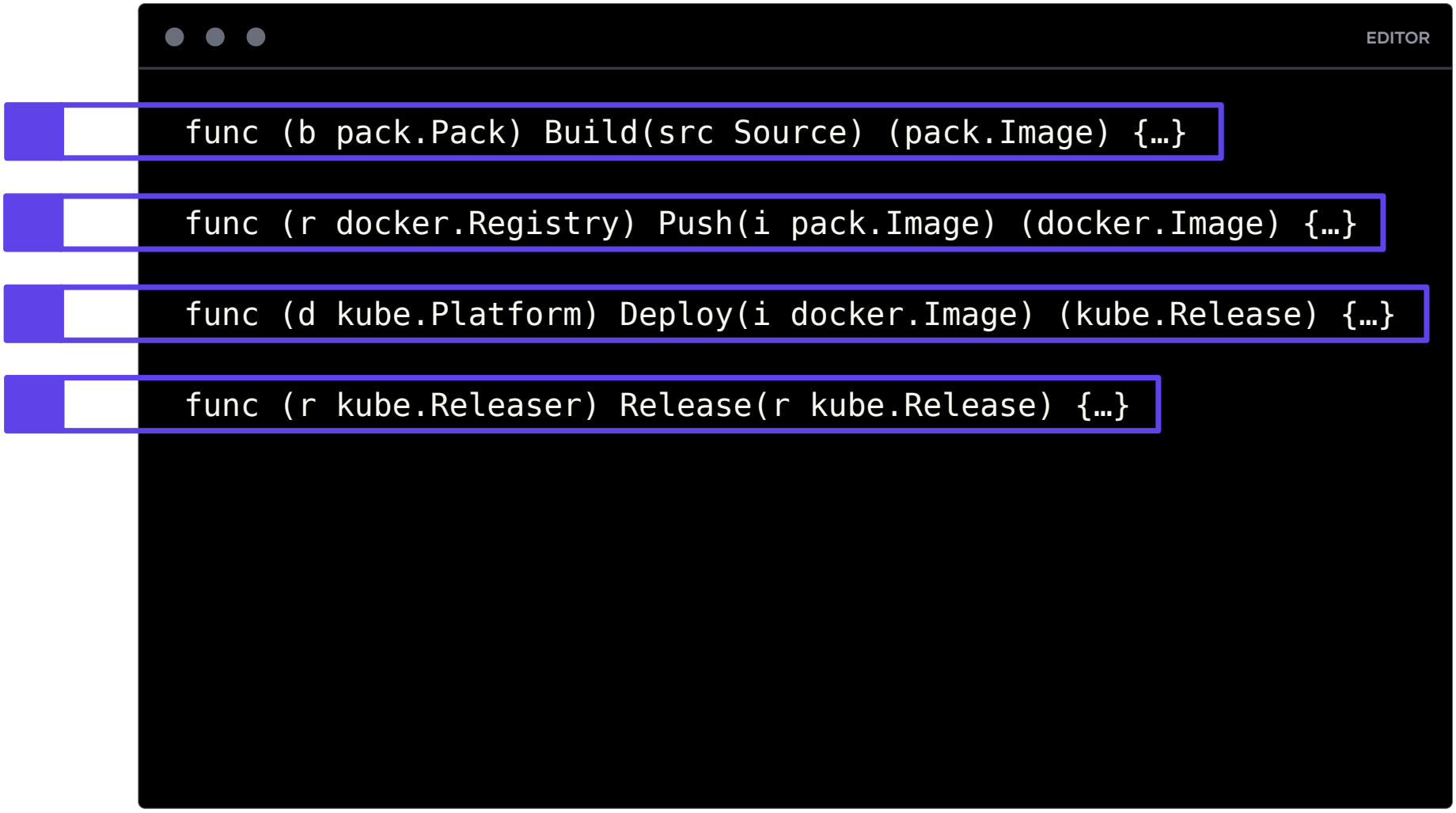
Securely allows for builtin and external plugins as processes.

SDK.

Simple Go SDK package provides all the building blocks to easily create plugins.

Rich UI.

SDK includes a set of UI components: Animated spinners, terminal output, tables, etc.


EDITOR
func (b pack.Pack) Build(src Source) (pack.Image) {…}
func (r docker.Registry) Push(i pack.Image) (docker.Image) {…}
func (d kube.Platform) Deploy(i docker.Image) (kube.Release) {…}
func (r kube.Releaser) Release(r kube.Release) {…}

Integration between plugins

- Plugins communicate by outputting values that are read by later plugins.
- Values are introduced by declaration in plugin code itself.
- Framework can map between compatible values automatically.

```
# waypoint.hcl

project = “my-project”

app “wp-react” {

    build {
        use “pack” {
        }
        registry {
            use “docker” {
                image = “waypoint/demojs”
            }
        }
    }

    deploy {
        use “kubernetes” {
            probe_path=“/“
        }
    }

    release {
        use “kubernetes” {
            load_balancer=true
            port=80
        }
    }
}
```

pack.Image

docker.Image

kube.Release

Value binding

Specific implementations.

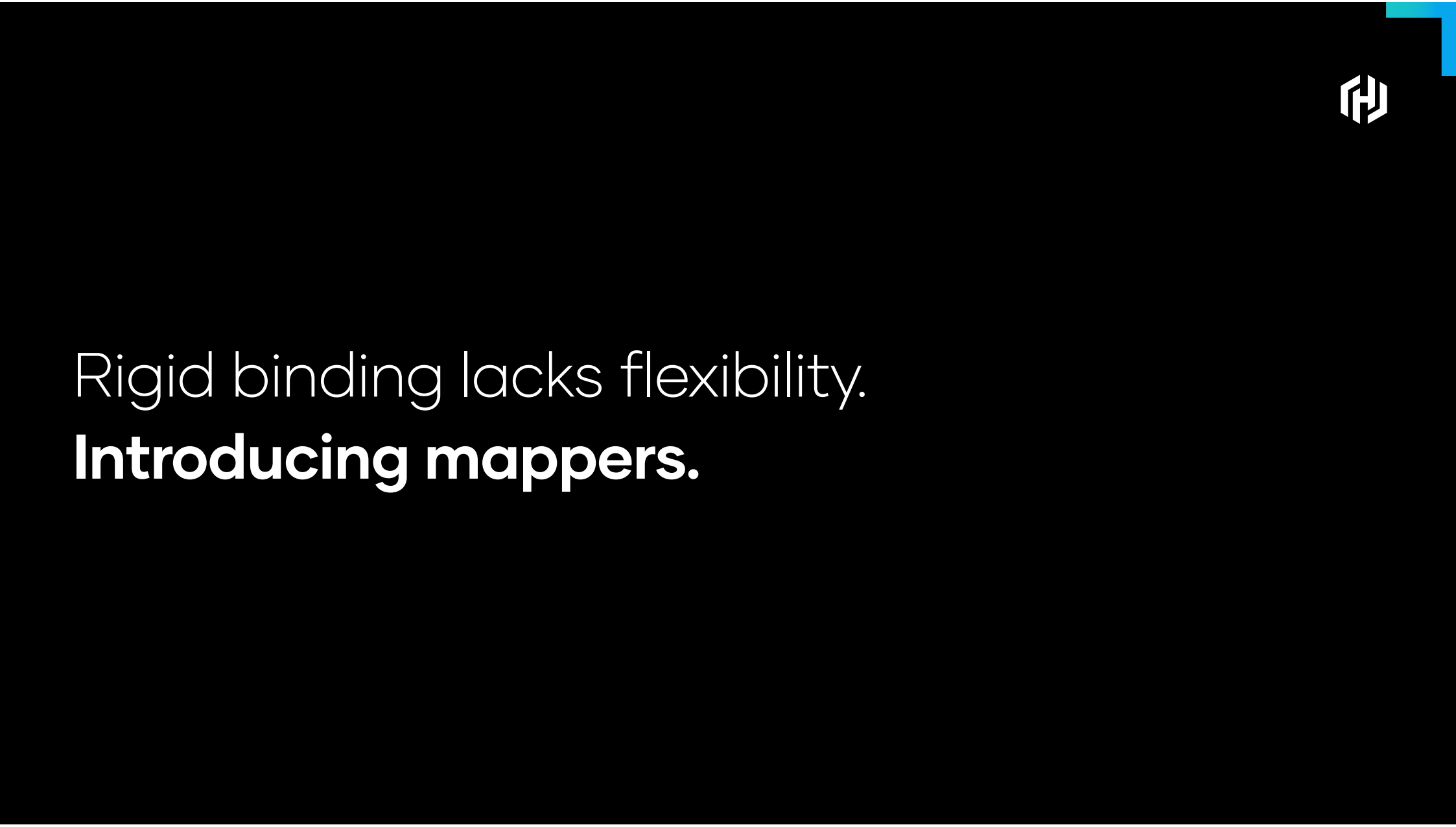
Plugins are easier because the have the data at hand, rather than dealing with generic data.

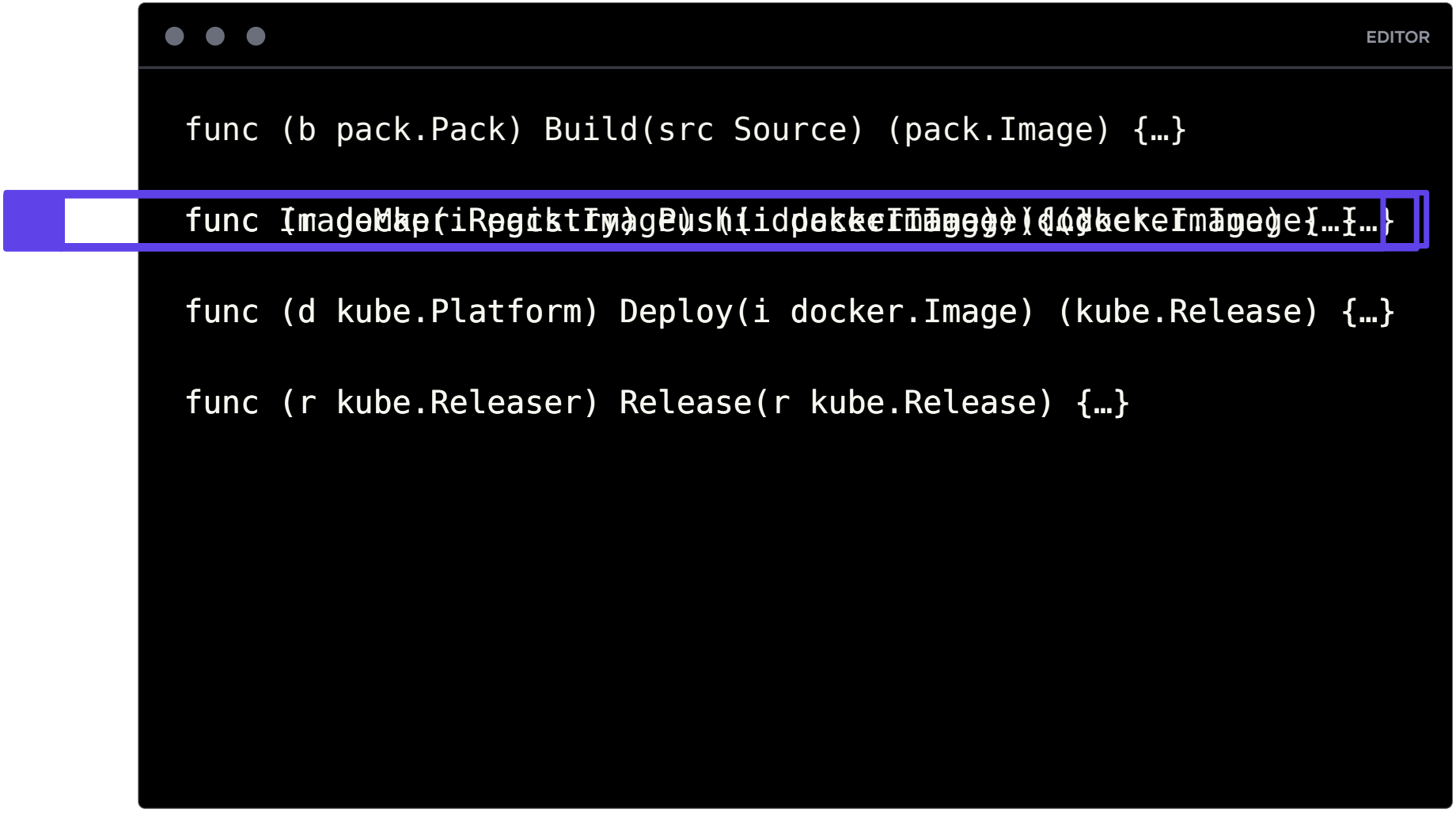
Minimal state.

Plugins don’t store data between runs, operating strictly on their inputs.

Rigid type expectations.

Plugins have to know the types to operate at all.


Rigid binding lacks flexibility.
Introducing mappers.


EDITOR
func (b pack.Pack) Build(src Source) (pack.Image) {…}
func (i docker.Image) Push(r registry.Registry) (docker.Image) {…}
func (d kube.Platform) Deploy(i docker.Image) (kube.Release) {…}
func (r kube.Releaser) Release(r kube.Release) {…}

Integration between plugins

- Plugins communicate by outputting values that are read by later plugins.
- Values are introduced by declaration in plugin code itself.
- Framework can map between compatible values automatically.

```
# waypoint.hcl

project = “my-project”

app “wp-react” {

    build {
        use “pack” {
        }
        registry {
            use “docker” {
                image = “waypoint/demojs”
            }
        }
    }

    deploy {
        use “kubernetes” {
                probe_path=“/“
        }
    }

    release {
        use “kubernetes” {
            load_balancer=true
            port=80
        }
    }
}
```

Mappers

Adds flexibility.

Mappers can figure out how to glue plugins together automatically.

Maintainability.

Mappers can live in any code package, unlocking future compatibility without changing plugins.

Extensibility.

Future work will allows mappers to be declared in config, to enhance functionality even more.

Waypoint workflow

Agility through component integration and runtime services

1. Config file provides unified description application
2. Architecture unlocks flexibility modes and services
3. Runtime services round out developer workflow
4. Easy and fast extensibility via plugin architecture

Thank you.