INERTIA

building a simple, self-hosted continuous deployment application

Robert Lin and Chad Lagore

UBC LAUNCH PAD 2018

Constantly changing cloud provider sponsors

Projects need to be redeployed to new servers

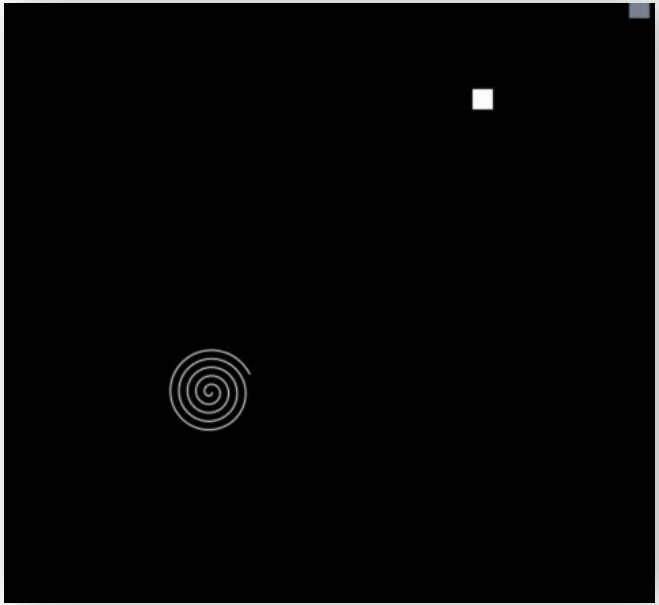
We want continuous deployment

Example Situation 1
UBC Launch Pad Project

@ubclaunchpad/bumper

Online game served from an Amazon ECS instance

Continuous deployment vital for user testing to find bugs and gameplay issues



Example Situation 1
UBC Launch Pad Project

Deployment Process

1. Cross-compile server for Amazon ECS
2. scp binary to remote VPS
3. Shut down active server and run the new binary
4. Push web application to gh-pages
5. Hope everything runs properly

Example Situation 2
Hackathon Team

Teams get free Amazon ECS instances

Bob and Co. decide to build a web service

Figuring out deployment can be time-consuming and frustrating in a hackathon scenario

Deployment would improve demo

Yes, many automated deployment solutions already exist.

- overkill, targeted at enterprise-level requirements
- not necessarily cloud agnostic (AWS CodeDeploy, Heroku, etc.)
- may require third party services (Docker Hub, etc.)
- intimidating setup process
- some even cost money - oh no!

Minimal setup
Convenient interface
Portable and platform-agnostic
Geared for small projects and teams

Enter Inertia.

Docker?

Flexible
Well supported
Works on a range of platforms
Lots of people are familiar with it

In a nutshell: the easiest way to deploy projects in unknown environments and takes away a lot of uncertainties

Compatibility is key

Command Line Application

Golang + Cobra

Simple cross-platform compilation

Serverside Daemon

Golang + Docker Client

Packaged in Docker Image

Simple setup without ever leaving your local shell

```
$> inertia init
$> inertia remote add my_vps

$> inertia my_vps init
$> inertia my_vps up
```

The Inertia Daemon

This is the agent running on your node

Only requires Docker installed server-side

It is a single image pulled from Docker Hub

Low overhead

Continuously deploys your project

Controlling Docker Containers from within a Docker Container

Current method uses a mounted Docker socket and Docker's Golang client

```
sudo docker run -d --rm \
    -p "$DAEMON_PORT":"$CONTAINER_PORT" \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v "$HOME":/app/host \
    --name inertia-daemon \
    ubclaunchpad/inertia:latest "$HOST_ADDRESS"
```

Simple to implement and works

But container with sudo access is dangerous

Building and deploying projects

Using just Docker we can build:

- Standard Dockerfile projects
- **docker-compose** projects using Docker's *docker/compose* image
- **Heroku buildpack** projects using *gliderlabs/herokuish* image

No extra dependencies to install

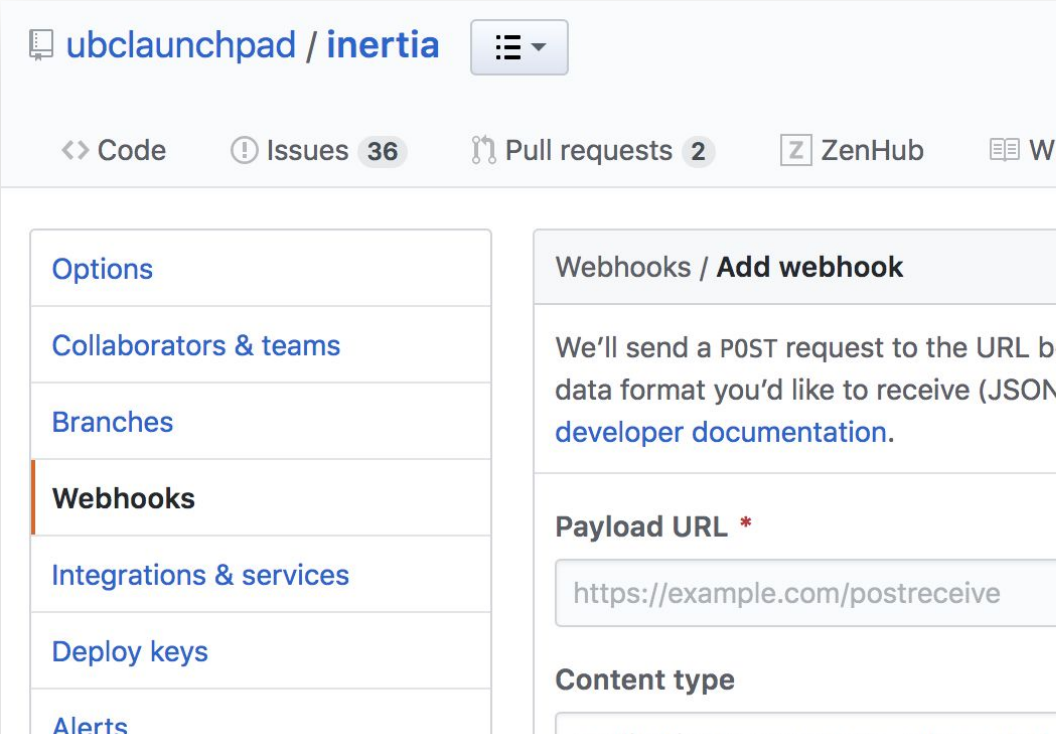
```
resp, _ := cli.ContainerCreate(
  ctx, &container.Config{
    Image:      "docker/compose,
    WorkingDir: "/build",
    Cmd:        []string{"up", "--build"},
  },
  &container.HostConfig{
    AutoRemove: true,
    Binds: []string{
        "/project:/build",
        "/var/run/docker.sock:/var/run/docker.sock",
    },
  }, nil, "docker-compose",
)

cli.ContainerStart(ctx, resp.ID, /* ... */)
```

Continuous Deployment

Daemon is a persistent server - listens for GitHub webhooks

User just has to register webhook on their repository

Daemon generates deploy key for user to grant read-only permission to repository



Deployment management

CLI can make requests over HTTPS to the daemon

Variety of commands available to start up and shut down deployment

Alter deployment configuration (such as deployed branch)

Stream logs from deployed containers

Shortcut to SSH into server

```dockerfile
# Web App build stage
FROM node:carbon AS web-build-env
ADD ./daemon/web ${BUILD_HOME}
WORKDIR ${BUILD_HOME}
RUN npm install --production
RUN npm run build

# Daemon build stage
FROM golang:alpine AS daemon-build-env
ADD . ${BUILD_HOME}
WORKDIR ${BUILD_HOME}
RUN go build -o /bin/inertia \
    ./daemon/inertia

# Final build stage
FROM alpine
WORKDIR /app
COPY --from=daemon-build-env \
    /bin/inertia \
    /usr/local/bin
COPY --from=web-build-env \
    ${BUILD_HOME}/public/ \
    /app/inertia-web
```

Inertia Web

Web application, served by the daemon, accessible from anywhere

Users can be added with varying permission levels

Deployment can be monitored through logs

All CLI functionality will eventually be accessible through the web interface

Packaged in the daemon image using multi-staged Docker build

Security

SSH for sensitive setup and HTTPS for everything else

Daemon generates a self-signed SSL certificate on startup

Webhooks verified using user-defined secret

CLI uses a JWT signed with RSA key

Looking for unprivileged way to build and start containers from daemon

Testing

Remote servers and platforms are simulated using Docker containers set up to allow SSH access

Travis CI runs integration tests against these mock servers verify that the installation script executed over SSH works as expected

```
test-integration:
    docker build -t $(VPS_OS)vps \
        -f ./test/env/Dockerfile.$(VPS_OS) \
        --build-arg VERSION=$(VPS_VERSION) \
        ./test
    docker run --rm -d \
        --name testvps \
        --privileged \
        $(VPS_OS)vps
    make testdaemon
    go test ./... -run 'Integration' --cover
```

Inertia

- Reduced time spent by teams deploying and redeploying
- Strong cross-platform compatibility with Docker and Golang
- Many handy features implemented, with room for more

https://github.com/ubclaunchpad/inertia