

GOLANG AS THE DREAM OF A DEVOPS ENGINEER

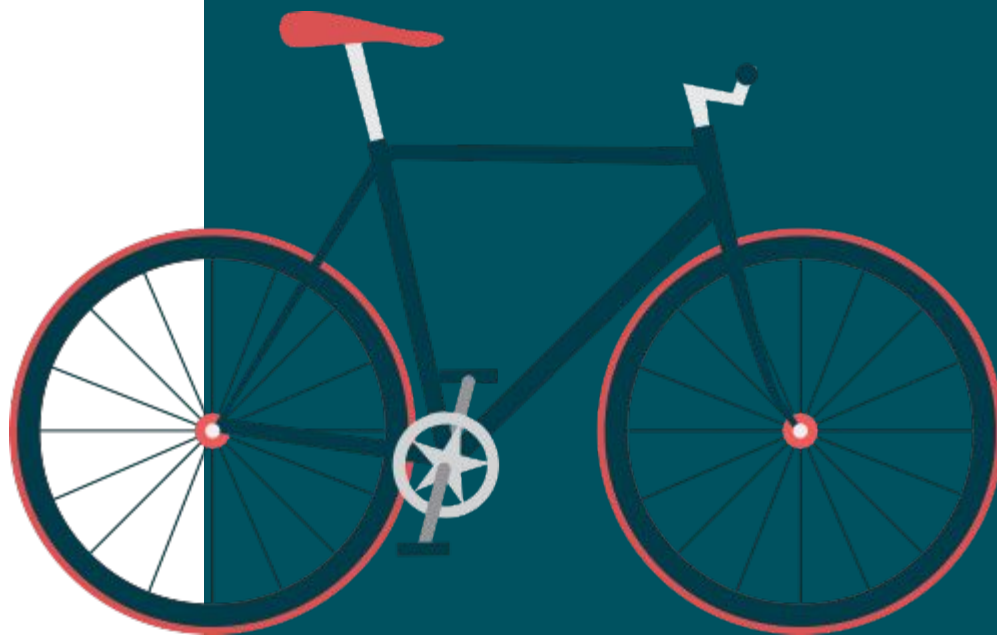




MÁTÉ GULYÁS

CTO

 [@gulyasm](https://twitter.com/gulyasm)



ENBRITELY LOVES SERVICES

ENBRITELY LOVES MICROSERVICES

MICROSERVICES LOVES GO

ENBRITELY LOVES GO





GO IS GREAT FOR DEPLOYMENT

GITHUB



PACKER



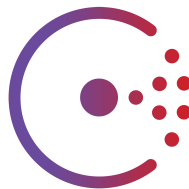
ANSIBLE



AMI



CONSUL



TERRAFORM



AMI-ID

STICK WITH THE STANDARD LIBRARY

**MIDDLEWARE?
I WANT MY MIDDLEWARES!**

LOGGING



**LOG ONLY WHAT'S
IMPORTANT**

ERROR MESSAGE DETAILS!

SET LOG PREFIX

GO TOOLS



TOTALLY AWESOME!

GO TOOLS

go fmt

go vet

go test

go get

golint

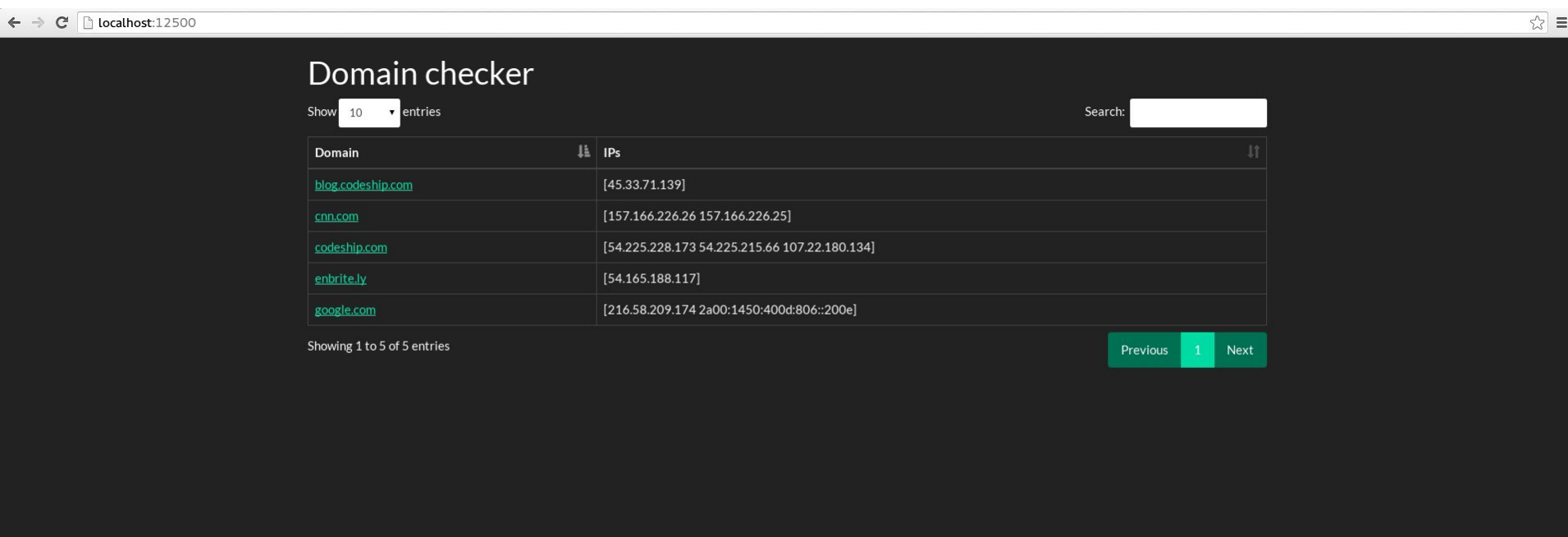
A FEW QUESTION...

1. is my service **RUNNING**?
2. what **VERSION** is running?
3. what is the **UPTIME**?

```
go RunHeartbeatService(":10101")
```

<https://github.com/enbrite/heartbeat-golang>

ALWAYS WRITE A UI



ALWAYS CREATE A UI

**YOUR TEAM WILL
THANK YOU!**

CONFIGURATION





THE TWELVE-FACTOR APP

INTRODUCTION

In the modern era, software is commonly delivered as a service: called *web apps*, or *software-as-a-service*. The twelve-factor app is a methodology for building software-as-a-service apps that:

- Use **declarative** formats for setup automation, to minimize time and cost for new developers joining the project;
- Have a **clean contract** with the underlying operating system, offering **maximum portability** between execution environments;
- Are suitable for **deployment** on modern **cloud platforms**, obviating the need for servers and systems administration;
- **Minimize divergence** between development and production, enabling **continuous deployment** for maximum agility;
- And can **scale up** without significant changes to tooling, architecture, or development practices.

The twelve-factor methodology can be applied to apps written in any programming language, and which use any combination of backing services (database, queue, memory cache, etc).

BACKGROUND

The contributors to this document have been directly involved in the development and deployment of hundreds of apps, and indirectly witnessed the development, operation, and scaling of hundreds of thousands of apps via our work on the Heroku platform.

This document synthesizes all of our experience and observations on a wide variety of software-as-a-service apps in the wild. It is a triangulation on ideal practices for app development, paying particular attention to the dynamics of the organic growth of an app over time, the dynamics of collaboration between developers working on the app's codebase, and avoiding the cost of software erosion.

Our motivation is to raise awareness of some systemic problems we've seen in modern application development, to provide a shared vocabulary for discussing those problems, and to offer a set of broad conceptual solutions to those problems with accompanying terminology. The format is inspired by Martin Fowler's books *Patterns of Enterprise Application Architecture* and *Refactoring*.

WHO SHOULD READ THIS DOCUMENT?

Any developer building applications which run as a service. Ops engineers who deploy or manage such applications.



THE TWELVE-FACTOR APP

III. Config

Store config in the environment

An app's *config* is everything that is likely to vary between deploys (staging, production, developer environments, etc). This includes:

- Resource handles to the database, Memcached, and other backing services
- Credentials to external services such as Amazon S3 or Twitter
- Per-deploy values such as the canonical hostname for the deploy

Apps sometimes store config as constants in the code. This is a violation of twelve-factor, which requires **strict separation of config from code**. Config varies substantially across deploys, code does not.

A litmus test for whether an app has all config correctly factored out of the code is whether the codebase could be made open source at any moment, without compromising any credentials.

Note that this definition of “config” does **not** include internal application config, such as `config/routes.rb` in Rails, or how code modules are connected in Spring. This type of config does not vary between deploys, and so is best done in the code.

Another approach to config is the use of config files which are not checked into revision control, such as `config/database.yml` in Rails. This is a huge improvement over using constants which are checked into the code repo, but still has weaknesses: it's easy to mistakenly check in a config file to the repo; there is a tendency for config files to be scattered about in different places and different formats, making it hard to see and manage all the config in one place. Further, these formats tend to be language- or framework-specific.

The twelve-factor app stores config in *environment variables* (often shortened to *env vars* or *env*). Env vars are easy to change between deploys without changing any code; unlike config files, there is little chance of them being checked into the code repo accidentally; and unlike custom config files, or other config mechanisms such as Java System Properties, they are a language- and OS-agnostic standard.

Another aspect of config management is grouping. Sometimes apps batch config into named groups (often called “environments”) named after specific deploys, such as the `development`, `test`, and `production` environments in Rails. This method does not scale cleanly: as more deploys of the app are created, new environment names are necessary, such as `staging` or `qa`. As the project grows further, developers may add their own special environments like `joes-staging`, resulting in a combinatorial explosion of config which makes managing deploys of the

.dotenv file

```
export DB_USER=root  
export DB_PASSWORD=root  
export DB_HOST=localhost:3306  
export DB_DATABASE=TestDB  
export ADDRESS=:12500  
export CS_ADDRESS=http://localhost:12300/api  
export HEARTBEAT_ADDRESS=:12501
```



SUMMARY

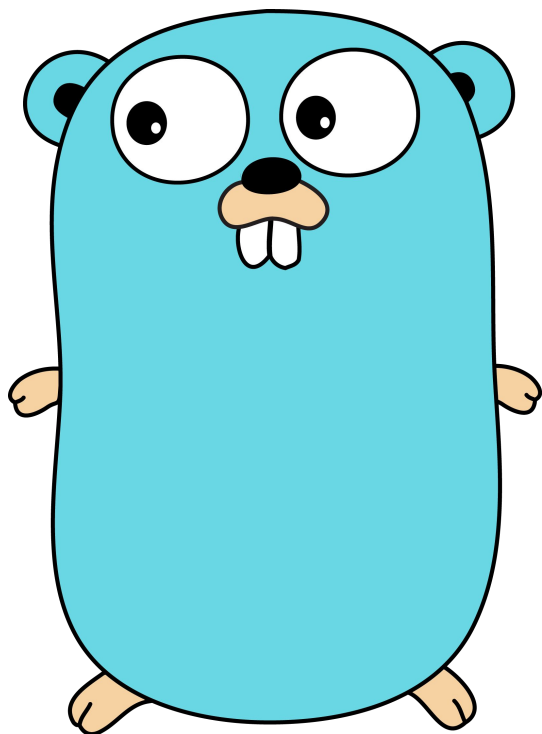
Uniform build, deployment

Uniform Heartbeat signal

Always provide UI

Use env variables for configuration

Use the  tools



WE ARE HIRING!

THANK YOU!

MATE GULYAS

CTO

gulyasm@enbrite.ly



| *@gulyasm*
@enbritely

