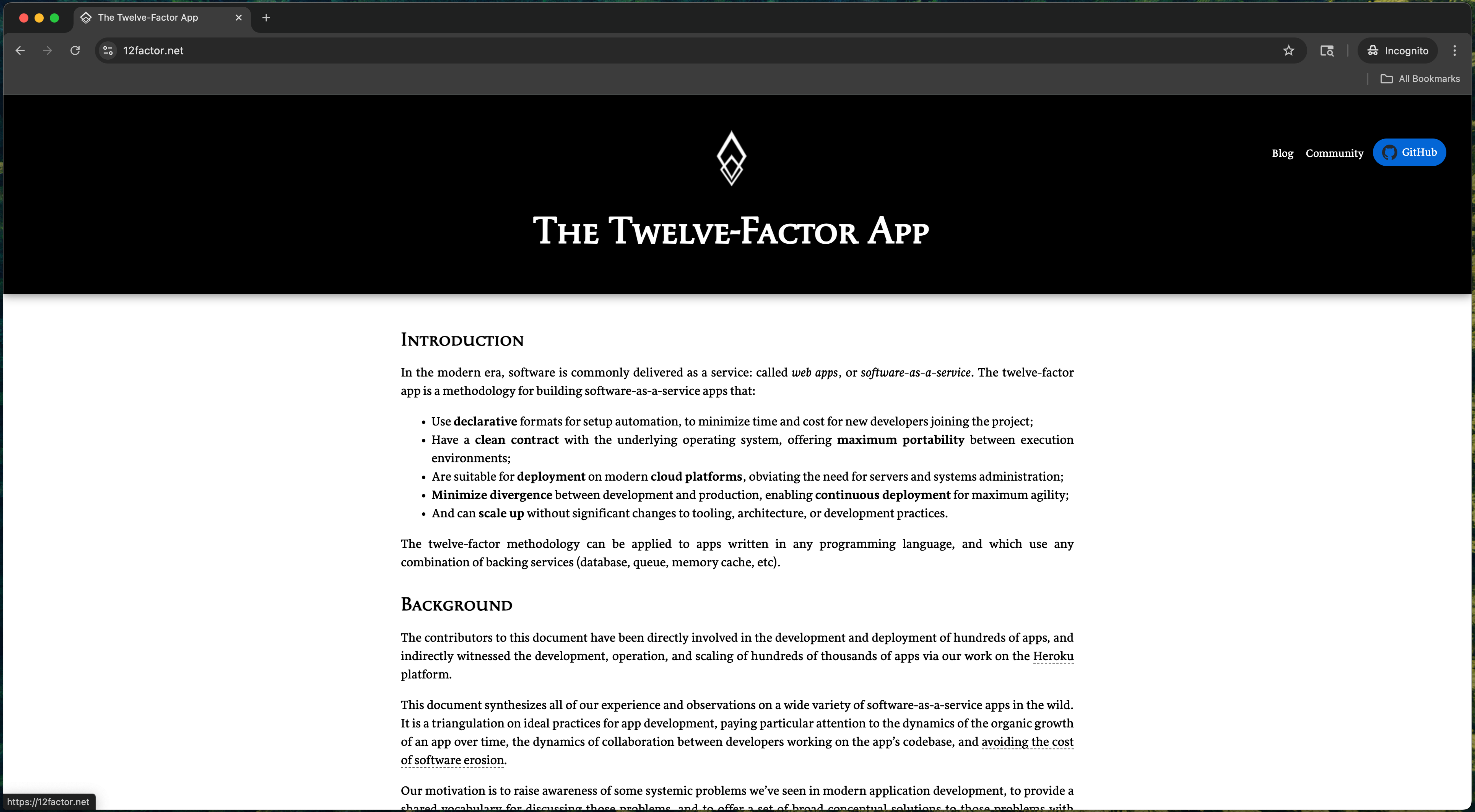
The Twelve Factor App Methodology

https://12factor.net

Jake Worth

Agenda

- What is the Twelve Factor App Methodology?
- Why does it matter?
- The Twelve Factors
- Questions



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

What is the Twelve Factor App Methodology?

- The twelve-factor app is a methodology for building software-as-a-service
- Drafted by developers at Heroku in 2011
- Best practices for app setup, portability, deployment, and scalability

Why Does it Matter?

» Professionally relevant to all software and devops engineers

» Part of our language and conventions

» I want our team to follow it

» And get us thinking ⚙️

The Twelve Factors

I. Codebase
II. Dependencies
III. Config
IV. Backing services
V. Build, release, run
VI. Processes
VII. Port binding
VIII. Concurrency
IX. Disposability
X. Dev/prod parity
XI. Logs
XII. Admin processes

N. Factor Name

"Book definition"

- Practical/modern translation

I. Codebase

"One codebase tracked in revision control, many deploys"

» Distributed version control system (DVCS)

» One repository per app ("Banking" -> `org/banking.git`)

» Multiple deploys (prod, staging, local)

II. Dependencies

"Explicitly declare and isolate dependencies"

- All dependencies explicitly stated in a manifest (`package.json`)
- Dependencies installed in isolation (`npm install`)
- No implicit dependencies (`cURL`, `imagemagick`)

III. Config

"Store config in the environment"

- Strict separation of config from code via env vars
- Configuration can be changed without a deployment
- ⭐ Repo could made OSS at any time without compromising credentials

IV. Backing services

"Treat backing services as attached resources"

» Resources can be attached and detached at will

» Could swap MySQL database with one managed by a third party such as Amazon RDS without any changes to code

V. Build, release, run

"Strictly separate build and run stages"

» Strict separation between the build, release, and run stages

» Changes only "go forward" and can't be made at runtime

» Ability to roll back but prefer to roll forward

VI. Processes

"Execute the app as one or more stateless processes"

- Processes are stateless and share-nothing
- Anything in memory is considered temporary and unreliable
- Any data that needs to persist must be stored in a database

VII. Port binding

"Export services via port binding"

» App is completely self-contained and does not rely on runtime webserver into the execution environment to create a web-facing service

» The web app exports HTTP as a service by binding to a port, and listening to requests coming in on that port

» Exposed in dev via `localhost:3000`

VIII. Concurrency

"Scale out via the process model"

- Processes are a first class citizen, supporting all operations available to other entities
- Processes share nothing with each other
- Scaling is built in: more work, more processes 📈

IX. Disposability

"Maximize robustness with fast startup and graceful shutdown"

» Processes are disposable, meaning they can be started or stopped at a moment's notice

» Minimal startup time

» Graceful error recovery and transactional behavior

» Graceful shutdown

X. Dev/prod parity

"Keep development, staging, and production as similar as possible"

» Developer can write code and deploy it prod in hours or minutes. Can be a day-one onboarding step! ⭐

» Developers are closely involved in deploying and monitoring production

» Development, staging, and production are as similar as possible

XI. Logs

"Treat logs as event streams"

- App never concerns itself with routing or storage of its output stream
- Does not attempt to write to or manage logfiles
- Each running process writes its event stream, unbuffered
- In dev, developers view this stream in the foreground of their terminal

XII. Admin processes

"Run admin/management tasks as one-off processes"

» Tasks like database migrations are one-off scripts (`manage.py migrate`), with dependency isolation

» Languages with REPL are preferred

Questions

- How are we succeeding?
- How could be get better?
- What is your role in our implementation?

Thank you