Kostenloses eBook

LERNEN

heroku

Free unaffiliated eBook created from
Stack Overflow contributors.

#heroku

Inhaltsverzeichnis

Über ... 1

Kapitel 1: Erste Schritte mit Heroku ... 2

Bemerkungen ... 2

Examples ... 2

Installation oder Setup ... 2

Herunterladen ... 2

Homebrew ... 2

Debian / Ubuntu ... 2

Verwenden des Heroku Toolbelt ... 3

Erstellen Sie eine Anwendung ... 3

Bereitstellen bei Heroku ... 3

Öffnen Sie Ihre Anwendung in einem Browser ... 3

Listen Sie Heroku-Befehle auf ... 3

Allgemeine Hilfe ... 3

Hilfe für einen bestimmten Befehl ... 3

Heroku-Anwendungen erstellen ... 3

Kapitel 2: Abhängigkeiten ... 5

Syntax ... 5

Examples ... 5

Bower-Abhängigkeit ... 5

Kapitel 3: Befehlszeile ... 6

Einführung ... 6

Syntax ... 6

Examples ... 6

Herunterladen und installieren ... 6

OS X ... 6

Windows ... 6

Debian / Ubuntu ... 6

Standalone-Version ... 6

Überprüfen Sie Ihre Installation ... 7
Fertig machen ... 7
Kapitel 4: Buildpack ... 8
Examples ... 8
Buildpacks einstellen ... 8
Mehrere Buildpacks ... 8
Kapitel 5: Einsatz ... 10
Syntax ... 10
Examples ... 10
Bereitstellen mit Git ... 10
Tracking Ihrer App in git ... 10
Heroku-Fernbedienung erstellen ... 10
Bereitstellen von Code ... 10
Kapitel 6: Heroku Add-Ons ... 12
Einführung ... 12
Examples ... 12
Heroku Scheduler ... 12
Kapitel 7: Heroku node.js Hallo Welt ... 13
Bemerkungen ... 13
Examples ... 13
Heroku node.js Hallo Welt ... 13
Kapitel 8: Heroku Postgres ... 15
Examples ... 15
Zurücksetzen der Postgres-Datenbank in Heroku ... 15
So kopieren Sie die Heroku-Datenbank in die lokale Datenbank ... 15
Kapitel 9: Heroku-Fehlercodes ... 16
Einführung ... 16
Syntax ... 16
Examples ... 17
H10 - App ist abgestürzt ... 17
H11 - Rückstand zu groß ... 17

H12 - Anforderungs-Timeout ... 17
H13 - Verbindung wurde ohne Antwort geschlossen ... 18
H14 - Keine Webdynos laufen ... 18
H15 - Leerlaufverbindung ... 18
Kapitel 10: Heroku-Grenzen ... 20
Examples ... 20
Liste aller Einschränkungen der Heroku-Plattform ... 20
Kapitel 11: Pipelines ... 21
Syntax ... 21
Bemerkungen ... 21
Examples ... 21
Pipelines über die CLI ... 21
Kapitel 12: Protokolle ... 23
Syntax ... 23
Examples ... 23
Arten von Protokollen ... 23
Protokollformat ... 23
Protokolle anzeigen ... 24
Echtzeit-Schwanz ... 24
Protokollfilterung ... 24
Credits ... 26

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: heroku

It is an unofficial and free heroku ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official heroku.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit Heroku

Bemerkungen

Heroku ist ein beliebter Platform-as-a-Service-Provider (PaaS), der es Entwicklern erleichtert, Webanwendungen ohne ein Operationsteam bereitzustellen. Heroku gibt es seit 2007 und ist jetzt im Besitz von Salesforce .

Dieser Abschnitt bietet einen Überblick über Heroku und warum ein Entwickler es möglicherweise verwenden möchte.

Es sollte auch große Themen in Heroku erwähnen und auf die verwandten Themen verweisen. Da die Dokumentation für Heroku neu ist, müssen Sie möglicherweise erste Versionen dieser verwandten Themen erstellen.

Examples

Installation oder Setup

Um Heroku-Apps lokal zu erstellen und zu verwalten, benötigen Sie den Heroku-Toolbelt.

Herunterladen

Laden Sie das Heroku Toolbelt- Installationsprogramm von der Heroku-Website herunter.

Homebrew

Installieren Sie `heroku` mit `brew` :

```
brew install heroku
```

Debian / Ubuntu

Führen Sie dieses Skript aus:

```
wget -O- https://toolbelt.heroku.com/install-ubuntu.sh | sh
```

Dieses Skript fügt apt das Heroku-Repository hinzu, installiert den Heroku-Freigabeschlüssel, installiert den Heroku-Toolbelt und installiert dann Ruby, wenn Sie es benötigen.

Wie bei jedem Skript, das Sie online finden und direkt an bash weiterleiten, empfehlen wir Ihnen

dringend, zuerst die Quelle zu lesen.

Verwenden des Heroku Toolbelt

Erstellen Sie eine Anwendung

```
heroku create your-app-name
```

Bereitstellen bei Heroku

```
git push heroku master
```

Öffnen Sie Ihre Anwendung in einem Browser

```
heroku open your-app-name
```

Listen Sie Heroku-Befehle auf

```
heroku commands
```

Allgemeine Hilfe

```
heroku help
```

Hilfe für einen bestimmten Befehl

```
heroku help <command>
```

Heroku-Anwendungen erstellen

Mit dem Befehl `heroku create` können Sie eine Heroku-Anwendung erstellen. Jede Anwendung, die Sie in Heroku bereitstellen, verfügt über eine eigene Codebasis, Umgebungsvariablen, Addons usw.

Jede Heroku-Anwendung hat einen weltweit eindeutigen Namen. Wenn Sie versuchen, eine Heroku-Anwendung zu erstellen, deren Name bereits vergeben ist, wird eine Fehlermeldung angezeigt.

So erstellen Sie eine neue Heroku-Anwendung:

```
heroku create [app_name]
```

Wenn Sie beim Ausführen von `heroku create` keinen Anwendungsnamen angeben, erstellt Heroku einen zufälligen Anwendungsnamen.

Sie können auch die Amazon-Region angeben, in der Ihre Heroku-Anwendung erstellt werden soll. Standardmäßig werden alle Heroku Anwendungen sind in der erstellten `us` Region. Wenn Sie die Region ändern möchten, erstellen Sie die Anwendung wie folgt:

```
heroku create [app_name] --region eu
```

Im Moment gibt es nur zwei öffentliche Regionen: `us` und die `eu` (Europa).

Erste Schritte mit Heroku online lesen: https://riptutorial.com/de/heroku/topic/959/erste-schritte-mit-heroku

Kapitel 2: Abhängigkeiten

Syntax

- "Abhängigkeiten": {...}

Examples

Bower-Abhängigkeit

Um Bower und seine Komponenten automatisch zu installieren, muss man dies tun

1. Geben Sie die `package.json` Abhängigkeit in `package.json` :

```
"dependencies": {
    "bower": "^1.7.9"
}
```

2. Verwenden Sie `scripts` , um einen `postinstall` Befehl auszuführen

```
"scripts": {
    "postinstall": "./node_modules/bower/bin/bower install"
}
```

3. Erstellen Sie eine `.bowerrc` Datei, um das Verzeichnis für die Installation von bower_components festzulegen. Andernfalls werden bower_components im Stammverzeichnis installiert.

```
{
    "directory" : "app/bower_components"
}
```

Nun Heroku automatisch ausführt `bower install` Befehl nach `npm install`

Abhängigkeiten online lesen: https://riptutorial.com/de/heroku/topic/6665/abhangigkeiten

Kapitel 3: Befehlszeile

Einführung

Das Heroku Command Line Interface (CLI), früher als Heroku Toolbelt bekannt, ist ein Tool zum Erstellen und Verwalten von Heroku-Apps über die Befehlszeile / Shell verschiedener Betriebssysteme.

Syntax

- $ heroku --version
- $ heroku login
- $ heroku erstellen

Examples

Herunterladen und installieren

OS X

Laden Sie das OS X-Installationsprogramm herunter und führen Sie es aus.

Windows

Laden Sie das Windows-Installationsprogramm 32-Bit 64-Bit herunter und führen Sie es aus.

Debian / Ubuntu

Führen Sie die folgenden Schritte aus, um unser apt-Repository hinzuzufügen und die CLI zu installieren:

```
$ sudo add-apt-repository "deb https://cli-assets.heroku.com/branches/stable/apt ./"
$ curl -L https://cli-assets.heroku.com/apt/release.key | sudo apt-key add -
$ sudo apt-get update
$ sudo apt-get install heroku
```

Standalone-Version

Laden Sie das Tarball herunter und extrahieren Sie es, damit Sie von Ihrem PFAD auf die Binärdatei zugreifen können. Zum Beispiel:

```
$ echo replace OS/ARCH with values as noted below
$ wget https://cli-assets.heroku.com/branches/stable/heroku-OS-ARCH.tar.gz
$ tar -xvzf heroku-OS-ARCH /usr/local/lib/heroku
$ ln -s /usr/local/lib/heroku/bin/heroku /usr/local/bin/heroku
```

Überprüfen Sie Ihre Installation

Um Ihre CLI-Installation zu überprüfen, verwenden Sie den Befehl `heroku --version` .

```
$ heroku --version
heroku-cli/5.6.0-010a227 (darwin-amd64) go1.7.4
```

Fertig machen

Sie werden aufgefordert, Ihre Heroku-Zugangsdaten einzugeben, wenn Sie zum ersten Mal einen Befehl ausführen. Nach dem ersten Mal werden Ihre E-Mail-Adresse und ein API-Token zur späteren Verwendung in `~/.netrc` gespeichert.

Im Allgemeinen ist es eine gute Idee, sich sofort nach der Installation der Heroku-CLI anzumelden und Ihren öffentlichen Schlüssel hinzuzufügen, sodass Sie mit git Heroku-App-Repositorys pushen oder clonen können:

```
$ heroku login
Enter your Heroku credentials.
Email: adam@example.com
Password (typing will be hidden):
Authentication successful.
```

Sie können jetzt Ihre erste Heroku-App erstellen:

```
$ cd ~/myapp
$ heroku create
Creating app... done,  sleepy-meadow-81798
https://sleepy-meadow-81798.herokuapp.com/ | https://git.heroku.com/sleepy-meadow-81798.git
```

Befehlszeile online lesen: https://riptutorial.com/de/heroku/topic/8324/befehlszeile

Kapitel 4: Buildpack

Examples

Buildpacks einstellen

Heroku unterstützt offiziell Buildpacks für Ruby, Node.js, Clojure, Python, Java, Gradle, Grails, Scala, Play, PHP und Go.

Buildpacks werden von Heroku automatisch in der obigen Reihenfolge erkannt. Sie können sie jedoch auch manuell über die CLI einstellen:

1. Zum Zeitpunkt der App-Erstellung

```
heroku create <app_name> --buildpack <buildpack_name>
```

2. Manuell,

```
heroku buildpacks:set <buildpack_name>
```

Der Buildpack-Name kann entweder über die Kurzschreibweise oder über die URL angegeben werden. Wie für PHP Buildpack,

```
heroku buildpacks:set heroku/php
```

oder

```
heroku buildpacks:set https://elements.heroku.com/buildpacks/heroku/heroku-buildpack-php
```

Mehrere Buildpacks

Eine Anwendung kann auch mehrere Buildpacks enthalten. Dies kann mit `add` erreicht werden:

```
heroku buildpacks:add --index 1 <buildpack_name>
```

Dabei gibt der Parameter `--index` die Ausführungsreihenfolge von Buildpack an.

Sagen,

```
heroku buildpacks:set heroku/php
heroku buildpacks:add --index 1 heroku/nodejs
```

setzt die Buildpack-Reihenfolge als:

```
heroku/nodejs
heroku/php
```

Denken Sie daran: Eine Heroku-App hat nur einen öffentlichen Port - 80. Daher wird einer der beiden Ports in einem Port dienen. Wenn beispielsweise `procfile` mit `web: node server.js` , wird die Knotenanwendung in Port 80 ausgeführt, ansonsten in PHP. Der Build wird jedoch in der angegebenen Reihenfolge ausgeführt. Wenn mehrere Anwendungen benötigt werden, richten Sie mehrere Projekte ein und stellen Sie die Kommunikation untereinander her.

Buildpack online lesen: https://riptutorial.com/de/heroku/topic/6126/buildpack

Kapitel 5: Einsatz

Syntax

- git Push Heroku-Meister

Examples

Bereitstellen mit Git

Tracking Ihrer App in git

Bevor Sie eine App auf Heroku pushen können, müssen Sie ein lokales Git-Repository initialisieren und Ihre Dateien in das Commit übernehmen. Wenn sich zum Beispiel eine App in einem Verzeichnis befindet, myapp, erstellen Sie ein neues Repository dafür:

```
$ cd myapp
$ git init
Initialized empty Git repository in .git/
$ git add .
$ git commit -m "my first commit"
Created initial commit 5df2d09: my first commit
 44 files changed, 8393 insertions(+), 0 deletions(-)
 create mode 100644 README
 create mode 100644 Procfile
 create mode 100644 app/controllers/source_file
...
```

Dies ist ein lokales Repository, das sich jetzt im `.git` Verzeichnis befindet. Es wurde bisher noch nichts gesendet. Sie müssen eine Fernbedienung erstellen und einen Push durchführen, um Ihren Code in Heroku bereitzustellen.

Heroku-Fernbedienung erstellen

```
$ heroku create
Creating falling-wind-1624... done, stack is cedar-14
http://falling-wind-1624.herokuapp.com/ | https://git.heroku.com/falling-wind-1624.git
Git remote heroku added
```

Git-Repository mit einer vorhandenen Anwendung. Der Befehl heroku git: remote wird diese Fernbedienung für Sie hinzufügen, basierend auf der git-URL Ihrer Anwendungen.

```
$ heroku git:remote -a falling-wind-1624
Git remote heroku added.
```

Bereitstellen von Code

Sie müssen einen Remote-Zweig angeben, zu dem ein Push ausgeführt werden soll. Sie können Ihren ersten Anstoß machen:

```
$ git push heroku master
Initializing repository, done.
updating 'refs/heads/master'
...
```

Verwenden Sie diese Syntax, um einen anderen Zweig als master zu pushen:

```
$ git push heroku yourbranch:master
```

Einsatz online lesen: https://riptutorial.com/de/heroku/topic/8325/einsatz

Kapitel 6: Heroku Add-Ons

Einführung

Details und Anweisungen zu verschiedenen Add-Ons, die bei Heroku verfügbar sind.

Examples

Heroku Scheduler

Heroku Scheduler installieren

```
heroku addons:create scheduler:standard
```

Heroku Add-Ons online lesen: https://riptutorial.com/de/heroku/topic/8906/heroku-add-ons

Kapitel 7: Heroku node.js Hallo Welt

Bemerkungen

Anmeldung

`heroku login`

App erstellen

`heroku create` oder `heroku create your_name`

klonen Sie das Beispiel

```
git clone https://github.com/zoutepopcorn/herokuworld
cd herokuworld
```

Besuchen Sie die App in Ihrem Browser

```
https://your_name.herokuapp.com/
```

Optional testen Sie es lokal:

```
heroku local web
```

überprüfen: lolhost: 5000

Was unterscheidet sich also von einer normalen node.js-App? package.json

```
"scripts": {
  "start": "node index.js"
},
"engines": {
  "node": "7.6.0"
}
```

index.js

```
process.env.PORT
```

Lokaler Port: 5000. Heroku ordnet den Port 80 Ihrer App-URL zu.

Examples

Heroku node.js Hallo Welt

index.js

```
var http = require("http");

http.createServer(function(request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Heroku world!");
  response.end();
}).listen(process.env.PORT);
```

package.json

```
{
  "name": "node-example",
  "version": "1.0.0",
  "description": "Hello world Heroku",
  "scripts": {
    "start": "node index.js"
  },

  "keywords": [
    "example",
    "heroku"
  ],
  "author": "Johan",
  "license": "MIT",
  "engines": {
    "node": "7.6.0"
  }
}
```

Heroku node.js Hallo Welt online lesen: https://riptutorial.com/de/heroku/topic/9897/heroku-node-js-hallo-welt

Kapitel 8: Heroku Postgres

Examples

Zurücksetzen der Postgres-Datenbank in Heroku

Schritte zum Zurücksetzen der Datenbank in Heroku:

1. Löschen Sie die Datenbank, wenn SHARED_DATABASE_URL verwendet wird:

```
heroku pg:reset DATABASE
```

2. Erstellen Sie die Datenbank mit nichts neu:

```
heroku run rake db:migrate
```

3. Füllen Sie die Datenbank mit Ihren Seed-Daten:

```
heroku run rake db:seed
```

Die Schritte 2 und 3 können durch Ausführen des Befehls zu einem Befehl kombiniert werden:

```
heroku run rake db:setup
```

So kopieren Sie die Heroku-Datenbank in die lokale Datenbank

Schritte zum Kopieren der Heroku-Datenbank in die lokale Datenbank:

1. Kopiervorgang in Terminal ausführen:

```
heroku pg:pull DATABASE_URL change_to_your_data_base_name --app change_to_your_app_name
```

2. Ändern Sie den Datenbankbesitzer mit dieser Abfrage:

```
GRANT ALL PRIVILEGES ON DATABASE change_to_your_data_base_name to change_to_your_user; ALTER
DATABASE change_to_your_data_base_name OWNER TO change_to_your_user;
```

3. Generieren Sie die Abfrage für alle Tabellen in Ihrer Datenbank und führen Sie sie aus:

```
SELECT 'ALTER TABLE '|| schemaname || '.' || tablename ||' OWNER TO change_to_your_user;' FROM
pg_tables WHERE NOT schemaname IN ('pg_catalog', 'information_schema') ORDER BY schemaname,
tablename;
```

Heroku Postgres online lesen: https://riptutorial.com/de/heroku/topic/6239/heroku-postgres

Kapitel 9: Heroku-Fehlercodes

Einführung

Wenn bei Ihrer App ein Fehler auftritt, gibt Heroku eine Standardfehlerseite mit dem HTTP-Statuscode 503 zurück. Um Ihnen das Debuggen des zugrunde liegenden Fehlers zu erleichtern, fügt die Plattform Ihren Protokollen außerdem benutzerdefinierte Fehlerinformationen hinzu. Jeder Fehlertyp erhält seinen eigenen Fehlercode, wobei alle HTTP-Fehler mit dem Buchstaben H beginnen und alle Laufzeitfehler mit R beginnen. Protokollierungsfehler beginnen mit L.

Syntax

- H10 - App ist abgestürzt
- H11 - Rückstand zu groß
- H12 - Anforderungs-Timeout
- H13 - Verbindung wurde ohne Antwort geschlossen
- H14 - Keine Webdynos laufen
- H15 - Leerlaufverbindung
- H16 - Weiterleitung zu herokuapp.com
- H17 - Schlecht formatierte HTTP-Antwort
- H18 - Serveranforderung unterbrochen
- H19 - Zeitüberschreitung beim Backend
- H20 - Zeitüberschreitung beim Booten der App
- H21 - Backend-Verbindung abgelehnt
- H22 - Verbindungsgrenze erreicht
- H23 - Endpunkt ist falsch konfiguriert
- H24 - Zwangsschließung
- H25 - HTTP-Einschränkung
- H26 - Anforderungsfehler
- H27 - Clientanforderung unterbrochen
- H28 - Client-Verbindungsleerlauf
- H80 - Wartungsmodus
- H81 - Leere App
- H82 - Das freie Kontingent ist erschöpft
- H99 - Plattformfehler
- R10 - Boot-Timeout
- R12 - Timeout beenden
- R13 - Fehler beim Anhängen
- R14 - Speicherkontingent überschritten
- R15 - Speicherkontingent weit überschritten
- R16 - Freistehend
- R17 - Prüfsummenfehler
- R99 - Plattformfehler
- L10 - Pufferüberlauf entleeren

- L11 - Endpufferüberlauf
- L12 - Lokaler Pufferüberlauf
- L13 - Lokaler Lieferfehler
- L14 - Zertifikatvalidierungsfehler

Examples

H10 - App ist abgestürzt

Ein abgestürzter Web-Dyno oder ein Boot-Timeout beim Web-Dyno führt zu diesem Fehler.

```
2010-10-06T21:51:04-07:00 heroku[web.1]: State changed from down to starting
2010-10-06T21:51:07-07:00 app[web.1]: Starting process with command: `bundle exec rails server
-p 22020`
2010-10-06T21:51:09-07:00 app[web.1]: >> Using rails adapter
2010-10-06T21:51:09-07:00 app[web.1]: Missing the Rails 2.3.5 gem. Please `gem install -
v=2.3.5 rails`, update your RAILS_GEM_VERSION setting in config/environment.rb for the Rails
version you do have installed, or comment out RAILS_GEM_VERSION to use the latest version
installed.
2010-10-06T21:51:10-07:00 heroku[web.1]: Process exited
2010-10-06T21:51:12-07:00 heroku[router]: at=error code=H10 desc="App crashed" method=GET
path="/" host=myapp.herokuapp.com fwd=17.17.17.17 dyno= connect= service= status=503 bytes=
```

H11 - Rückstand zu groß

Wenn HTTP-Anforderungen schneller eintreffen, als sie von Ihrer Anwendung verarbeitet werden können, können sie auf einer Reihe von Routern einen großen Rückstand bilden. Wenn der Rückstand eines bestimmten Routers einen Schwellenwert überschreitet, stellt der Router fest, dass Ihre Anwendung nicht mit dem eingehenden Anforderungsvolumen Schritt hält. Für jede eingehende Anforderung wird ein Fehler H11 angezeigt, solange der Rückstand über dieser Größe liegt. Der genaue Wert dieses Schwellenwerts kann sich abhängig von verschiedenen Faktoren ändern, z. B. der Anzahl der Dynos in Ihrer App, der Antwortzeit für einzelne Anforderungen und dem normalen Anforderungsvolumen Ihrer App.

```
2010-10-06T21:51:07-07:00 heroku[router]: at=error code=H11 desc="Backlog too deep" method=GET
path="/" host=myapp.herokuapp.com fwd=17.17.17.17 dyno= connect= service= status=503 bytes=
```

Die Lösung besteht darin, den Durchsatz Ihrer App zu erhöhen, indem Sie mehr Dynos hinzufügen, Ihre Datenbank optimieren (z. B. einen Index hinzufügen) oder den Code selbst beschleunigen. Wie immer ist die Leistungssteigerung sehr anwendungsspezifisch und erfordert ein Profiling.

H12 - Anforderungs-Timeout

Eine HTTP-Anforderung hat länger als 30 Sekunden gedauert. Im folgenden Beispiel dauert eine Rails-App 37 Sekunden, um die Seite zu rendern. Der HTTP-Router gibt einen 503 zurück, bevor Rails seinen Anforderungszyklus abschließt. Der Rails-Prozess wird jedoch fortgesetzt und die Abschlussmeldung wird nach der Routernachricht angezeigt.

```
2010-10-06T21:51:07-07:00 app[web.2]: Processing PostController#list (for 75.36.147.245 at
2010-10-06 21:51:07) [GET]
2010-10-06T21:51:08-07:00 app[web.2]: Rendering template within layouts/application
2010-10-06T21:51:19-07:00 app[web.2]: Rendering post/list
2010-10-06T21:51:37-07:00 heroku[router]: at=error code=H12 desc="Request timeout" method=GET
path="/" host=myapp.herokuapp.com fwd=17.17.17.17 dyno=web.1 connect=6ms service=30001ms
status=503 bytes=0
2010-10-06T21:51:42-07:00 app[web.2]: Completed in 37000ms (View: 27, DB: 21) | 200 OK
[http://myapp.heroku.com/]
```

Diese 30-Sekunden-Grenze wird vom Router gemessen und umfasst alle im Dyno verbrachten Zeiten, einschließlich der eingehenden Verbindungswarteschlange des Kernels und der App selbst.

H13 - Verbindung wurde ohne Antwort geschlossen

Dieser Fehler wird ausgelöst, wenn ein Prozess in Ihrem Web-Dyno eine Verbindung akzeptiert, den Socket dann schließt, ohne etwas zu schreiben.

```
2010-10-06T21:51:37-07:00 heroku[router]: at=error code=H13 desc="Connection closed without
response" method=GET path="/" host=myapp.herokuapp.com fwd=17.17.17.17 dyno=web.1
connect=3030ms service=9767ms status=503 bytes=0
```

Dies kann beispielsweise der Fall sein, wenn ein Unicorn-Webserver mit einem Timeout von weniger als 30 Sekunden konfiguriert ist und eine Anforderung vor dem Timeout nicht von einem Worker verarbeitet wurde. In diesem Fall schließt Unicorn die Verbindung, bevor Daten geschrieben werden, was zu einem H13 führt.

H14 - Keine Webdynos laufen

Dies ist höchstwahrscheinlich das Ergebnis der Skalierung Ihrer Webdynos auf 0 Dynos. Um dies zu beheben, skalieren Sie Ihre Webdynos auf einen oder mehrere Dynos:

```
$ heroku ps:scale web=1
```

Verwenden Sie den Befehl heroku ps, um den Status Ihrer Webdynos zu ermitteln.

```
2010-10-06T21:51:37-07:00 heroku[router]: at=error code=H14 desc="No web processes running"
method=GET path="/" host=myapp.herokuapp.com fwd=17.17.17.17 dyno= connect= service=
status=503 bytes=
```

H15 - Leerlaufverbindung

Der Prüfling sendete keine vollständige Antwort und wurde aufgrund von 55 Sekunden Inaktivität abgebrochen. In der Antwort wurde beispielsweise eine `Content-Length` von 50 Byte angegeben, die nicht rechtzeitig gesendet wurde.

```
2010-10-06T21:51:37-07:00 heroku[router]: at=error code=H15 desc="Idle connection" method=GET
path="/" host=myapp.herokuapp.com fwd=17.17.17.17 dyno=web.1 connect=1ms service=55449ms
status=503 bytes=18
```

Heroku-Fehlercodes online lesen: https://riptutorial.com/de/heroku/topic/8321/heroku-fehlercodes

Kapitel 10: Heroku-Grenzen

Examples

Liste aller Einschränkungen der Heroku-Plattform

1. Protokolle: Standardmäßig erlaubt Heroku nur 1500 Zeilen konsolidierter Protokolle. Wenn mehr als 1500 Zeilen Protokolle erforderlich sind, müssen die von Heroku bereitgestellten Addons verwendet werden.

2. Router: Die HTTP-Anforderung hat ein Zeitlimit von 30 Sekunden für die erste Antwort und danach von 55 Sekunden. Maximal 1 MB Puffer für die Antwort zulässig.

3. Dynos: Dyno- *Speicherlimits* basierend auf dem ausgewählten Typ . Für freie Dynos werden Schlafstunden festgelegt, in denen nach 30 Minuten Inaktivität geschlafen wird. Verifizierte Konten verfügen außerdem über einen monatlichen Pool von 1000 Free-Dyno-Stunden, und ungeprüfte Konten erhalten 550. Eine Anwendung kann bis zu 100 Dynos enthalten, und ein *Prozesstyp* kann nicht auf mehr als 10 Dynastaten skaliert werden. Ein freier Dyno-Typ kann maximal zwei gleichzeitig ausgeführte Dynos enthalten.

4. Config Vars: Das Konfigurationsschlüssel- und Wertepaar ist für eine App auf 32 KB beschränkt.

5. Build: Benutzer sind auf 75 Anfragen an Heroku Git Repos pro Stunde, App und Benutzer beschränkt. Die unkomprimierte Größe während des Bezahlvorgangs kann nicht mehr als 1 GB erreichen. Die Slug-Größe ist auf 300 MB begrenzt und die Kompilierungsdauer darf 15 Minuten nicht überschreiten.

6. Datenclips: Jede Abfrage kann maximal 10 Minuten dauern und maximal 100.000 Zeilen zurückgeben.

7. Heroku Postgres: Die Ausfallzeit variiert mit verschiedenen Stufen zwischen 4 Stunden und 15 Minuten pro Monat.

8. API-Limits: Die maximale Anzahl von Aufrufen an die Heroku-API ist auf 2400 / Stunde beschränkt.

9. Mitgliedschaftsgrenzen: Für ein Unternehmenskonto sind maximal 500 Mitglieder und für andere Mitglieder 25 Mitglieder zulässig.

10. Anzahl der Anwendungen: Maximal 100 Apps können von einem verifizierten Benutzer erstellt werden. Nicht verifizierte Benutzer sind auf 5 Anwendungen beschränkt.

Heroku-Grenzen online lesen: https://riptutorial.com/de/heroku/topic/6190/heroku-grenzen

Kapitel 11: Pipelines

Syntax

- Heroku-Pipelines: <Installieren | Erstellen | Fördern> ...

Bemerkungen

Eine Pipeline ist eine Gruppe von Heroku-Apps, die dieselbe Codebase verwenden. Apps in einer Pipeline werden in die Phasen "Review", "Entwicklung", "Staging" und "Produktion" unterteilt, die verschiedene Bereitstellungsschritte in einem kontinuierlichen Bereitstellungsworkflow darstellen.

Examples

Pipelines über die CLI

Installation der Pipeline

Nach der Installation von Heroku Toolbelt ist auch ein Pipelines-Plugin erforderlich.

```
heroku plugins:install heroku-pipelines
```

Pipelines erstellen

Sie müssen mit einer App beginnen, um sie zur Pipeline hinzuzufügen, obwohl dies nicht für eine bestimmte Phase erforderlich ist. Wenn Sie `--stage STAGE` nicht angeben, wird die CLI zum entsprechenden Zeitpunkt erraten, Sie können jedoch auch die Standardeinstellung überschreiben. Der Name der Pipeline wird ebenfalls anhand des Anwendungsnamens erraten, kann jedoch überschrieben werden, indem entweder der `NAME` in die Befehlszeile eingefügt oder ein anderer Name eingegeben wird, wenn Sie dazu aufgefordert werden.

```
heroku pipelines:create -a example
```

Fördern

Die Ziel-App (s) werden automatisch von der nachgelagerten Stufe bestimmt

```
heroku pipelines:promote -r staging
```

Es ist auch möglich, für eine bestimmte App (oder eine Reihe von Apps) zu werben.

```
heroku pipelines:promote -r staging --to my-production-app1,my-production-app2
```

Hilfebefehl

Eine vollständige Liste von Pipelines-Befehlen mit Verwendungsdetails ist in der Konsole verfügbar

```
heroku help pipelines
```

Pipelines online lesen: https://riptutorial.com/de/heroku/topic/2389/pipelines

Kapitel 12: Protokolle

Syntax

- Heroku-Protokolle
- $ heroku logs -n 200
- $ heroku logs --tail
- $ heroku protokolliert - Dyno-Router
- $ heroku logs - Quellen-App
- $ heroku logs - Quellen-App - Dyno-Arbeiter
- $ heroku logs - Quellen-App - Schwanz

Examples

Arten von Protokollen

Heroku fasst drei Kategorien von Protokollen für Ihre App zusammen:

- **App-Protokolle** - Ausgabe aus Ihrer Anwendung. Dazu gehören Protokolle, die innerhalb Ihrer Anwendung, des Anwendungsservers und der Bibliotheken generiert werden. (Filter: `--source app`)
- **Systemprotokolle** - Meldungen zu Aktionen, die von der Heroku-Plattform-Infrastruktur für Ihre App ausgeführt werden, z. (Filter: `--source heroku`)
- **API-Protokolle** - Meldungen zu Verwaltungsaktionen, die von Ihnen und anderen an Ihrer App arbeitenden Entwicklern ausgeführt werden, z. (Filter: `--source heroku --dyno api`)

Protokollformat

Jede Protokollzeile wird wie folgt formatiert:

```
timestamp source[dyno]: message
```

- **Zeitstempel** - Das Datum und die Uhrzeit, die zu dem Zeitpunkt aufgezeichnet wurden, als die Protokollzeile vom Prüfstand oder der Komponente erzeugt wurde. Der Zeitstempel hat das von RFC5424 festgelegte Format und eine Genauigkeit im Mikrosekundenbereich.
- **Quelle** - Alle Dynas Ihrer App (Webdynos, Hintergrundarbeiter, Cron) haben die Quelle, `app` . Alle Systemkomponenten von Heroku (HTTP-Router, Dyno-Manager) haben die Quelle `heroku` .
- **Dyno** - Der Name des Dynos oder der Komponente, die die Protokollzeile geschrieben hat. Beispielsweise wird Worker # 3 als `worker.3` und der Heroku HTTP-Router als `router` angezeigt.

- **Message** - Der Inhalt der Protokollzeile. Zeilen, die von Dynos generiert werden, die 10000 Byte überschreiten, werden in 10000 Byte-Blöcke ohne zusätzliche Zeilenumbrüche aufgeteilt. Jeder Block wird als separate Protokollzeile übermittelt.

Protokolle anzeigen

Verwenden `heroku logs` zum Abrufen Ihrer Protokolle den Befehl `heroku logs` .

```
$ heroku logs
```

Der Befehl logs ruft standardmäßig 100 Protokollzeilen ab. Sie können die Anzahl der abzurufenden Protokollzeilen (bis zu maximal 1.500 Zeilen) mit der Option `--num` (oder `-n`) `--num` .

```
$ heroku logs -n 200
```

Echtzeit-Schwanz

Ähnlich wie `tail -f` zeigt das Echtzeitendstück die letzten Protokolle an und lässt die Sitzung für Echtzeitprotokolle zum Streamen geöffnet. Durch Anzeigen eines Live-Streams von Protokollen aus Ihrer App können Sie Einblick in das Verhalten Ihrer Live-Anwendung erhalten Debuggen Sie aktuelle Probleme. Sie können Ihre Protokolle mit `--tail` (oder `-t`) `--tail` .

```
$ heroku logs --tail
```

Wenn Sie fertig sind, drücken Sie Strg + C, um zur Eingabeaufforderung zurückzukehren.

Protokollfilterung

Wenn Sie nur Protokolle mit einer bestimmten Quelle, einem bestimmten Dyno oder beiden `--source` , können Sie die `--source` (oder `-s`) und `--dyno` (oder `-d`) verwenden:

```
$ heroku logs --dyno router
2012-02-07T09:43:06.123456+00:00 heroku[router]: at=info method=GET path="/stylesheets/dev-
center/library.css" host=devcenter.heroku.com fwd="204.204.204.204" dyno=web.5 connect=1ms
service=18ms status=200 bytes=13
2012-02-07T09:43:06.123456+00:00 heroku[router]: at=info method=GET path="/articles/bundler"
host=devcenter.heroku.com fwd="204.204.204.204" dyno=web.6 connect=1ms service=18ms status=200
bytes=20375

$ heroku logs --source app
2012-02-07T09:45:47.123456+00:00 app[web.1]: Rendered shared/_search.html.erb (1.0ms)
2012-02-07T09:45:47.123456+00:00 app[web.1]: Completed 200 OK in 83ms (Views: 48.7ms |
ActiveRecord: 32.2ms)
2012-02-07T09:45:47.123456+00:00 app[worker.1]: [Worker(host:465cf64e-61c8-46d3-b480-
362bfd4ecff9 pid:1)] 1 jobs processed at 23.0330 j/s, 0 failed ...
2012-02-07T09:46:01.123456+00:00 app[web.6]: Started GET "/articles/buildpacks" for 4.1.81.209
at 2012-02-07 09:46:01 +0000

$ heroku logs --source app --dyno worker
```

```
2012-02-07T09:47:59.123456+00:00 app[worker.1]: [Worker(host:260cf64e-61c8-46d3-b480-
362bfd4ecff9 pid:1)] Article#record_view_without_delay completed after 0.0221
2012-02-07T09:47:59.123456+00:00 app[worker.1]: [Worker(host:260cf64e-61c8-46d3-b480-
362bfd4ecff9 pid:1)] 5 jobs processed at 31.6842 j/s, 0 failed ...
```

Sie können die `--tail` mit `--tail` , um einen Echtzeit-Stream gefilterter Ausgabe zu erhalten.

```
$ heroku logs --source app --tail
```

Protokolle online lesen: https://riptutorial.com/de/heroku/topic/8327/protokolle

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Heroku	Community, rdegges, thejonanshow
2	Abhängigkeiten	Thamilan
3	Befehlszeile	Sender
4	Buildpack	Thamilan
5	Einsatz	Sender
6	Heroku Add-Ons	jophab
7	Heroku node.js Hallo Welt	Johan Hoeksma
8	Heroku Postgres	Denis Savchuk, Hardik Kanjariya ツ, Thamilan
9	Heroku-Fehlercodes	Sender
10	Heroku-Grenzen	autoboxer, Thamilan
11	Pipelines	Thamilan
12	Protokolle	Sender