EBook Gratuito

APPRENDIMENTO heroku

Free unaffiliated eBook created from
Stack Overflow contributors.

#heroku

Sommario

Di

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

Capitolo 1: Iniziare con heroku

Osservazioni

Heroku è un noto fornitore di piattaforma come servizio (PaaS) che consente agli sviluppatori di distribuire facilmente applicazioni Web senza un team operativo. Heroku è in giro dal 2007 ed è ora di proprietà di Salesforce .

Questa sezione fornisce una panoramica di cosa sia Heroku e perché uno sviluppatore potrebbe volerlo utilizzare.

Dovrebbe anche menzionare qualsiasi argomento di grandi dimensioni all'interno di Heroku e collegarsi agli argomenti correlati. Poiché la documentazione di Heroku è nuova, potrebbe essere necessario creare versioni iniziali di tali argomenti correlati.

Examples

Installazione o configurazione

Per creare e gestire localmente le app di Heroku avrai bisogno di Heroku Toolbelt, ecco alcuni modi per ottenerlo.

Scaricare

Scarica il programma di installazione di Heroku Toolbelt dal sito Web di Heroku.

homebrew

Installa `heroku` con `brew` :

```
brew install heroku
```

Debian / Ubuntu

Esegui questo script:

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

Questo script aggiunge il repository Heroku ad apt, installa la chiave di rilascio di Heroku, installa Heroku Toolbelt e installa Ruby se ne hai bisogno.

Come con qualsiasi script che trovi online e condividi direttamente su bash, ti consigliamo vivamente di leggere prima la fonte .

Usando il cinturone Heroku

Crea un'applicazione

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

Distribuire a Heroku

```
git push heroku master
```

Apri la tua applicazione in un browser

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

Elenca i comandi di Heroku

```
heroku commands
```

Aiuto generale

```
heroku help
```

Aiuto per un comando specifico

```
heroku help <command>
```

Creazione di applicazioni Heroku

Puoi usare il comando `heroku create` per creare un'applicazione Heroku. Ogni applicazione distribuita su Heroku ha una propria base di codice, variabili d'ambiente, componenti aggiuntivi, ecc.

Ogni applicazione Heroku ha un nome globalmente unico. Se si tenta di creare un'applicazione Heroku il cui nome è già stato acquisito, si verificherà un errore.

Ecco come è possibile creare una nuova applicazione Heroku:

```
heroku create [app_name]
```

Se non si specifica il nome di un'applicazione durante l'esecuzione di `heroku create` , Heroku creerà un nome di applicazione casuale per te.

Puoi anche specificare la regione Amazon in cui creare la tua applicazione Heroku. Per impostazione predefinita, tutte le applicazioni Heroku sono create nella `us` regione. Se desideri modificare la regione, puoi farlo creando l'applicazione in questo modo:

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

Al momento, ci sono solo due regioni pubbliche: `us` , `eu` (Europa).

Leggi Iniziare con heroku online: https://riptutorial.com/it/heroku/topic/959/iniziare-con-heroku

Capitolo 2: Buildpack

Examples

Impostazione dei Buildpack

Heroku supporta ufficialmente i buildpack per Ruby, Node.js, Clojure, Python, Java, Gradle, Grails, Scala, Play, PHP e Go.

I buildpack vengono rilevati automaticamente da Heroku nell'ordine sopra, tuttavia, può anche essere impostato manualmente tramite CLI utilizzando:

1. Al momento della creazione di app

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

2. manualmente,

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

Il nome del buildpack può essere specificato usando la stenografia o l'URL. Come per il buildpack PHP,

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

o

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

Più buildpacks

Un'applicazione può anche contenere più di un buildpack. Può essere raggiunto usando `add` :

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

dove, parametro `--index` specifica l'ordine di esecuzione di buildpack.

Dire,

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

imposterà l'ordine buildpack come:

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

Ricorda: un'app Heroku ha un solo porto pubblico - 80. Quindi, uno dei due servirà in un'unica porta. Supponiamo, se `procfile` è specificato con `web: node server.js` , l'applicazione nodo verrà eseguita nella porta 80, altrimenti PHP. Tuttavia, la build verrà eseguita nell'ordine specificato. Se è necessario più di un'applicazione, impostare più progetti e farli comunicare tra loro.

Leggi Buildpack online: https://riptutorial.com/it/heroku/topic/6126/buildpack

Capitolo 3: Codici di errore Heroku

introduzione

Ogni volta che la tua app presenta un errore, Heroku restituirà una pagina di errore standard con il codice di stato HTTP 503. Per aiutarti a eseguire il debug dell'errore sottostante, la piattaforma aggiungerà anche informazioni di errore personalizzate ai tuoi registri. Ogni tipo di errore riceve il proprio codice di errore, con tutti gli errori HTTP che iniziano con la lettera H e tutti gli errori di runtime che iniziano con R. Gli errori di registrazione iniziano con L.

Sintassi

- H10 - L'app si è bloccata
- H11 - Backlog troppo profondo
- H12 - Timeout richiesta
- H13 - Connessione chiusa senza risposta
- H14 - Nessun dynos Web in esecuzione
- H15 - Connessione inattiva
- H16 - Reindirizza a herokuapp.com
- H17 - Risposta HTTP scarsamente formattata
- H18 - Richiesta server interrotta
- H19 - Timeout della connessione back-end
- H20 - Timeout di avvio dell'app
- H21 - Connessione back-end rifiutata
- H22 - Limite di connessione raggiunto
- H23 - Endpoint non configurato correttamente
- H24 - Chiusura forzata
- H25 - Restrizione HTTP
- H26 - Richiesta di errore
- H27 - Richiesta client interrotta
- H28 - Connessione client inattiva
- H80 - Modalità di manutenzione
- H81 - App vuota
- H82 - Dyno quota libera esaurita
- H99 - Errore di piattaforma
- R10 - Timeout di avvio
- R12 - Timeout di uscita
- R13 - Allega errore
- R14 - Quota di memoria superata
- R15 - La quota di memoria è stata ampiamente superata
- R16 - Staccato
- R17 - Errore di checksum
- R99 - Errore di piattaforma
- L10 - Scarico del buffer di scarico

- L11 - Troppo pieno del buffer di coda
- L12 - Overflow del buffer locale
- L13 - Errore di consegna locale
- L14 - Errore di convalida del certificato

Examples

H10 - L'app si è bloccata

Un dyno web arrestato o un timeout di avvio sul web dyno presenterà questo errore.

```
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 - Backlog troppo profondo

Quando le richieste HTTP arrivano più velocemente di quanto la tua applicazione possa elaborarle, possono formare un grosso backlog su un numero di router. Quando il backlog su un determinato router supera una soglia, il router determina che l'applicazione non è al passo con il suo volume di richieste in entrata. Verrà visualizzato un errore H11 per ogni richiesta in entrata purché il backlog abbia superato questa dimensione. Il valore esatto di questa soglia può variare in base a vari fattori, come il numero di dynos nell'app, il tempo di risposta per le singole richieste e il volume di richieste normale dell'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=
```

La soluzione è aumentare il throughput della tua app aggiungendo più dynos, ottimizzando il tuo database (ad esempio aggiungendo un indice) o rendendo il codice stesso più veloce. Come sempre, l'aumento delle prestazioni è altamente specifico per le applicazioni e richiede la profilazione.

H12 - Timeout richiesta

Una richiesta HTTP ha richiesto più di 30 secondi per essere completata. Nell'esempio seguente, un'app Rails impiega 37 secondi per il rendering della pagina; il router HTTP restituisce un 503 prima che Rails completi il suo ciclo di richieste, ma il processo Rails continua e il messaggio di completamento viene visualizzato dopo il messaggio del router.

```
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/]
```

Questo limite di 30 secondi viene misurato dal router e include tutto il tempo trascorso nel banco prova, inclusa la coda di connessione in entrata del kernel e l'app stessa.

H13 - Connessione chiusa senza risposta

Questo errore viene generato quando un processo nel tuo dino web accetta una connessione, ma poi chiude il socket senza scrivere nulla su di esso.

```
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
```

Un esempio in cui ciò potrebbe accadere è quando un server Web Unicorn è configurato con un timeout inferiore a 30 secondi e una richiesta non è stata elaborata da un lavoratore prima che si verifichi il timeout. In questo caso, Unicorn chiude la connessione prima che venga scritto qualsiasi dato, risultando in un H13.

H14 - Nessun dynos Web in esecuzione

Questo è probabilmente il risultato del ridimensionamento dei tuoi dynos Web fino a 0 dynos. Per risolvere il problema, ridimensiona i tuoi dynos Web su 1 o più dynos:

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

Usa il comando heroku ps per determinare lo stato dei tuoi dynos web.

```
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 - Connessione inattiva

Il banco prova non ha inviato una risposta completa ed è stato interrotto a causa di 55 secondi di inattività. Ad esempio, la risposta indica una `Content-Length` del `Content-Length` di 50 byte che non sono stati inviati in tempo.

```
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
```

Leggi Codici di errore Heroku online: https://riptutorial.com/it/heroku/topic/8321/codici-di-errore-heroku

Capitolo 4: Condotte

Sintassi

- condutture heroku: <installa | crea | promuovi> ...

Osservazioni

Una pipeline è un gruppo di app di Heroku che condividono la stessa base di codice. Le app in una pipeline sono raggruppate in fasi di "revisione", "sviluppo", "staging" e "produzione" che rappresentano fasi di distribuzione diverse in un flusso di lavoro di consegna continua.

Examples

Pipeline tramite la CLI

Installazione della pipeline

Una volta installato Heroku Toolbelt, richiede anche il plug-in Pipelines .

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

Creazione di pipeline

È necessario iniziare con un'app da aggiungere alla pipeline, sebbene non sia necessario per una determinata fase. Se non si specifica `--stage STAGE` , la CLI indovina nella fase appropriata, ma consente anche di sovrascrivere il valore predefinito. Il nome della pipeline verrà indovinato anche dal nome dell'app, ma può essere sovrascritto aggiungendo il `NAME` sulla riga di comando o immettendo un nome diverso quando richiesto.

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

Promozione

Le app di destinazione saranno determinate automaticamente dalla fase a valle

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

È anche possibile promuovere un'app specifica (o una serie di app)

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

Aiuto Comando

Un elenco completo di comandi Pipelines con i dettagli sull'utilizzo è disponibile nella console

```
heroku help pipelines
```

Leggi Condotte online: https://riptutorial.com/it/heroku/topic/2389/condotte

Capitolo 5: dipendenze

Sintassi

- "dipendenze": {...}

Examples

Bower dependency

Per installare automaticamente Bower e i suoi componenti, è necessario

1. Specificare la dipendenza del bower in `package.json` :

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

2. Utilizzare gli `scripts` per eseguire un comando `postinstall`

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

3. Creare un file `.bowerrc` per impostare la directory per bower_components da installare. Altrimenti bower_components sono installati nella directory principale.

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

Ora, Heroku esegue automaticamente il comando di `bower install` dopo `npm install`

Leggi dipendenze online: https://riptutorial.com/it/heroku/topic/6665/dipendenze

Capitolo 6: Distribuzione

Sintassi

- git push heroku master

Examples

Distribuzione con Git

Tracciamento della tua app in git

Prima di poter inviare un'app a Heroku, è necessario inizializzare un repository Git locale e impegnare i file su di esso. Ad esempio, se hai un'app in una directory, myapp, quindi crea un nuovo repository per esso:

```
$ 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
...
```

Questo è un repository locale, ora residente nella directory `.git` . Nulla è stato inviato da nessuna parte ancora; dovrai creare un telecomando e fare una spinta per distribuire il tuo codice su Heroku.

Creazione di un telecomando Heroku

```
$ 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
```

Repository Git con un'applicazione esistente. Il guru heroku: comando remoto aggiungerà questo telecomando per te in base alle tue applicazioni git url.

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

Distribuzione del codice

dovrai specificare un ramo remoto da inviare. Puoi fare la tua prima spinta:

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

Per inserire un ramo diverso da quello principale, utilizzare questa sintassi:

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

Leggi Distribuzione online: https://riptutorial.com/it/heroku/topic/8325/distribuzione

Capitolo 7: Heroku Add-ons

introduzione

Dettagli e istruzioni per l'uso dei vari componenti aggiuntivi disponibili con Heroku.

Examples

Heroku Scheduler

Installazione di Heroku Scheduler

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

Leggi Heroku Add-ons online: https://riptutorial.com/it/heroku/topic/8906/heroku-add-ons

Capitolo 8: Heroku Limits

Examples

Elenco di tutte le limitazioni nella piattaforma Heroku

1. Log: per impostazione predefinita, Heroku consente solo 1500 righe di log consolidati. Quando sono richieste più di 1500 righe di log, è necessario utilizzare gli addon forniti da Heroku.

2. Router: la richiesta HTTP ha un timeout di 30 secondi per la risposta iniziale e un timeout di 55 secondi. È consentito un massimo di 1 MB di buffer per la risposta.

3. Dynos: *limiti di memoria* Dyno in base al tipo scelto. Per i dynos gratuiti, le ore di sonno sono imposte dove dormono dopo 30 minuti di inattività. Inoltre, gli account verificati sono dotati di un pool mensile di 1000 ore di prova libere e gli account non verificati ricevono 550. Un'applicazione può avere fino a 100 dynos e un *tipo di processo* non può essere ridimensionato a più di 10 dynos. Il tipo di dyno libero può avere un massimo di due dinamiche in esecuzione simultanee.

4. Config Vars: la chiave di configurazione e la coppia di valori sono limitate a 32kb per un'app.

5. Build: gli utenti sono limitati a 75 richieste di repository Git Heroku all'ora, per app, per utente. Le dimensioni non compresse durante il checkout non possono raggiungere più di 1 GB. La dimensione di Slug è limitata a 300 MB e la lunghezza della compilazione non può superare i 15 minuti.

6. Clip di dati: ogni query può essere eseguita fino a un massimo di 10 minuti e può restituire un massimo di 100.000 righe.

7. Heroku Postgres: i tempi di inattività variano con livelli diversi da meno di 4 ore a 15 minuti al mese.

8. Limiti API: le chiamate massime verso l'API Heroku sono limitate a 2400 / ora.

9. Limiti di appartenenza: per un account aziendale, massimo 500 membri e per gli altri, sono ammessi 25 membri.

10. Conteggio applicazioni: è possibile creare un massimo di 100 app da un utente verificato. Gli utenti non verificati sono limitati a 5 applicazioni.

Leggi Heroku Limits online: https://riptutorial.com/it/heroku/topic/6190/heroku-limits

Capitolo 9: Heroku node.js Ciao mondo

Osservazioni

accesso

`heroku login`

crea app

`heroku create` o `heroku create your_name`

clonare l'esempio

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

visita l'app nel tuo browser

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

Prova **facoltativa** locale:

```
heroku local web
```

controllare: lolhost: 5000

Che cosa è diverso da una normale app node.js? package.json

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

index.js

```
process.env.PORT
```

Porto locale: 5000. Heroku lo mapperà alla porta 80 dell'URL della tua app.

Examples

Heroku node.js ciao mondo

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"
  }
}
```

Leggi Heroku node.js Ciao mondo online: https://riptutorial.com/it/heroku/topic/9897/heroku-node-js-ciao-mondo

Capitolo 10: Heroku Postgres

Examples

Come ripristinare il database Postgres in Heroku

Passos per ripristinare il database in Heroku:

1. Rilasciare il database quando viene utilizzato SHARED_DATABASE_URL:

```
heroku pg:reset DATABASE
```

2. Ricrea il database con niente in esso:

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

3. Popolare il database con i tuoi dati seme:

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

I passaggi 2 e 3 possono essere combinati in un unico comando eseguendo questo:

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

Come copiare il database heroku nel database locale

Passos per copiare il database heroku nel database locale:

1. Eseguire il processo di copia nel terminale:

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

2. Cambia proprietario db usando questa query:

```
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. Genera ed esegui query per tutte le tabelle nel tuo database:

```
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;
```

Leggi Heroku Postgres online: https://riptutorial.com/it/heroku/topic/6239/heroku-postgres

Capitolo 11: logs

Sintassi

- $ logs di heroku
- $ heroku registra -n 200
- $ heroku logs --tail
- $ logs heroku - router fisso
- $ heroku registra l'app --source
- $ heroku registra l'app --source --dyno worker
- $ heroku registra l'app --source --tail

Examples

Tipi di log

Heroku aggrega tre categorie di log per la tua app:

- **Registri delle app** : uscita dalla tua applicazione. Ciò includerà i log generati dall'interno dell'applicazione, dal server applicazioni e dalle librerie. (Filtro: `--source app`)
- **Registri di sistema** - Messaggi relativi alle azioni intraprese dall'infrastruttura della piattaforma Heroku per conto della tua app, come ad esempio: riavvio di un processo bloccato, sospensione o risveglio di un dinoo Web o visualizzazione di una pagina di errore a causa di un problema nell'app. (Filtro: `--source heroku`)
- **Log delle API** : messaggi sulle azioni amministrative intraprese da te e altri sviluppatori che lavorano sulla tua app, come ad esempio: distribuzione di nuovo codice, ridimensionamento della formazione del processo o attivazione della modalità di manutenzione. (Filtro: `--source heroku --dyno api`)

Formato del registro

Ogni riga di registro è formattata come segue:

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

- **Timestamp** - La data e l'ora registrate nel momento in cui la linea di log è stata prodotta dal banco prova o componente. Il timestamp è nel formato specificato da RFC5424 e include la precisione in microsecondi.
- **Fonte** : tutte le dinamiche della tua app (web dynos, background worker, cron) hanno la fonte, `app` . Tutti i componenti di sistema di Heroku (router HTTP, gestore dyno) hanno la fonte, `heroku` .

- **Dyno** : il nome del banco prova o componente che ha scritto la riga del registro. Ad esempio, worker # 3 viene visualizzato come `worker.3` e il router Heroku HTTP viene visualizzato come `router` .
- **Messaggio** : il contenuto della riga di registro. Le linee generate da dinamiche che superano i 10000 byte vengono suddivise in blocchi da 10000 byte senza linee di fine riga aggiuntive. Ogni blocco viene inviato come una linea di registro separata.

Visualizza i log

Per recuperare i tuoi registri, usa il comando `heroku logs` .

```
$ heroku logs
```

Il comando logs recupera 100 linee di registro per impostazione predefinita. È possibile specificare il numero di righe di registro da recuperare (fino a un massimo di 1.500 linee) utilizzando l' `--num` (o `-n`).

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

Coda in tempo reale

Simile alla `tail -f` , coda in tempo reale mostra i registri recenti e lascia aperta la sessione per i flussi in tempo reale in streaming. Visualizzando un flusso di registri dal vivo dalla tua app, puoi ottenere informazioni sul comportamento della tua applicazione live e eseguire il debug dei problemi attuali. Puoi `--tail` tuoi registri usando `--tail` (o `-t`).

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

Al termine, premere Ctrl + C per tornare al prompt.

Log Filtering

Se si desidera recuperare i registri con una determinata fonte, un certo banco di `--source` o entrambi, è possibile utilizzare gli argomenti di filtro `--source` (o `-s`) e `--dyno` (o `-d`):

```
$ 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 ...
```

Puoi anche combinare gli interruttori di filtro con `--tail` per ottenere un flusso in tempo reale di output filtrato.

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

Leggi logs online: https://riptutorial.com/it/heroku/topic/8327/logs

Capitolo 12: Riga di comando

introduzione

Heroku Command Line Interface (CLI), precedentemente noto come Heroku Toolbelt, è uno strumento per creare e gestire app di Heroku dalla riga di comando / shell di vari sistemi operativi.

Sintassi

- $ heroku --version
- $ login heroku
- $ heroku crea

Examples

Scarica e installa

OS X

Scarica ed esegui il programma di installazione di OS X.

finestre

Scarica ed esegui il programma di installazione di Windows a 32 bit a 64 bit .

Debian / Ubuntu

Esegui quanto segue per aggiungere il nostro repository apt e installare la CLI:

```
$ 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
```

Versione standalone

Scarica il tarball ed estrai in modo da poter accedere al binario dal tuo PERCORSO. Per esempio:

```
$ 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
```

Verifica la tua installazione

Per verificare l'installazione della CLI, utilizzare il comando `heroku --version` .

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

Iniziare

Ti verrà chiesto di inserire le tue credenziali Heroku la prima volta che esegui un comando; dopo la prima volta, il tuo indirizzo email e un token API verranno salvati in `~/.netrc` per uso futuro.

Generalmente è una buona idea effettuare il login e aggiungere la tua chiave pubblica subito dopo aver installato la CLI di Heroku in modo da poter usare git per spingere o clonare i repository delle app di Heroku:

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

Ora sei pronto per creare la tua prima app di Heroku:

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

Leggi Riga di comando online: https://riptutorial.com/it/heroku/topic/8324/riga-di-comando

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con heroku	Community, rdegges, thejonanshow
2	Buildpack	Thamilan
3	Codici di errore Heroku	Sender
4	Condotte	Thamilan
5	dipendenze	Thamilan
6	Distribuzione	Sender
7	Heroku Add-ons	jophab
8	Heroku Limits	autoboxer, Thamilan
9	Heroku node.js Ciao mondo	Johan Hoeksma
10	Heroku Postgres	Denis Savchuk, Hardik Kanjariya ツ, Thamilan
11	logs	Sender
12	Riga di comando	Sender