EBook Gratis

APRENDIZAJE heroku

Free unaffiliated eBook created from
Stack Overflow contributors.

#heroku

Tabla de contenido

Acerca de

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

Capítulo 1: Empezando con heroku

Observaciones

Heroku es un popular proveedor de plataforma como servicio (PaaS) que facilita a los desarrolladores el despliegue de aplicaciones web sin un equipo de operaciones. Heroku ha existido desde 2007, y ahora es propiedad de Salesforce .

Esta sección proporciona una descripción general de qué es Heroku y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema importante dentro de Heroku, y vincular a los temas relacionados. Dado que la Documentación para Heroku es nueva, es posible que deba crear versiones iniciales de esos temas relacionados.

Examples

Instalación o configuración

Para crear y administrar las aplicaciones de Heroku localmente, necesitará el cinturón de herramientas de Heroku, aquí hay algunas maneras de obtenerlo.

Descargar

Descargue el instalador Heroku Toolbelt desde el sitio web de Heroku.

Homebrew

Instalar `heroku` con `brew`

```
brew install heroku
```

Debian / Ubuntu

Ejecute este script:

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

Este script agrega el repositorio Heroku a apt, instala la clave de lanzamiento de Heroku, instala el cinturón de herramientas Heroku y luego instala Ruby si lo necesita.

Como con cualquier script que encuentre en línea y canalice directamente a bash, le recomendamos que lea la fuente primero.

Usando el Heroku Toolbelt

Crear una aplicación

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

Desplegar a Heroku

```
git push heroku master
```

Abra su aplicación en un navegador

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

Listar comandos de Heroku

```
heroku commands
```

Ayuda general

```
heroku help
```

Ayuda para un comando específico

```
heroku help <command>
```

Creando Aplicaciones Heroku

Puede usar el comando `heroku create` para crear una aplicación Heroku. Cada aplicación que implementa en Heroku tiene su propia base de código, variables de entorno, complementos, etc.

Cada aplicación Heroku tiene un nombre único a nivel mundial. Si intentas crear una aplicación Heroku cuyo nombre ya está en uso, obtendrás un error.

Así es como puedes crear una nueva aplicación de Heroku:

```
heroku create [app_name]
```

Si no especifica un nombre de aplicación cuando ejecuta `heroku create` , Heroku creará un nombre de aplicación aleatorio para usted.

También puede especificar la región de Amazon en la que se debe crear su aplicación Heroku. Por defecto, todas las aplicaciones de Heroku se crean en la región de Estados `us` . Si desea cambiar la región, puede hacerlo creando la aplicación así:

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

En este momento, solo hay dos regiones públicas: `us` , y `eu` (Europa).

Lea Empezando con heroku en línea: https://riptutorial.com/es/heroku/topic/959/empezando-con-heroku

Capítulo 2: Buildpack

Examples

Configuración de Buildpacks

Heroku es oficialmente compatible con buildpacks para Ruby, Node.js, Clojure, Python, Java, Gradle, Grails, Scala, Play, PHP y Go.

Heroku detecta automáticamente los paquetes de compilación en el orden anterior, sin embargo, también se puede configurar manualmente a través de CLI usando:

1. En el momento de la creación de la aplicación.

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

2. A mano,

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

El nombre del buildpack puede especificarse usando la taquigrafía o la URL. Al igual que para PHP buildpack,

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

o

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

Múltiples paquetes de construcción

Una aplicación también puede contener más de un buildpack. Se puede lograr usando `add` :

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

donde, el parámetro `--index` especifica el orden de ejecución de buildpack.

Decir,

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

establecerá el orden de buildpack como:

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

Recuerde: una aplicación Heroku tiene un solo puerto público: 80. Por lo tanto, cualquiera de los dos servirá en un puerto. Por `procfile` , si `procfile` se especifica con `web: node server.js` , la aplicación de nodo se ejecutará en el puerto 80, de lo contrario PHP. Sin embargo, la construcción se ejecutará en el orden especificado. Si necesita más de una aplicación, configure múltiples proyectos y haga que se comuniquen entre sí.

Lea Buildpack en línea: https://riptutorial.com/es/heroku/topic/6126/buildpack

Capítulo 3: Códigos de error de Heroku

Introducción

Siempre que su aplicación experimente un error, Heroku devolverá una página de error estándar con el código de estado HTTP 503. Sin embargo, para ayudarlo a depurar el error subyacente, la plataforma también agregará información de error personalizada a sus registros. Cada tipo de error obtiene su propio código de error, con todos los errores HTTP que comienzan con la letra H y todos los errores en tiempo de ejecución que comienzan con R. Los errores de registro comienzan con L.

Sintaxis

- H10 - La aplicación se estrelló
- H11 - Backlog demasiado profundo
- H12 - Tiempo de espera de solicitud
- H13 - Conexión cerrada sin respuesta.
- H14 - No hay dines web en ejecución
- H15 - Conexión inactiva
- H16 - Redirigir a herokuapp.com
- H17 - Respuesta HTTP mal formateada
- H18 - solicitud de servidor interrumpida
- H19 - Tiempo de espera de conexión backend
- H20 - Tiempo de espera de arranque de la aplicación
- H21 - conexión de backend rechazada
- H22 - Límite de conexión alcanzado
- H23 - Endpoint mal configurado
- H24 - Cierre forzado
- H25 - Restricción de HTTP
- H26 - Error de solicitud
- H27 - Solicitud de cliente interrumpida
- H28 - Conexión de cliente inactiva
- H80 - Modo de mantenimiento
- H81 - Aplicación en blanco
- H82 - Cuota de dinas gratis agotada
- H99 - Error de plataforma
- R10 - Tiempo de espera de arranque
- R12 - Tiempo de espera de salida
- R13 - Adjuntar error
- R14 - Cuota de memoria excedida
- R15 - Cuota de memoria superada ampliamente
- R16 - Independiente
- R17 - Error de suma de comprobación
- R99 - Error de plataforma

- L10 - Desbordamiento de búfer de drenaje
- L11 - Desbordamiento de buffer de cola
- L12 - Desbordamiento de búfer local
- L13 - Error de entrega local
- L14 - Error de validación de certificado

Examples

H10 - La aplicación se estrelló

Un error dinamométrico web o un tiempo de espera de arranque en la web presentará este error.

```
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 demasiado profundo

Cuando las solicitudes HTTP llegan más rápido de lo que su aplicación puede procesarlas, pueden formar un gran atraso en varios enrutadores. Cuando la acumulación en un enrutador en particular pasa un umbral, el enrutador determina que su aplicación no está al día con su volumen de solicitudes entrantes. Verá un error H11 para cada solicitud entrante siempre que la acumulación de pedidos supere este tamaño. El valor exacto de este umbral puede cambiar dependiendo de varios factores, como el número de dinámicos en su aplicación, el tiempo de respuesta para solicitudes individuales y el volumen de solicitudes normal de su aplicación.

```
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 solución es aumentar el rendimiento de su aplicación agregando más dinámicos, ajustando su base de datos (por ejemplo, agregando un índice) o haciendo que el código en sí sea más rápido. Como siempre, el aumento del rendimiento es altamente específico de la aplicación y requiere perfiles.

H12 - Tiempo de espera de solicitud

Una solicitud HTTP tardó más de 30 segundos en completarse. En el siguiente ejemplo, una aplicación Rails tarda 37 segundos en renderizar la página; el enrutador HTTP devuelve un 503 antes de que Rails complete su ciclo de solicitud, pero el proceso de Rails continúa y el mensaje de finalización se muestra después del mensaje del enrutador.

```
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/]
```

El enrutador mide este límite de 30 segundos e incluye todo el tiempo empleado en el dinamómetro, incluida la cola de conexión entrante del kernel y la aplicación en sí.

H13 - Conexión cerrada sin respuesta.

Este error se produce cuando un proceso en su sitio web acepta una conexión, pero luego cierra el zócalo sin escribirle nada.

```
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 ejemplo en el que esto puede suceder es cuando un servidor web Unicorn está configurado con un tiempo de espera inferior a 30 segundos y un trabajador no ha procesado una solicitud antes de que se agote el tiempo de espera. En este caso, Unicorn cierra la conexión antes de escribir cualquier dato, lo que resulta en un H13.

H14 - No hay dines web en ejecución

Es muy probable que este sea el resultado de la reducción de los dinamismos web a 0 dinámicos. Para solucionarlo, escala tus dinamismos web a 1 o más dinamismos:

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

Use el comando heroku ps para determinar el estado de sus dinamas 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 - Conexión inactiva

El dino no envió una respuesta completa y se terminó debido a 55 segundos de inactividad. Por ejemplo, la respuesta indicaba una `Content-Length` de `Content-Length` de 50 bytes que no se enviaron a tiempo.

```
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
```

Lea Códigos de error de Heroku en línea: https://riptutorial.com/es/heroku/topic/8321/codigos-de-error-de-heroku

Capítulo 4: Dependencias

Sintaxis

- "dependencias": {...}

Examples

Dependencia de la planta

Para instalar automáticamente bower y sus componentes, uno debe

1. Especifique la dependencia de `package.json` en `package.json` :

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

2. Use `scripts` para ejecutar un comando `postinstall`

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

3. Cree un archivo `.bowerrc` para configurar el directorio para que instalen los bower_components. De lo contrario, los bower_components se instalan en el directorio raíz.

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

Ahora, Heroku ejecuta automáticamente `bower install` comando después de `npm install`

Lea Dependencias en línea: https://riptutorial.com/es/heroku/topic/6665/dependencias

Capítulo 5: Despliegue

Sintaxis

- git push heroku master

Examples

Despliegue con Git

Seguimiento de su aplicación en git

Antes de poder enviar una aplicación a Heroku, deberás inicializar un repositorio local de Git y comprometer tus archivos en él. Por ejemplo, si tiene una aplicación en un directorio, myapp, cree un nuevo repositorio para ella:

```
$ 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
...
```

Este es un repositorio local, que ahora reside dentro del directorio `.git` . Nada ha sido enviado a ninguna parte todavía; Tendrá que crear un control remoto y hacer un impulso para implementar su código en Heroku.

Creando un control remoto de 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
```

Git repositorio con una aplicación existente. El comando heroku git: remote agregará este control remoto basado en las aplicaciones git url.

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

Despliegue de código

necesitarás especificar una rama remota para empujar a. Puedes hacer tu primer empujón:

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

Para empujar una rama que no sea la maestra, use esta sintaxis:

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

Lea Despliegue en línea: https://riptutorial.com/es/heroku/topic/8325/despliegue

Capítulo 6: Heroku complementos

Introducción

Detalles y cómo usar las instrucciones sobre varios complementos que están disponibles con Heroku.

Examples

Heroku Scheduler

Instalación de Heroku Scheduler

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

Lea Heroku complementos en línea: https://riptutorial.com/es/heroku/topic/8906/heroku-complementos

Capítulo 7: Heroku node.js Hola Mundo

Observaciones

iniciar sesión

`heroku login`

crear aplicación

`heroku create` o `heroku create your_name`

clonar el ejemplo

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

visita la aplicación en tu navegador

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

Opcional prueba local:

```
heroku local web
```

cheque: lolhost: 5000

Entonces, ¿qué es diferente a una aplicación normal node.js? paquete.json

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

index.js

```
process.env.PORT
```

Puerto local: 5000. Heroku lo asignará al puerto 80 en su url de aplicación.

Examples

Heroku node.js hola mundo

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);
```

paquete.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"
  }
}
```

Lea Heroku node.js Hola Mundo en línea: https://riptutorial.com/es/heroku/topic/9897/heroku-node-js-hola-mundo

Capítulo 8: Heroku Postgres

Examples

Cómo restablecer la base de datos Postgres en Heroku

Pasos para restablecer la base de datos en Heroku:

1. Descarte la base de datos, cuando se usa SHARED_DATABASE_URL:

```
heroku pg:reset DATABASE
```

2. Recrear la base de datos sin nada en ella:

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

3. Rellene la base de datos con sus datos de semilla:

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

Los pasos 2 y 3 se pueden combinar en un solo comando ejecutando esto:

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

Cómo copiar la base de datos heroku a la base de datos local

Pasos para copiar la base de datos heroku a la base de datos local:

1. Ejecutar el proceso de copia en el terminal:

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

2. Cambiar el propietario de la base de datos utilizando esta consulta:

```
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. Genere y ejecute la consulta para todas las tablas en su base de datos:

```
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;
```

Lea Heroku Postgres en línea: https://riptutorial.com/es/heroku/topic/6239/heroku-postgres

Capítulo 9: Límites de Heroku

Examples

Lista de todas las limitaciones en la plataforma Heroku.

1. Registros: de forma predeterminada, Heroku permite solo 1500 líneas de registros consolidados. Cuando se requieren más de 1500 líneas de registros, se debe usar los complementos proporcionados por Heroku.

2. Enrutador: la solicitud HTTP tiene un tiempo de espera de 30 segundos para la respuesta inicial y un tiempo de espera de 55 segundos a partir de entonces. Máximo de 1 MB de búfer permitido para la respuesta.

3. Dynos: *límites de memoria* Dyno en función del tipo elegido. Para las dinámicas gratuitas, las horas de sueño se imponen donde duerme después de 30 minutos de inactividad. Además, las cuentas verificadas vienen con un grupo mensual de 1000 horas de dinamización gratuitas, y las cuentas no verificadas reciben 550. Una aplicación puede tener hasta 100 días y un *tipo de proceso* no se puede escalar a más de 10 días. El tipo de dino libre puede tener un máximo de dos dines en ejecución concurrentes.

4. Config Vars: la clave de configuración y el par de valores están limitados a 32kb para una aplicación.

5. Construcción: los usuarios están limitados a 75 solicitudes a los repositorios de Heroku Git por hora, por aplicación, por usuario. El tamaño sin comprimir durante el pago no puede alcanzar más de 1 GB. El tamaño del slug está limitado a 300 MB y la duración de la compilación no puede exceder los 15 minutos.

6. Clips de datos: cada consulta puede ejecutarse hasta un máximo de 10 minutos y puede devolver un máximo de 100,000 filas.

7. Heroku Postgres: el tiempo de inactividad varía con los diferentes niveles de menos de 4 horas a 15 minutos por mes.

8. Límites de la API: el número máximo de llamadas a la API de Heroku está restringido a 2400 / hour.

9. Límites de membresía: para una cuenta empresarial, máximo de 500 miembros y para otros, se permiten 25 miembros.

10. Recuento de aplicaciones: un usuario verificado puede crear un máximo de 100 aplicaciones. Los usuarios no verificados están restringidos a 5 aplicaciones.

Lea Límites de Heroku en línea: https://riptutorial.com/es/heroku/topic/6190/limites-de-heroku

Capítulo 10: Línea de comando

Introducción

La Interfaz de línea de comandos de Heroku (CLI), anteriormente conocida como Heroku Toolbelt, es una herramienta para crear y administrar aplicaciones Heroku desde la línea de comandos / shell de varios sistemas operativos.

Sintaxis

- $ heroku --version
- $ heroku login
- $ heroku crear

Examples

Descargar e instalar

OS X

Descarga y ejecuta el instalador de OS X.

Windows

Descargue y ejecute el instalador de Windows de 32 bits y 64 bits .

Debian / Ubuntu

Ejecute lo siguiente para agregar nuestro repositorio apt e instalar 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
```

Versión independiente

Descargue el archivo tar y extráigalo para que pueda acceder al binario desde su PATH. Por ejemplo:

```
$ 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 tu instalación

Para verificar su instalación de CLI use el comando `heroku --version` .

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

Empezando

Se le pedirá que ingrese sus credenciales de Heroku la primera vez que ejecute un comando; después de la primera vez, su dirección de correo electrónico y un token de API se guardarán en `~/.netrc` para su uso futuro.

En general, es una buena idea iniciar sesión y agregar su clave pública inmediatamente después de instalar el CLI de Heroku para que pueda usar git para empujar o clonar los repositorios de aplicaciones de Heroku:

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

Ya estás listo para crear tu primera aplicación 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
```

Lea Línea de comando en línea: https://riptutorial.com/es/heroku/topic/8324/linea-de-comando

Capítulo 11: Oleoductos

Sintaxis

- Heroku pipelines: <instalar | crear | promover> ...

Observaciones

Una canalización es un grupo de aplicaciones de Heroku que comparten el mismo código base. Las aplicaciones en una tubería se agrupan en etapas de "revisión", "desarrollo", "organización" y "producción" que representan diferentes pasos de implementación en un flujo de trabajo de entrega continua.

Examples

Tuberías a través del CLI

Instalacion de tuberia

Una vez que Heroku Toolbelt está instalado, también requiere el complemento Pipelines .

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

Creando tuberías

Debe comenzar con una aplicación para agregar a la canalización, aunque no tiene que ser para una etapa en particular. Si no especifica `--stage STAGE` , el CLI adivinará la etapa apropiada, pero también le permitirá anular el valor predeterminado. El nombre de la canalización también se adivinará a partir del nombre de la aplicación, pero se puede anular agregando el `NAME` en la línea de comandos o ingresando un nombre diferente cuando se le solicite.

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

Promoviendo

Las aplicaciones de destino serán determinadas automáticamente por la etapa posterior

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

También es posible promocionar a una aplicación específica (o conjunto de aplicaciones)

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

Comando de Ayuda

Una lista completa de comandos de tuberías con detalles de uso está disponible en la consola

```
heroku help pipelines
```

Lea Oleoductos en línea: https://riptutorial.com/es/heroku/topic/2389/oleoductos

Capítulo 12: Troncos

Sintaxis

- $ heroku registros
- $ heroku registros -n 200
- $ heroku logs --tail
- $ heroku registros - no enrutador
- $ heroku logs --source app
- $ heroku logs --source app - no trabajador
- $ heroku logs --source app --tail

Examples

Tipos de registros

Heroku agrega tres categorías de registros para su aplicación:

- **Registros de la aplicación** - Salida de su aplicación. Esto incluirá los registros generados desde su aplicación, servidor de aplicaciones y bibliotecas. (Filtro: `--source app`)
- **Registros del sistema** : mensajes sobre acciones tomadas por la infraestructura de la plataforma Heroku en nombre de su aplicación, como: reiniciar un proceso bloqueado, suspender o activar un dinamómetro web o mostrar una página de error debido a un problema en su aplicación. (Filtro: - `--source heroku`)
- **Registros de la API** : mensajes sobre las acciones administrativas tomadas por usted y otros desarrolladores que trabajan en su aplicación, como: implementar un nuevo código, escalar la formación del proceso o alternar el modo de mantenimiento. (Filtro: `--source heroku --dyno api`)

Formato de registro

Cada línea de registro tiene el siguiente formato:

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

- **Marca de tiempo** : la fecha y la hora registradas en el momento en que la línea de registro fue producida por el dinamómetro o componente. La marca de tiempo está en el formato especificado por RFC5424 e incluye microsegundos de precisión.
- **Fuente** : todos los dinamizadores de su aplicación (dinámicos web, trabajadores de fondo, cron) tienen la fuente, la `app` . Todos los componentes del sistema de Heroku (enrutador HTTP, administrador de dinamómetro) tienen la fuente, `heroku` .

- **Dyno** : el nombre del dino o componente que escribió la línea de registro. Por ejemplo, el trabajador n. ° 3 aparece como `worker.3` y el enrutador HTTP de Heroku aparece como `router` .
- **Mensaje** - El contenido de la línea de registro. Las líneas generadas por dynos que exceden los 10000 bytes se dividen en segmentos de 10000 bytes sin nuevas líneas adicionales. Cada fragmento se envía como una línea de registro independiente.

Ver los registros

Para recuperar sus registros, use el comando `heroku logs` .

```
$ heroku logs
```

El comando de registros recupera 100 líneas de registro de forma predeterminada. Puede especificar el número de líneas de registro para recuperar (hasta un máximo de 1,500 líneas) utilizando la `--num` (o `-n`).

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

Cola en tiempo real

Al igual que `tail -f` , tail en tiempo real muestra los registros recientes y deja la sesión abierta para que se puedan transmitir los registros en tiempo real. Al ver un flujo en vivo de registros de su aplicación, puede obtener información sobre el comportamiento de su aplicación en vivo y Depurar problemas actuales. Puedes `--tail` tus registros usando `--tail` (o `-t`).

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

Cuando haya terminado, presione Ctrl + C para volver a la solicitud.

Filtrado de registros

Si solo desea obtener registros con una determinada fuente, un determinado dinamómetro o ambos, puede utilizar los argumentos de filtrado `--source` (o `-s`) y `--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 ...
```

También puede combinar los interruptores de filtrado con `--tail` para obtener un flujo en tiempo real de salida filtrada.

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

Lea Troncos en línea: https://riptutorial.com/es/heroku/topic/8327/troncos

Creditos

S. No	Capítulos	Contributors
1	Empezando con heroku	Community, rdegges, thejonanshow
2	Buildpack	Thamilan
3	Códigos de error de Heroku	Sender
4	Dependencias	Thamilan
5	Despliegue	Sender
6	Heroku complementos	jophab
7	Heroku node.js Hola Mundo	Johan Hoeksma
8	Heroku Postgres	Denis Savchuk, Hardik Kanjariya ツ, Thamilan
9	Límites de Heroku	autoboxer, Thamilan
10	Línea de comando	Sender
11	Oleoductos	Thamilan
12	Troncos	Sender