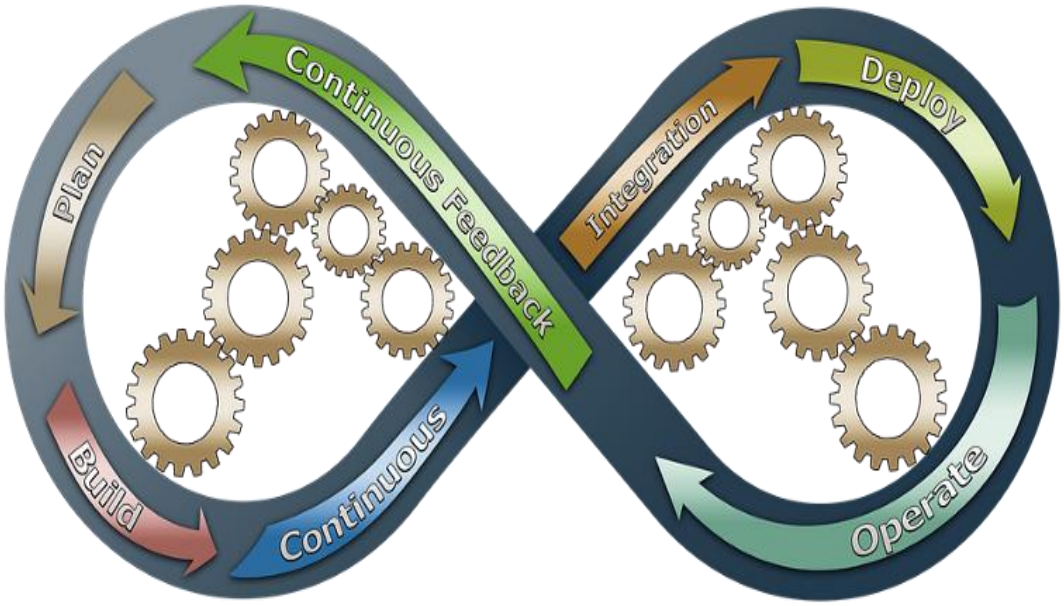
Configuración de un proceso de CI / CD en GitHub con Travis CI



Has creado algo asombroso. Has publicado en GitHub. La gente está descargando, usando, bifurcando y contribuyendo. La comunidad está encantada. Pero, ¿estás listo para el exceso de solicitudes de extracción?

Integración continua / Entrega continua (" **CI / CD** "), dos términos tan frecuentemente mencionados juntos que se han fusionado en un concepto, se refiere a la automatización de tareas repetitivas como probar, construir e implementar software. Con el tiempo, se ha convertido en una herramienta invaluable para mantener un equipo productivo y funcional.

¿Como funciona?

Integración continua

Un proceso de CI es un proceso en el que el software es desarrollado por múltiples fuentes y se integra automáticamente mediante un procedimiento establecido. El flujo podría ser algo como esto:

1. Empujar a Git;
2. Se desencadena un proceso;
3. Se extrae la rama relevante, se crea la aplicación y se ejecutan las pruebas;
4. Los resultados de este proceso se envían a quien corresponda.

Si todas las pruebas pasan, significa que el código puede fusionarse de manera segura (bueno, de forma segura en las áreas que cubren las pruebas) en la rama principal.

Despliegue continuo

Un proceso de CD es un proceso que generalmente sigue al proceso de fusión. Toma la versión recién fusionada y generalmente hace lo siguiente:

1. Ejecuta pruebas (generalmente más largas y más de extremo a extremo ("E2E") - pruebas de tipo que las ejecutadas durante los procesos de CI);
2. Crea un artefacto y lo almacena;
3. Se implementa en producción (o preproducción);
4. Ejecuta pruebas de posproducción E2E. Por lo general, configuraría un desencadenante de reversión utilizando la carpeta de artefactos prefabricados en caso de fallas durante la postproducción.

¿Cómo puedes crear tal proceso?

Muchas herramientas pueden ayudarlo a crear un proceso de CI / CD. Una herramienta brillante y fácil de usar es Travis CI . En las secciones que siguen, le mostraré paso a paso cómo integrar un proyecto existente con Travis CI y configurar un proceso simple de CI / CD.

El proyecto

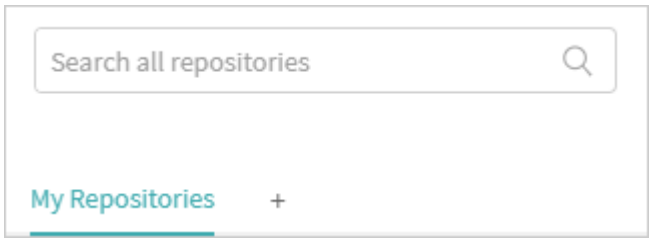
El proyecto es una biblioteca de componentes web que actualmente solo cuenta con una ventana modal. Aquí está el enlace de GitHub .

Actualmente tiene un flujo de prueba y compilación (a través de `npm run test:prod` y `npm run build`). Tiene una demostración que se implementa manualmente en Firebase Hosting. Puedes ver la demo aquí .

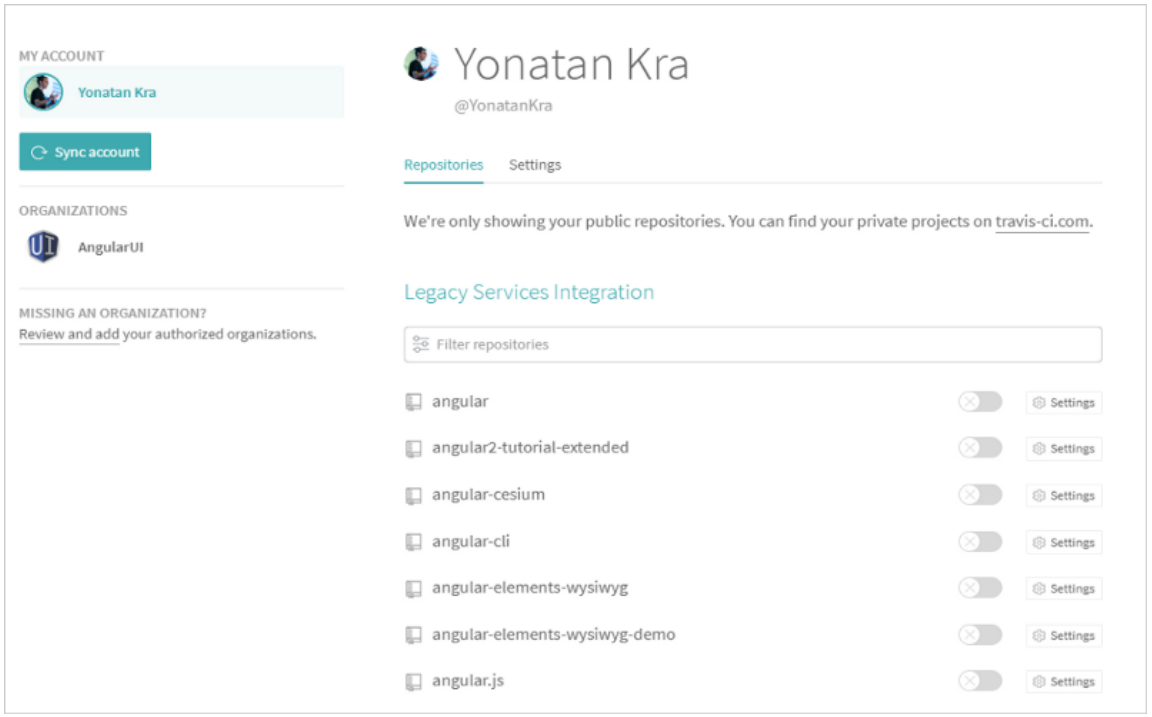
Conecte Travis CI a su proyecto

La mayoría de las herramientas de CI se integran perfectamente con los servicios de Git, especialmente GitHub. Vaya a travis-ci.com y regístrese. Sugiero iniciar sesión con su cuenta de GitHub: facilitará las cosas en las etapas posteriores.

Dentro de Travis CI, hay una barra de búsqueda en el lado izquierdo de la pantalla; Haga clic en el signo + debajo de él.



Si conectó su cuenta de GitHub, debería ver una lista de sus repositorios existentes, desde la cual puede agregar:



Una lista de tus proyectos. Bueno ... el mio. Supongo que los usuarios de React dejarían este artículo ahora;)

Busque el proyecto relevante (en mi caso, web-components-ui-elements) y haga clic en el interruptor cercano.

El rojo es el interruptor, el azul es la configuración :)

Después de habilitarlo, haga clic en el botón de configuración (indicado por una flecha azul arriba). Ahora debería ver varias configuraciones. Le animo a leer sobre cada opción en la documentación de Travis CI y elegir la configuración que más le convenga. Utilizaremos la configuración predeterminada para este tutorial.

Ahora que el backend está configurado, conectemos el proyecto desde el final del código a Travis CI.

El archivo de configuración

La mayoría de las herramientas de CI esperan que exista un archivo de configuración en el proyecto. Travis CI espera
que *exista* un archivo *.travis.yml* en la raíz del proyecto.

Veamos cómo construimos un archivo en Travis CI:

```yaml
language: node_js
node_js:
  - "8"
script:
  - echo 'Build starts!!'
  - echo 'Installing Deps!'
  - yarn
  - echo 'Testing!'
  - npm run test:prod
```

Repasemos las partes:

1. **idioma** : utilizamos JavaScript, pero Travis lo llama "node_js" (bien por mí);
2. **node_js** : bastante sencillo también; solo indicamos la versión que queremos usar;
3. **script** : es solo un conjunto de comandos bash para ejecutar. En nuestro caso, hacemos eco de algunos comentarios útiles, instalamos nuestras dependencias (usando Yarn) y luego ejecutamos nuestras pruebas.

El proceso de CI

Como se mencionó anteriormente, el proceso de CI se asegura principalmente de que no hayamos roto nada y que estemos listos para fusionarnos / integrarnos. Entonces esta fase incluirá el siguiente proceso:`Clone -> yarn/npm install -> test`

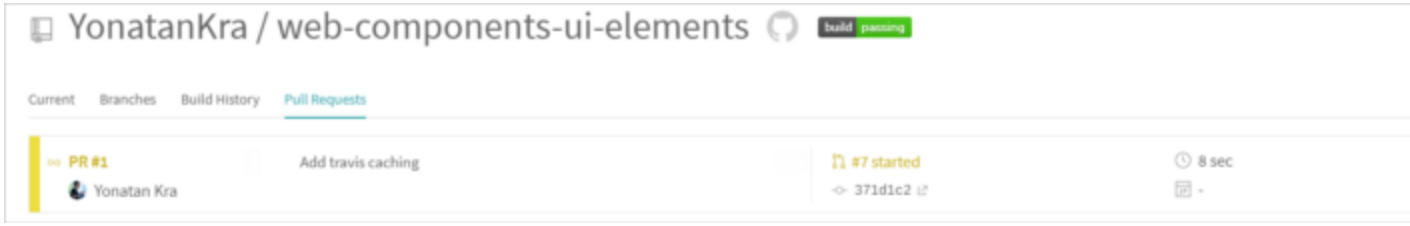
También queremos acelerar nuestras compilaciones, por lo que le diremos a Travis que guarde en caché nuestra carpeta npm o Yarn (o cualquier otro administrador de paquetes que esté utilizando; Travis admite muchos).

Estamos usando hilo; digamos a Travis que guarde en caché la carpeta Yarn:

```
language: node_js
node_js:
  - "8"
cache: yarn
script:
  - echo 'Build starts!!'
  - echo 'Installing Deps!'
  - yarn
  - echo 'Testing!'
  - npm run test:prod
```

Puede leer más sobre el almacenamiento en caché aquí .

En el momento en que presione un cambio, Travis activará el proceso de construcción.

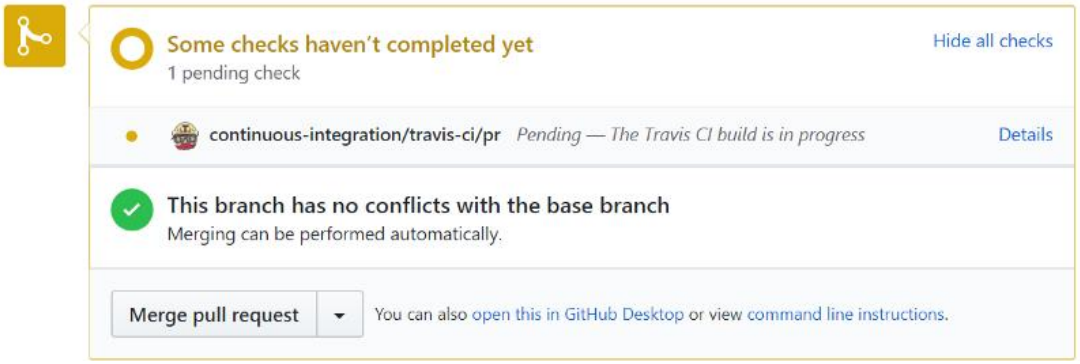


Proceso iniciado: Yarn, caché y pruebas ahora se están ejecutando.

Haga clic aquí para ver los resultados.

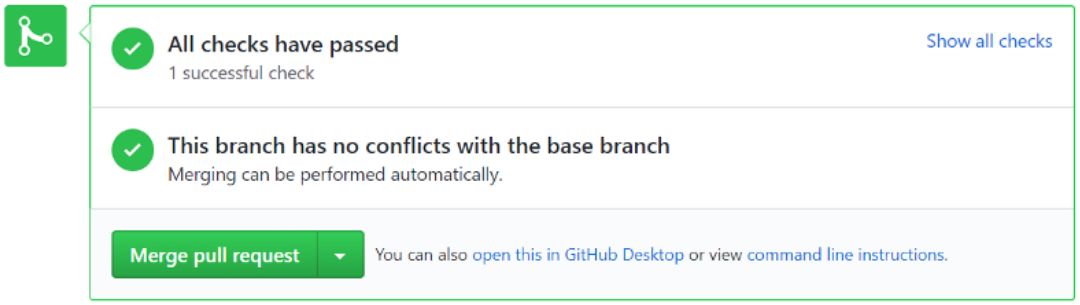
En la página de resultados , puede ver el resumen (el proceso pasó o falló) en la parte superior. A continuación, puede ver el registro del trabajo y ver nuestros ecos y toda la salida del proceso de CI.

En este caso, presioné a una solicitud de extracción. En GitHub, puede ver los resultados en la página Solicitud de extracción (" **PR** ") de esta manera:



Las pruebas se están ejecutando.

Una vez que pase el proceso, verá el cambio en GitHub:



Yahoo! ¡Podemos fusionarnos para dominar ahora!

Las pruebas son solo un paso en el proceso de CI. Puede agregar más líneas a la `script`etapa, incluidos procesos como linting, pruebas E2E, pruebas de rendimiento y cualquier métrica que se le ocurra para probar el nuevo código antes de fusionarlo con la base de código principal.

El proceso de CD

Hasta ahora, hemos visto cómo usar CI para sentirse más seguro antes de combinar cambios en nuestra rama principal. Este es el papel de CI.

El proceso de cargar la nueva y maravillosa característica que se acaba de fusionar sigue siendo manual. Necesitamos escribir manualmente en la consola de nuestra máquina local para implementar. Es hora de automatizar el proceso.

El proceso de CD aquí será bastante simple: solo suba el dist a Firebase.

Si ha seguido hasta ahora, es posible que pueda deducir que solo usaremos las mismas directivas de línea de comando que usamos en nuestra propia computadora, solo que le diremos a la VM remota que lo haga por nosotros:

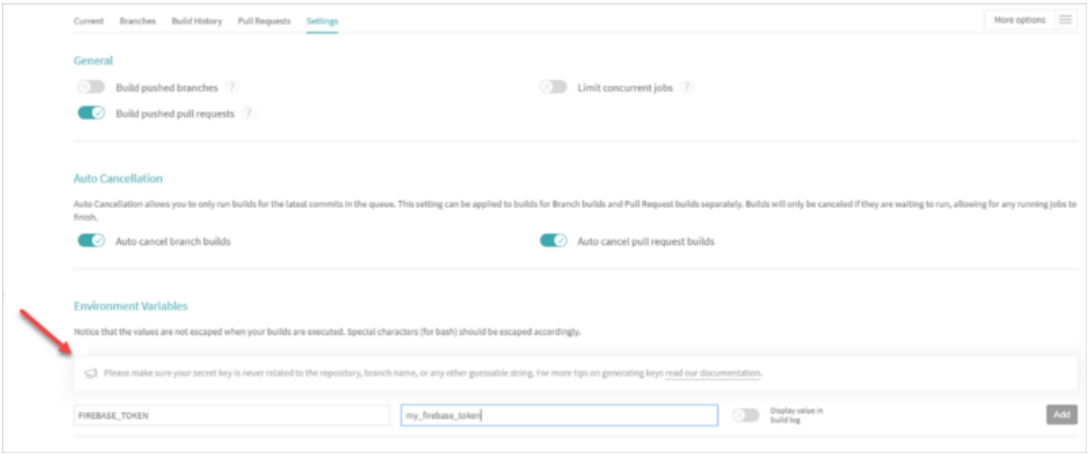
```
install:
  - npm install -g firebase-tools
after_success:
  - firebase deploy --token "$FIREBASE_TOKEN"
```

Estas líneas le dicen a Travis que instale `firebase-tools`globalmente y, después de una compilación exitosa, implemente en Firebase.

Tenga en cuenta que he agregado dos etapas más al proceso: 1) instalar y 2) after_success. No es obligatorio usarlos; podría agregar más líneas de script en su lugar.

Observe el `$FIREBASE_TOKEN`en el código yml anterior. Esta es una variable de entorno. Podemos obtener el token usando `firebase login:ci`. Este comando abre un navegador que lo guía a través del proceso de creación de un token.

Luego configuramos el token como una variable de entorno en la configuración del proyecto que vimos antes en travis-ci:



La flecha roja marca el lugar.

De esta manera, puede agregar cualquier variable de entorno que desee. También hay una manera de definir variables de entorno a través

del archivo yml, pero, por simplicidad, lo omitiré. Si estás interesado, lee más aquí.

Tenga en cuenta que he configurado Firebase hosting para el proyecto (puede ver el archivo firebase.json aquí).

El archivo *yml* completo sería:

```
language: node_js
node_js:
  - "8"
cache: yarn
script:
  - echo 'Build starts!!'
  - echo 'Installing Deps!'
  - yarn
  - echo 'Testing!'
  - yarn test:prod
install:
  - npm install -g firebase-tools
after_success:
  - yarn build
  - firebase deploy --token "$FIREBASE_TOKEN"
```

Ahora, cada vez que empujamos una rama que pasa las pruebas, tenemos la nueva versión en producción.

Si desea utilizar una opción de implementación integrada, asegúrese de consultar todas [las opciones de implementación que Travis ofrece aquí .

Espera ... ¿no te olvidaste de unirte?

Me tienes allí ... nuestro proceso ahora hace lo siguiente:

1. Clona la rama PR;
2. Realiza la instalación del hilo;
3. Ejecuta las pruebas;
4. Instala herramientas de Firebase;
5. Si las pruebas son exitosas, compila el proyecto y lo implementa en el alojamiento de Firebase.

Esto plantea un problema: mientras implementamos la sucursal en producción, nuestro maestro sigue siendo el mismo. Esto podría causar inconsistencias en muchos casos.

Lo que me gustaría hacer es fusionarme en master y luego implementar. Entonces el paso 5 se convertiría en el siguiente:

1. Después de la fusión manual (p. Ej., Después de la revisión del código), active la implementación

Introducir trabajos

Una tubería en Travis se llama "trabajo". Prácticamente se traduce en un proceso que se ejecuta en una máquina virtual (" **VM** "). Travis nos permite crear varios trabajos (es decir, varias máquinas virtuales que pueden ejecutarse en paralelo) y definir ciertas etapas para cada trabajo. Esto nos ayuda a organizar nuestros procesos y usar condicionales para manipularlos fácilmente.

Aquí está el código resultante:

```
language: node_js
node_js:
  - "8"
cache: yarn
jobs:
  include:
    - stage: test
      script:
        - echo 'Deploy!!'
        - echo 'Installing Deps!'
        - yarn
        - echo 'Testing!'
        - yarn test:prod
    - stage: deploy
      script:
        - npm install -g firebase-tools
        - firebase deploy --token "$FIREBASE_TOKEN"
```

Un poco más grande, pero la idea sigue siendo la misma.

Primero, creo la `jobs`propiedad que describe las dos etapas que hemos visto antes:

1. prueba: ejecuta el hilo y las pruebas;
2. deploy: ejecuta la instalación, la compilación y la implementación de Firebase en Firebase.

En este punto, por cada impulso a cualquier rama, se ejecutará todo el proceso. Nos gustaría que las pruebas se ejecuten para los RP y el despliegue para push to master (por ejemplo, fusionar).

Travis nos permite agregar fácilmente acondicionamiento a cada proceso, trabajo o etapa. Agreguemos algunas condiciones:

```
language: node_js
node_js:
  - "8"
cache: yarn
jobs:
  include:
    - stage: test
      script:
        - echo 'Deploy!!'
        - echo 'Installing Deps!'
        - yarn
        - echo 'Testing!'
        - yarn test:prod
    - stage: deploy
      script:
        - npm install -g firebase-tools
        - yarn build
        - firebase deploy --token "$FIREBASE_TOKEN"
stages:
  - name: test
    # require the type to be a PR
    if: type = pull_request
  - name: deploy
    # require the type to be push to master
    if: type = push AND branch = master
```

La nueva propiedad que se muestra en el código anterior - `stages` nos ayuda a definir fácilmente las condiciones para ejecutar cada etapa de nuestra tubería. Defino la prueba que se ejecutará si el tipo de disparador es un RP y la implementación si el disparador fue un empujón para dominar.

Si solo crea una nueva rama y empuja a GitHub, nada sucederá en Travis, porque no se aplica a ninguna condición. Puede ver qué sucede con esas solicitudes de compilación en la página de solicitudes de su repositorio .

El envío de un RP para la sucursal satisface la regla de RP para la etapa de prueba y correría las pruebas y nos daría este resultado .

Observe que solo ejecuta la etapa de pruebas. Hoorah!

Ahora vamos a fusionarnos desde la página de relaciones públicas. Para hacerlo, simplemente haga clic en el botón de fusión y siga para aprobar. Después de fusionar la rama para dominar, veremos esto.

¡El proceso de implementación de Firebase para la nueva función se realizó sin problemas!

¡Y eso es todo lo que ella escribió! Tenemos nuestra propia tubería de CI / CD.

Puede clonar el proyecto y probar todo este bien por su cuenta. Siéntase libre de dejar un comentario si tiene algún problema. También puede contactarme a través de Twitter (@yonatankra) o preguntar en el foro de Travis CI . Si bien no soy un experto, prometo ayudar lo mejor que pueda.

Resumen

En este artículo, aprendimos que CI / CD es un proceso de automatización que le permite ahorrar tiempo y le permite escalar. Por lo tanto, le ahorra mucho tiempo y dinero, y aumenta la productividad y el valor que puede proporcionar a sus usuarios finales.

También aprendimos cómo integrar un proceso de CI / CD en un proyecto en GitHub. Puede agregar muchos otros pasos a su canalización de CI / CD, por ejemplo, pruebas de posproducción y los pasos de reversión. No los cubrimos en este breve tutorial de

introducción, pero sugiero implementarlos solo para familiarizarse con las cosas.

Otros pasos podrían incluir limitar las capacidades de fusión, agregar diferentes tuberías para algunos escenarios, agregar una rama de control de calidad si aún no está seguro de sus pruebas previas y posteriores a la producción, y mucho más.

La conclusión es esta: no importa qué herramienta elija, puede automatizar fácilmente cada proceso. Al automatizar procesos simples pero que requieren mucho tiempo, puede agregar más colaboradores / miembros del equipo a su proyecto de manera más segura mientras acelera el proceso de desarrollo y brinda más valor a los usuarios finales de su producto (ya sea una biblioteca de código abierto o una aplicación).