LORENZO SAVINI
Developer

Introduzione a Buildkite

Develer webinar

28/04/2021

develer 20

OBIETTIVI DEL WEBINAR

- Chiarire il concetto di continuous integration
- Comprendere Buildkite e i suoi concetti principali
- Prendere familiarità con le pipeline e i vari strumenti a disposizione

Programma del webinar

- Overview di Buildkite
  - Architettura
  - Feature principali
- Come si configura il nostro primo progetto
- Pipeline e step
- Agent plugins
- Meta-data e Artifacts
- Pipeline dinamiche
- Conditionals e dependencies
- Pipeline schedulate
- Bonus

Repository con esempi

https://github.com/savo92/webinar-buildkite

Continuous Integration

Metodologia dello sviluppo del software in cui si allinea spesso la mainline del progetto dagli ambienti di lavoro degli sviluppatori (aka: si fa merge spesso delle branch su main/master).

Buildkite: le basi

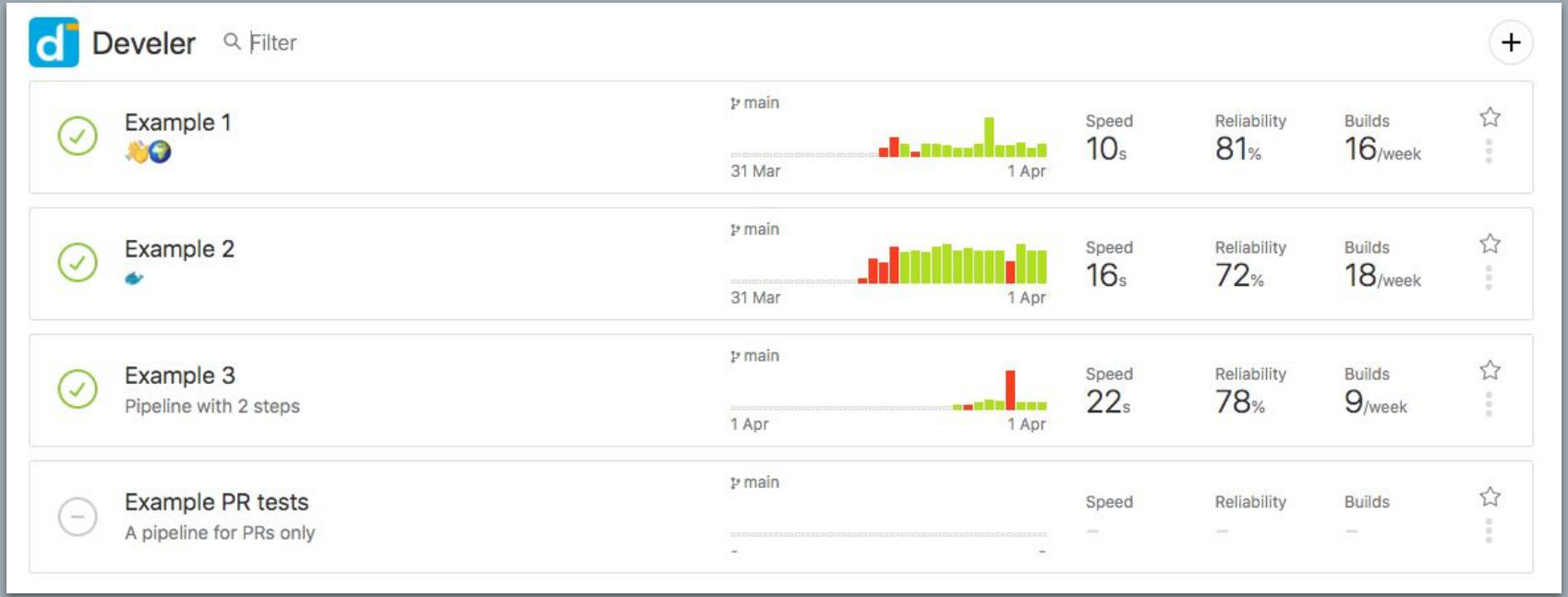
Buildkite

Buildkite è una piattaforma per continuous integration che permette di creare pipeline per i vostri progetti, che verranno poi eseguite da agent installati sulla vostra infrastruttura.

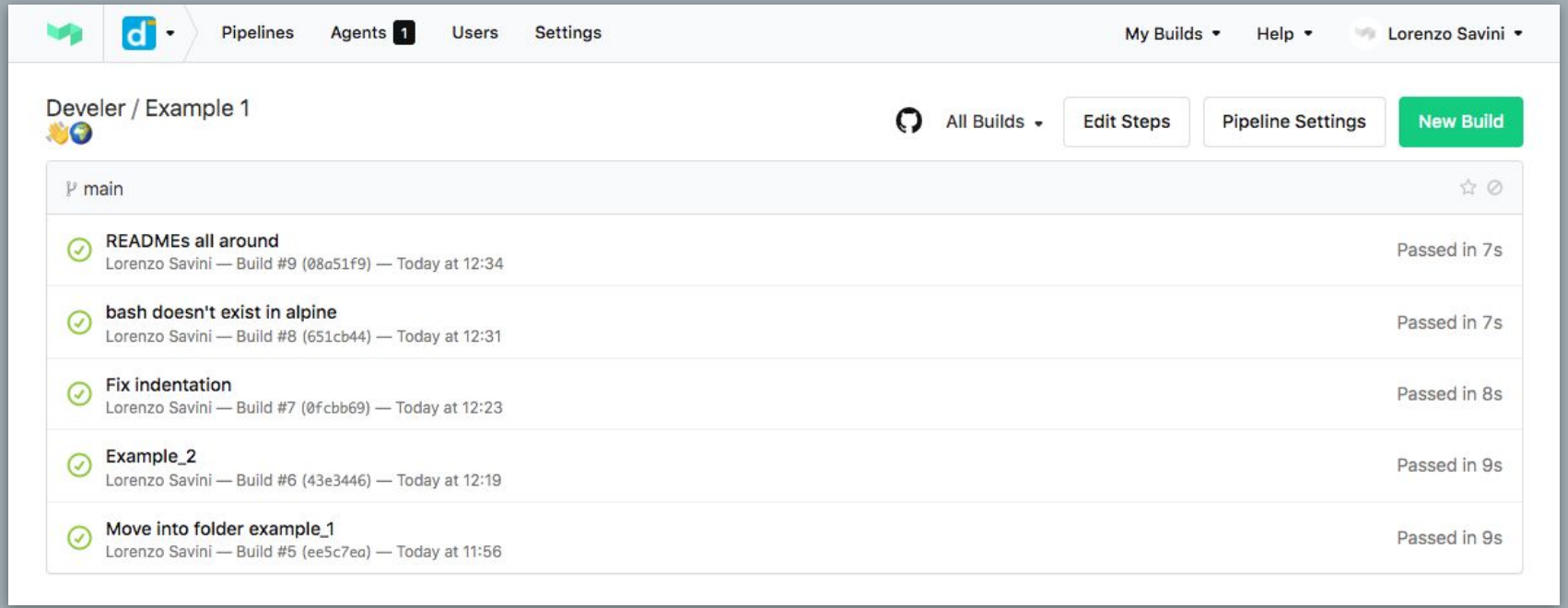
Concetti base - Buildkite: le basi

- **Pipeline**: insieme di **step** che vengono eseguiti da Buildkite.
- **Build**: una specifica esecuzione di una pipeline.
- **Agent**: build runner che gira sulla vostra infrastruttura (VPS, server, Docker) connesso con Buildkite, ed esegue le vostre build.

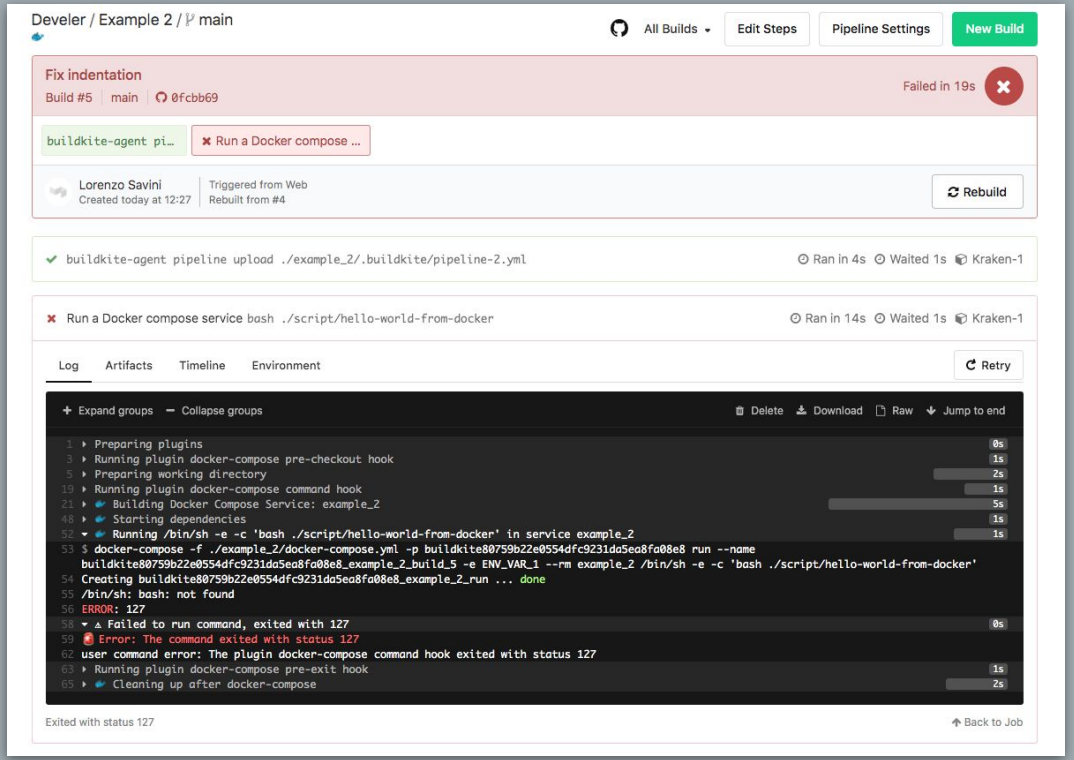
Homepage - Buildkite: le basi



UI di una pipeline - Buildkite: le basi



UI di una build - Buildkite: le basi



Integrazione con repository Git - Buildkite: le basi

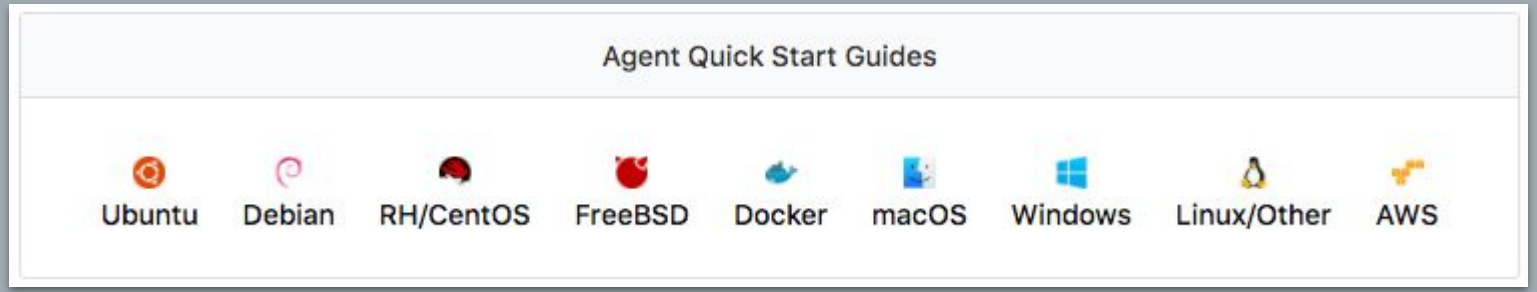
- GitHub
- GitHub Enterprise Server

- GitLab
- GitLab Community / Enterprise Edition

- Bitbucket
- Bitbucket Server

Qualsiasi Git server (niente integrazione assistita però)

Agent - Buildkite: le basi



Agent su MacOS - Buildkite: le basi

```
$ brew install buildkite/buildkite/buildkite-agent
$ sed -i '' "s/xxx/<YOUR AGENT TOKEN>/g" "$(brew
--prefix)"/etc/buildkite-agent/buildkite-agent.cfg
```

Pipeline e step

- Creazione di una pipeline
- Tipi di step

Dichiarare una pipeline - Pipeline e step

Una pipeline si dichiara in YAML:

- direttamente dalla UI di Buildkite;
- con un file YAML nella vostra repository.

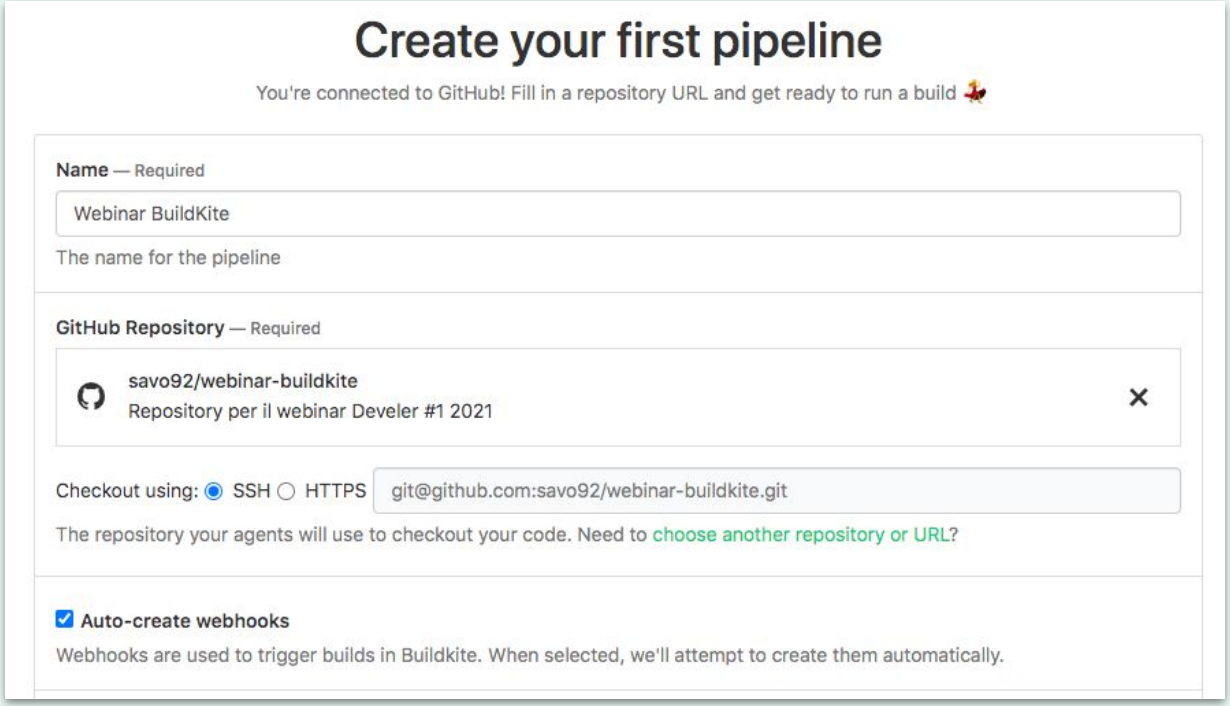
Dichiarare una pipeline in un file YAML - Pipeline e step

Se volete dichiarare una pipeline tramite un file YAML nella vostra repository, in alcuni casi dovrete comunque usare la UI per instruire Buildkite a caricare la configurazione.

Basta uno step con questo comando:

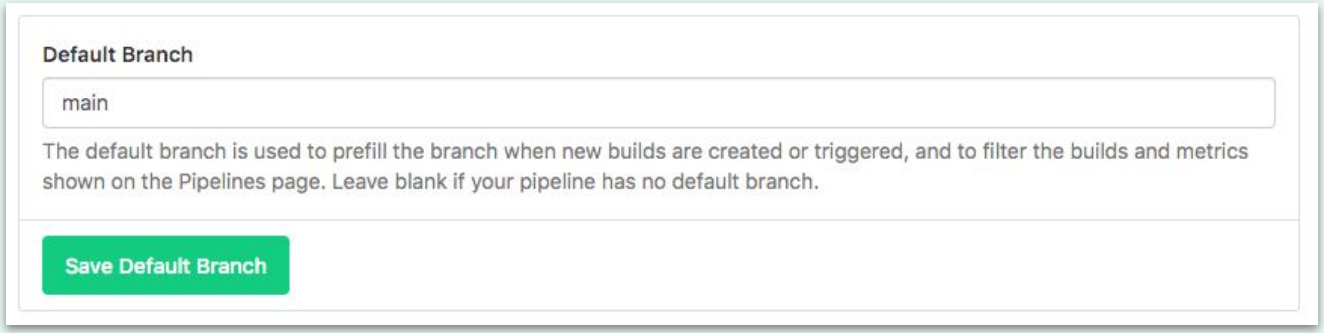
```
buildkite-agent pipeline upload ./example_1/.buildkite/pipeline-1.yml
```

Prima pipeline - Pipeline e step



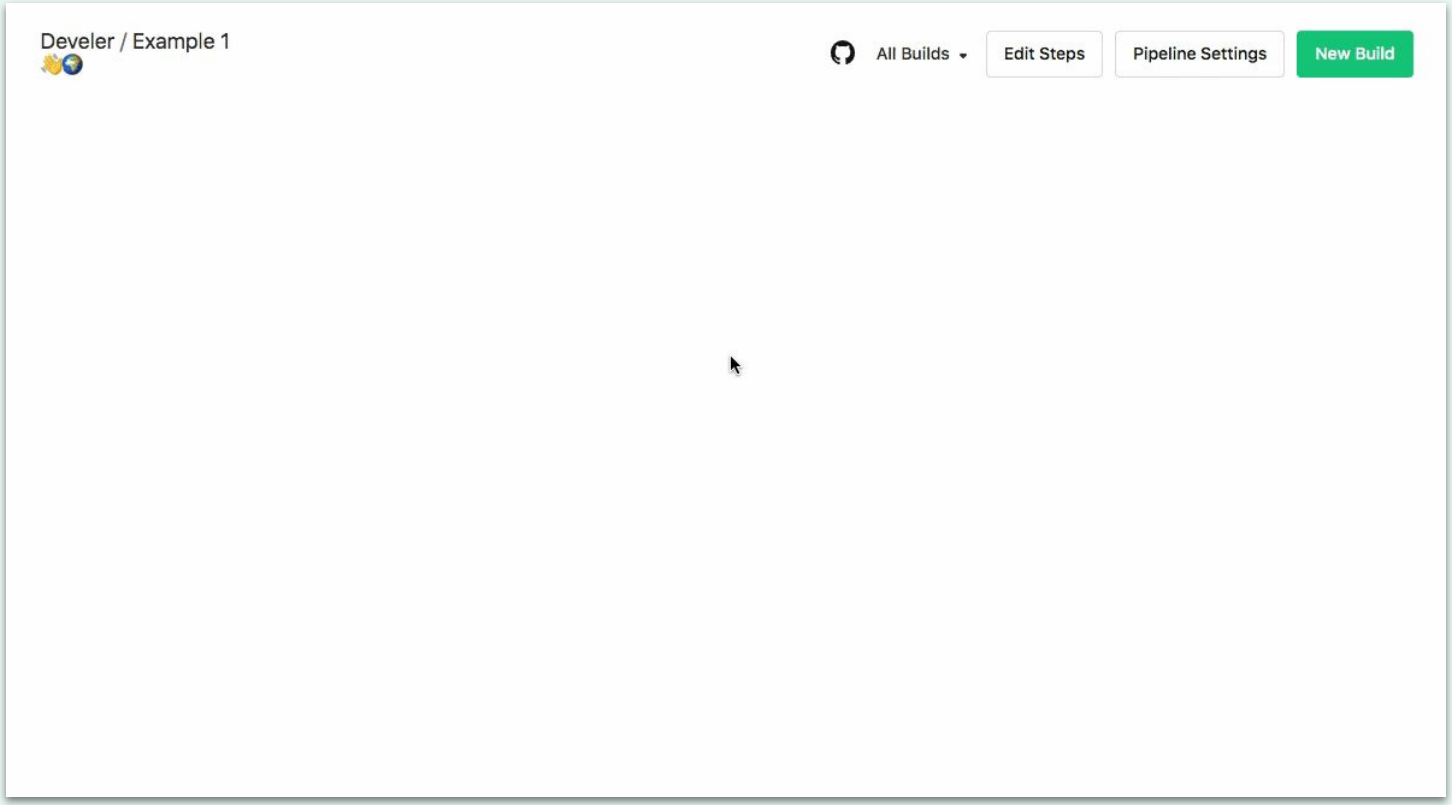
Prima di cominciare... - Pipeline e step

Adesso alcuni provider, come GitHub, creano repositories con “main” come default branch.
Buildkite di default usa ancora “master”, quindi dovrete configurare a dovere la vostra pipeline se necessario:



Esempio #1

Esempio 1: Prima pipeline - Pipeline e step



Un esempio di sintassi - Pipeline e step

```yaml
steps:
  - name: 'Hello world!'
    command: 'bash ./script/hello-world'

  - name: 'Another Hello world'
    command: 'bash ./script/hello-world'
```

Per dichiarare una pipeline è sufficiente creare un file YAML contenente una lista di steps, ciascuno con un commando ed altri metadati (tipo il nome dello step, per visualizzarlo nella UI di Buildkite).

Nota bene: in questo esempio i 2 step vengono eseguiti in **parallelo** su differenti agent, se disponibili.

Tipi di step

Buildkite permette di dichiarare 5 diversi tipi di step:

- Command step
- Wait step
- Block step
- Input step
- Trigger step

Command step - Tipi di step - Pipeline e step

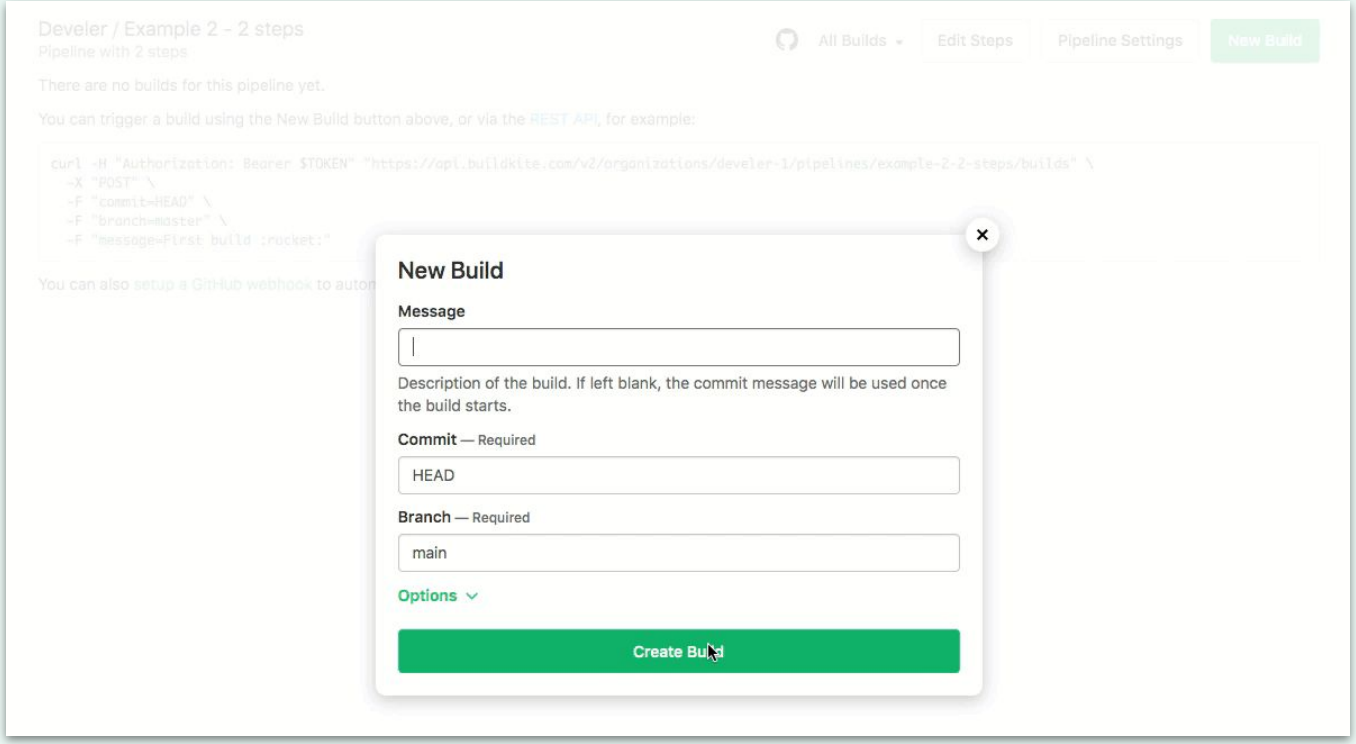
```yaml
steps:
 - command: "dosomething.sh"

 - commands:
   - "yarn install && yarn test"
   - "dosomethingelse.sh"
```

Un command step permette di eseguire uno o più comandi.

Nel secondo step, che prevede una lista di comandi, un eventuale fallimento di Yarn impedirebbe l'esecuzione del secondo comando `dosomethingelse.sh`.

Esempio #2

Esempio 2 - Pipeline e step



Wait step - Tipi di step - Pipeline e step

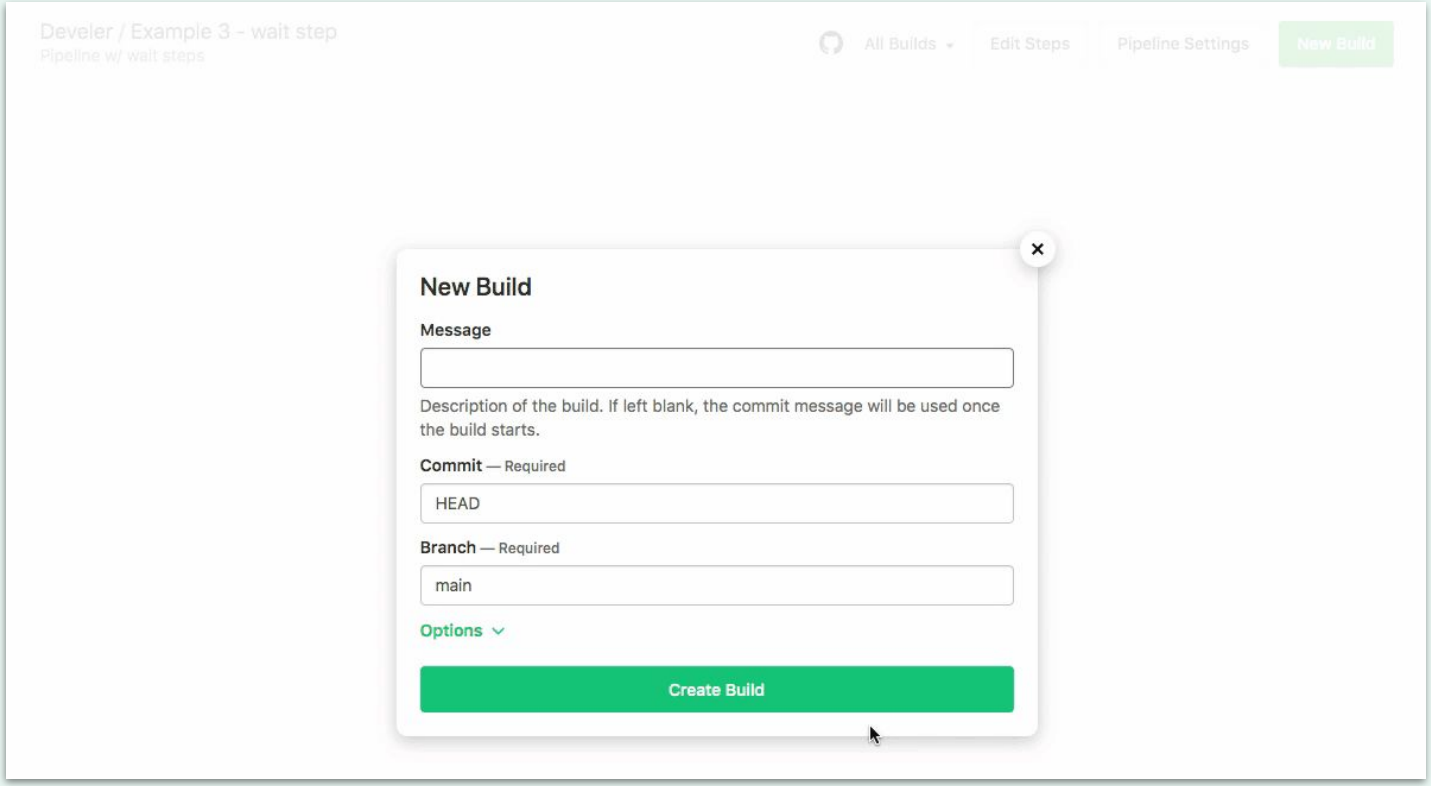
```yaml
steps:
  - command: "dosomething.sh"
  - wait
  - command: "dosomethingelse.sh"
```

Un wait step, di default, attende che tutti i comandi precedenti siano terminati con successo.
Permette quindi di impedire la parallelizzazione che BK applica di norma.

Buildkite permette anche di configurare il wait step in modo da continuare anche in caso di fallimento di uno step precedente.

Esempio #3

Esempio 3: Pipeline con wait step - Pipeline e step



Block step - Tipi di step - Pipeline e step

```yaml
steps:
  - command: "dosomething.sh"
  - block: "Go ahead!"
  - commands: "release.sh"
```

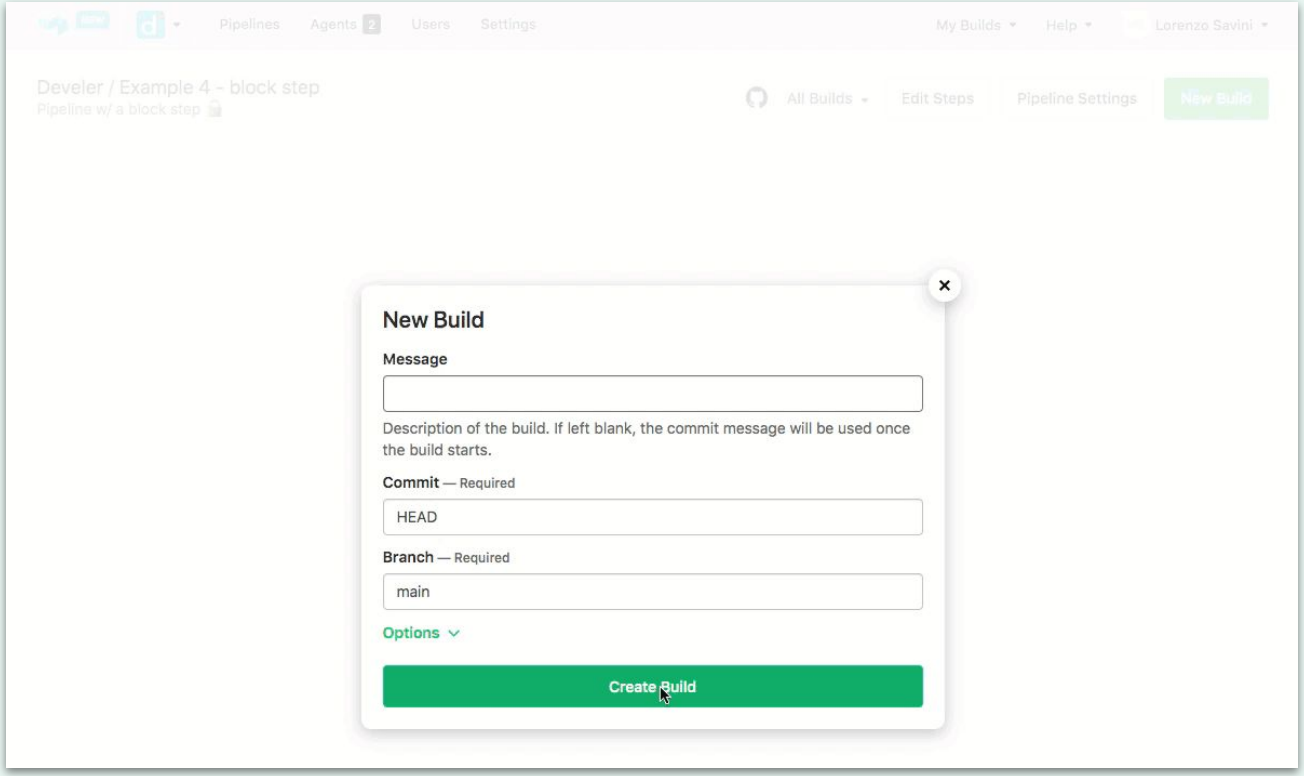
Un block step mette in pausa l'esecuzione di una build e aspetta di essere sbloccato da un'operazione manuale dalla UI o via API.

Questo step crea delle dipendenze implicite su gli step precedenti e successivi.

Può essere configurato anche per richiedere parametri.

Esempio #4

Esempio 4: Pipeline con block step - Pipeline e step



Input step - Tipi di step - Pipeline e step

```yaml
steps:
  - input: "Request Release"
```

Un input step richiede informazioni all'utente.

A differenza del block step, questo tipo **NON** crea dipendenze implicite su gli step precedenti e successivi. Questa configurazione deve essere esplicita (vedremo tra poco come si dichiarano step dependencies esplicite).

Trigger step - Tipi di step - Pipeline e step

```yaml
steps:
  - trigger: "another-pipeline-slug"
```

Il trigger step permette di creare una nuova build su un'altra pipeline.

Di default, l'esecuzione della nuova build metterà in attesa la build madre fino al completamento. La build madre verrà interrotta se la nuova build fallisce.
È però possibile anche avviare build affinché vengano eseguite in maniera asincrona.

Agent plugin

Cosa sono i plugin - Agent plugin

I plugin sono repository Git che permettono di estendere il comportamento di un command step utilizzando degli hook.

È possibile creare nuovi plugin oppure utilizzare quelli messi a disposizione da Buildkite o da contributor terzi, che si possono cercare qui https://buildkite.com/plugins.

I plugin non hanno bisogno di essere *installati* ma alcuni necessitano di un supporto degli agent (ad esempio: per i plugin legati a Docker, serve Docker attivo sull'agent).

Plugin più utilizzati - Agent plugin

Tra alcuni dei plugin più utilizzati:

- Docker: per eseguire i tuoi step all'interno di un container;
- Docker compose: utilizzare Docker compose per eseguire i tuoi step sempre all'interno di un container;
- Cache: per attivare una cache persistente fra gli step;

Ma esistono anche plugin per AWS (ECS, S3 , Lambda), K8S, Go, Terraform, keyring.

Utilizzare il plugin Docker-compose - Agent plugin

```
steps:
  - name: 'Run a Docker compose service'
    command: 'dosomething.sh'
    plugins:
        docker-compose#v3.3.0:
            run: example_2
            config:
                - ./docker-compose.yml
            env:
                - ENV_VAR_1
```

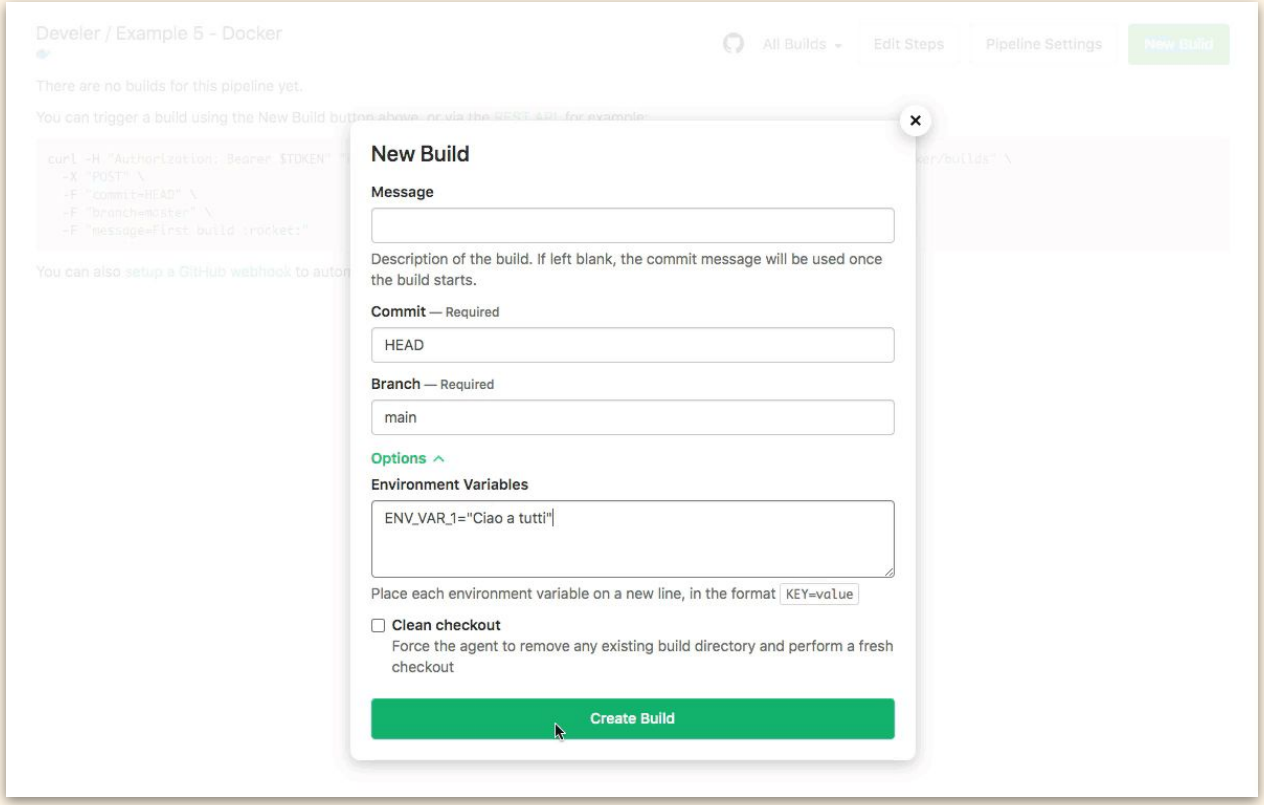
Vediamo un esempio di un command step eseguito all'interno di un Docker container creato tramite Docker compose.

In questo caso, BK eseguirà `dosomething.sh` all'interno del service `example_2`, creato a partire dal docker-compose.yml indicato.

Esempio 5: Command step con Docker-compose - Agent plugin

Esempio #5

Esempio 5: Command step con Docker-compose - Agent plugin



Meta-data e artifacts

Passare informazioni e artefatti tra step

Metadati - Meta-data e artifacts

Buildkite permette di salvare dati, della dimensione massima di 1 KB, per essere utilizzati fra step, che potrebbero anche girare su agent differenti.

Per esempio, si può salvare la tag di una build da rilasciare.

Tra lo step di set e quello di get ci deve essere una dipendenza o un wait step.

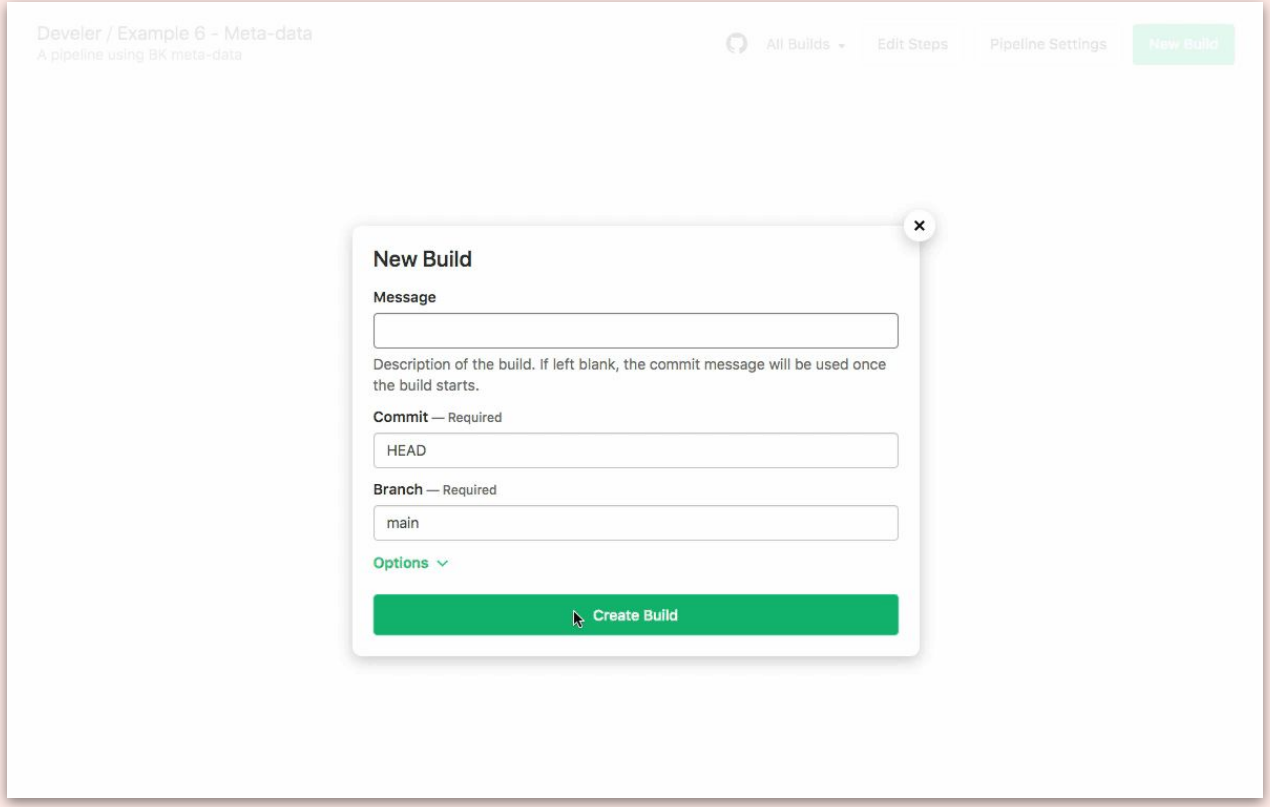
```
buildkite-agent meta-data set "release-name" "v1"

buildkite-agent meta-data get "release-name"
```

Metadati - Meta-data e artifacts

```yaml
steps:
  - block: "Request Release"
    fields:
      - text: "Code Name"
        key: "release-name"

  - name: 'Print the code name'
    command: 'buildkite-agent meta-data get "release-name"'
```

Esempio #6

Esempio 6 - Meta-data e artifacts



Artefatti - Meta-data e artifacts

Per tutte le altre risorse con dimensioni superiori ad 1 KB, è possibile utilizzare gli artifacts di Buildkite, per salvare e scaricare file in diversi step di una build.

Di default, Buildkite salva questi file in un loro bucket S3 ma è anche possibile utilizzare un proprio bucket S3, Google Cloud Storage o Artifactory.

Tra lo step di upload e quello di download ci deve essere una dipendenza o un wait step.

Upload di Artefatti tramite configurazione - Meta-data e artifacts

```yaml
steps:
 - command: "buildsomething.sh"
   artifact_paths:
     - "dist/**/*"
```

È sufficiente configurare l'opzione `artifact_paths` in uno step per istruire Buildkite a caricare i file indicati sullo storage.

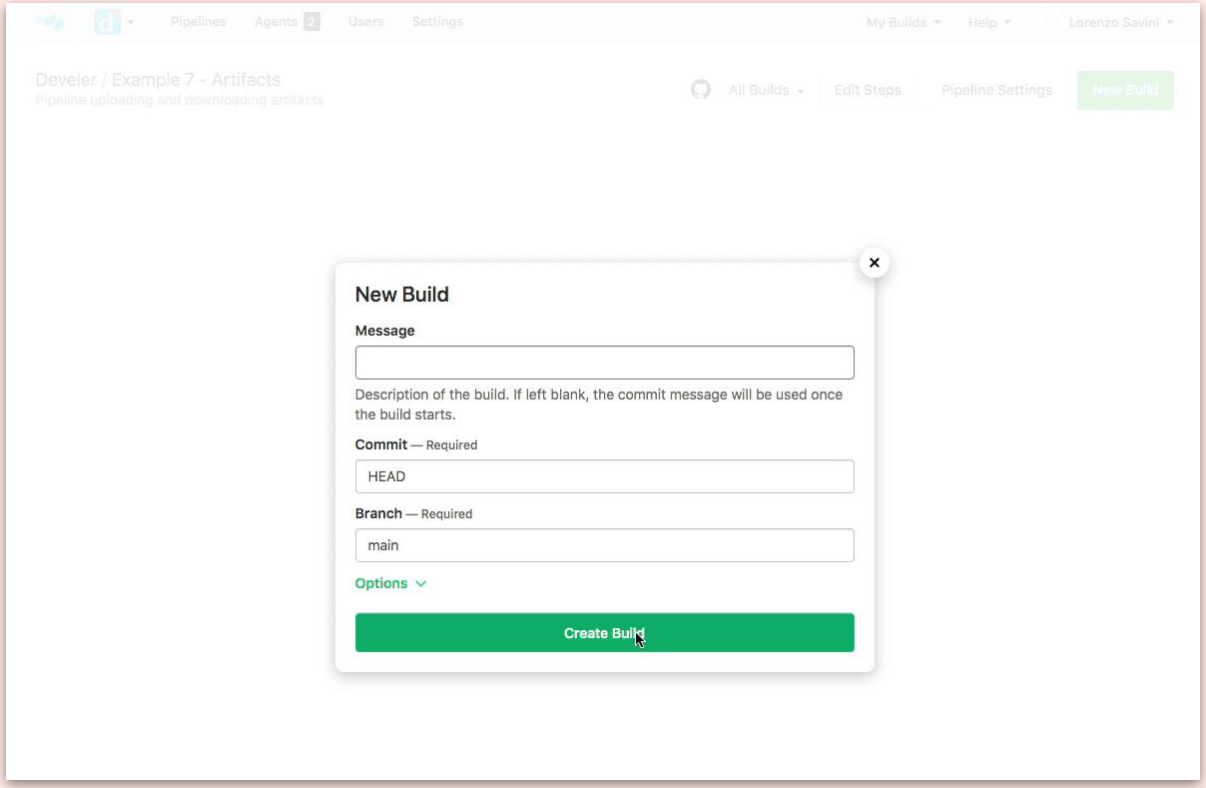
Upload/Download tramite CLI - Meta-data e artifacts

```
buildkite-agent artifact upload dist/index.html

buildkite-agent artifact download dist/* local-dist/
```

Potete anche scaricare artefatti caricati da una build lanciata da un trigger step, basta aggiungere `--build $BUILDKITE_TRIGGERED_FROM_BUILD_ID`

Esempio #7

Esempio 7 - Meta-data e artifacts



Pipeline dinamiche

Buildkite permette di generare dinamicamente nuovi step in una pipeline

Pipeline dinamiche

Buildkite permette di generare pipeline dinamiche, semplicemente passando ulteriori step a buildkite-agent:

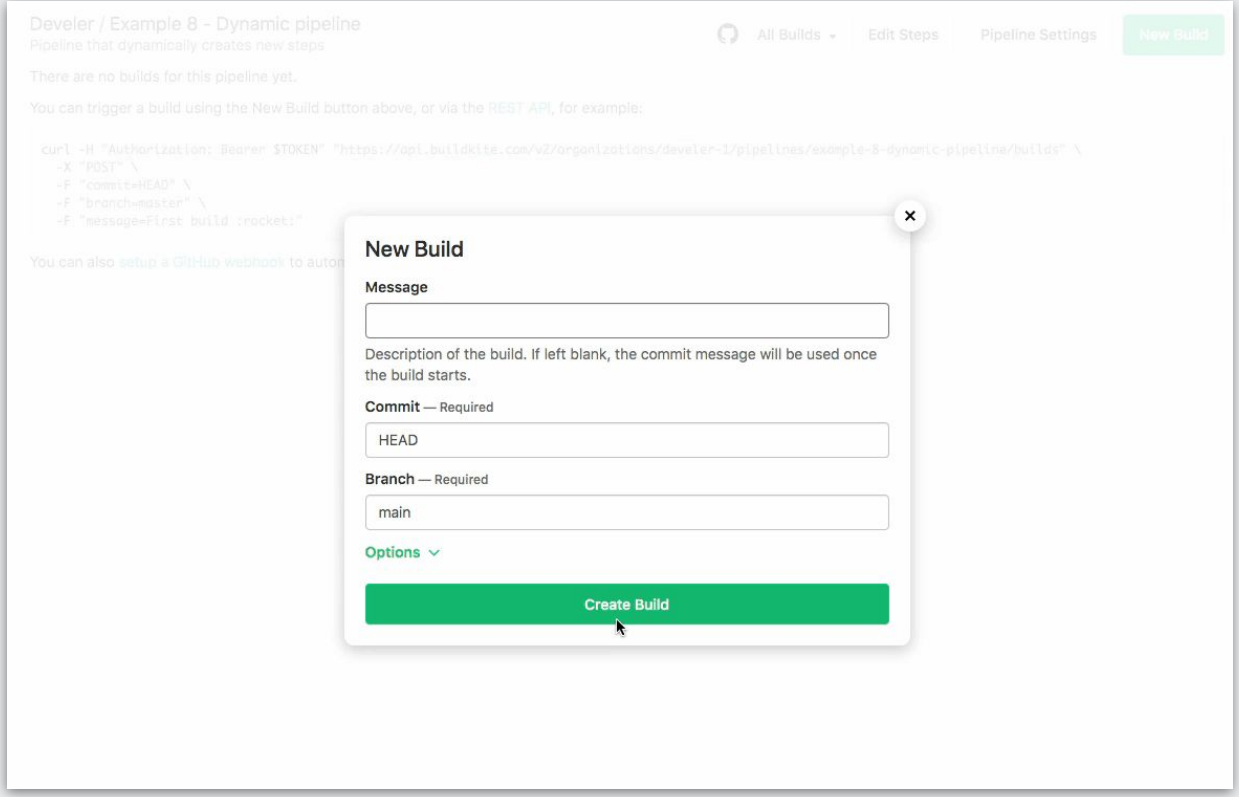
```
./pipeline.sh | buildkite-agent pipeline upload
```

Script per pipeline dinamiche

```
# Opzione 1
echo "  - name: 'Step #$i'"
echo "    command: 'echo \"This is step #$i\"'"

# Opzione 2, usando heredoc
cat <<YAML
- name: 'Step #$i'
 command: 'echo "This is step #$i"'
YAML
```

Esempio #8

Esempio 8 - Pipeline dinamiche



Conditionals and dependencies

Vediamo ora un po’ di opzioni extra per gli step

If - Conditionals e dependencies

```yaml
steps:
  - name: 'Hello world!'
    command: './hello-world'
    if: build.message !~ /skip hello world/
```

Buildkite permette di configurare i nostri step per essere eseguiti soltanto in certe condizioni.

Una delle opzioni che possiamo utilizzare è `if`.

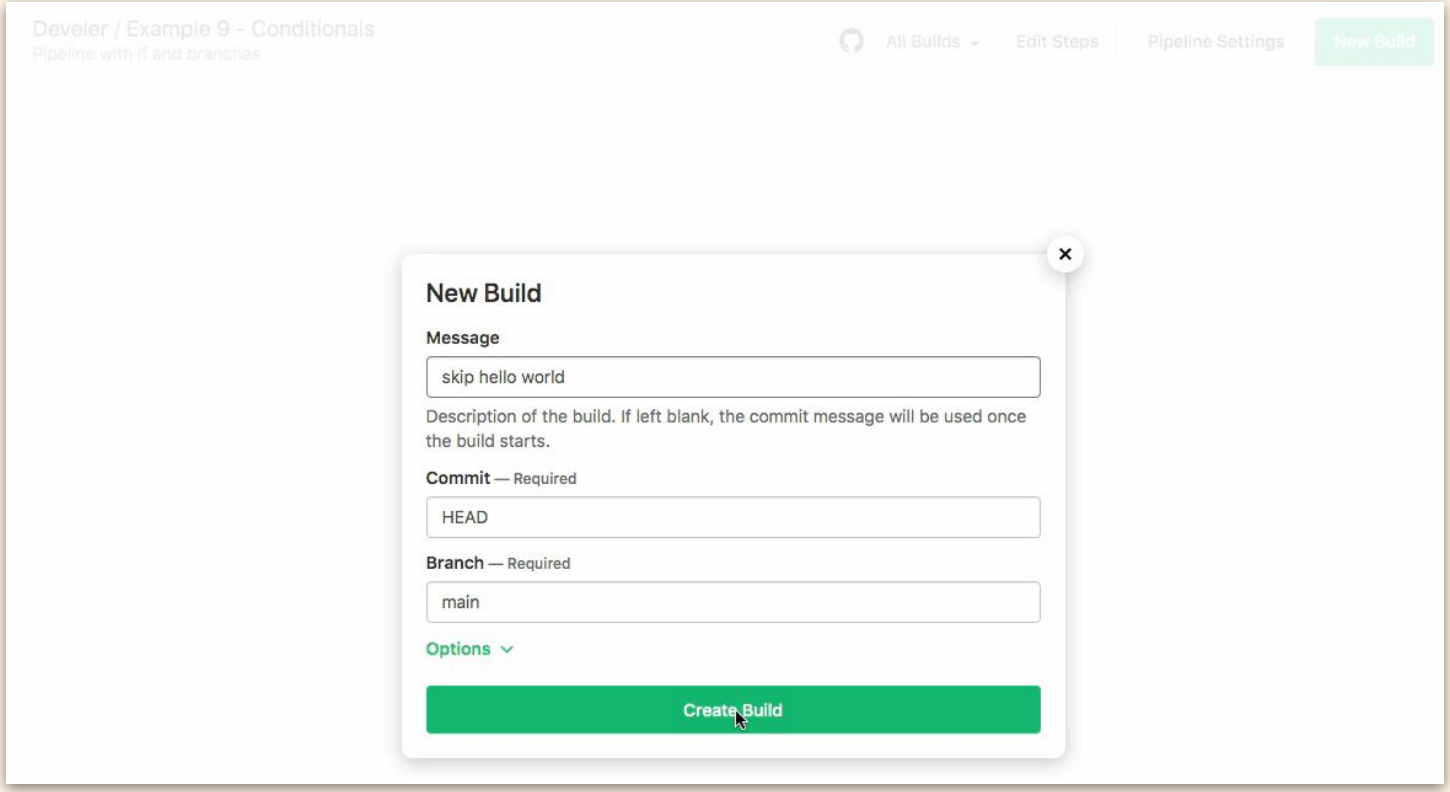
Branches - Conditionals e dependencies

```
steps:
  - name: 'Hello world!'
    command: './hello-world'
    branches: 'main'
```

Un’altra opzione è `branches`, per attivare uno step soltanto per alcune branch.

Esempio #9

Esempio 9: if e branches - Conditionals e dependencies



Depends_on - Conditionals e dependencies

```
steps:
  - key: 'hello-world-1'
    command: './hello-world'

  - key: 'hello-world-2'
    command: './hello-world'
    depends_on: 'hello-world-1'
```

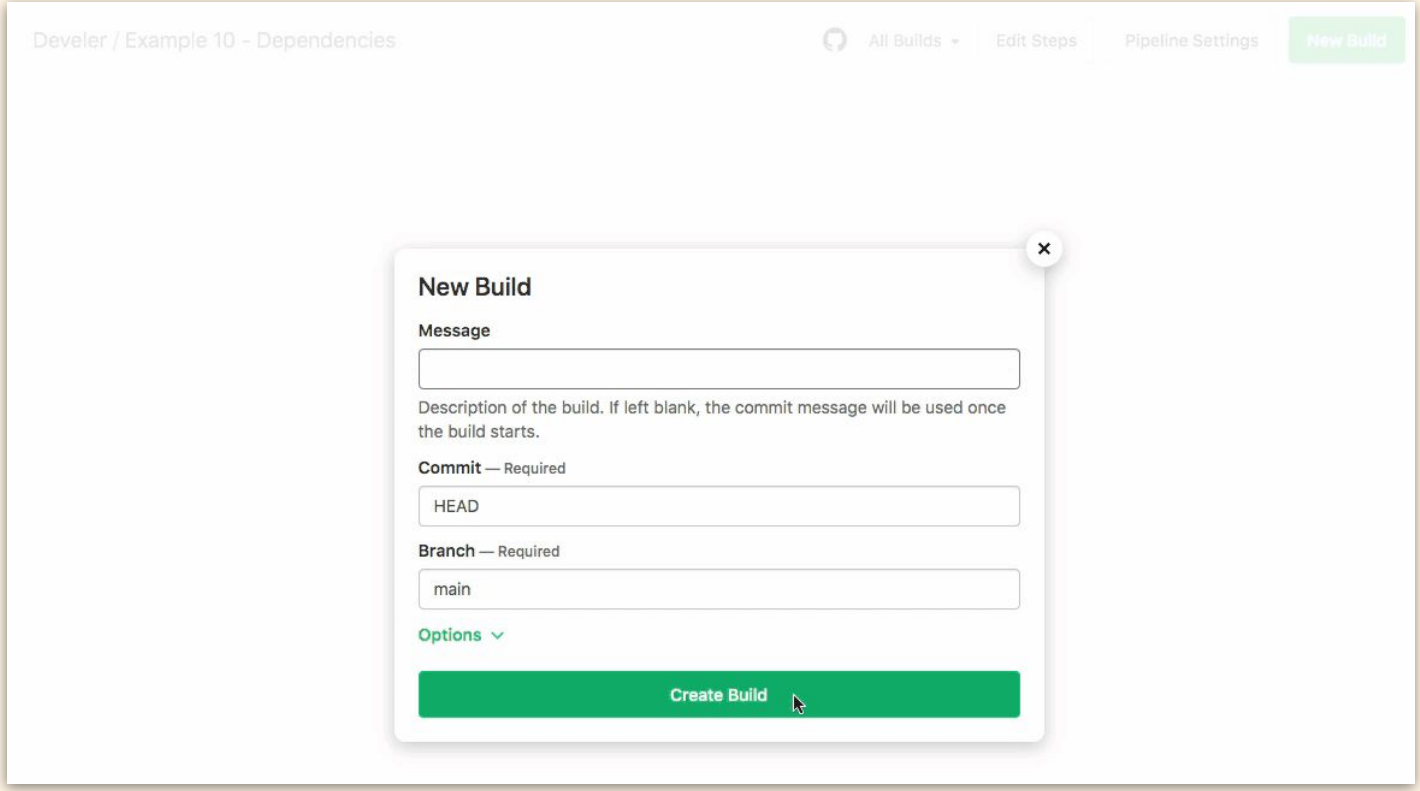
Depends_on invece permette di dichiarare dipendenze esplicite tra step.

È un'alternativa più flessibile all'uso di wait, ad esempio.

Di default, lo step `hello-world-2` non partirà se `hello-world-1` dovesse fallire. Ma è possibile settare `allow_dependency_failure: true` per cambiare questo comportamento.

Esempio #10

Esempio 10: depends_on - Conditionals e dependencies



Concurrency - Conditionals e dependencies

```yaml
steps:
  - key: 'hello-world-1'
    command: './hello-world'
    concurrency: 1
    concurrency_group: 'hello-world/run'
```

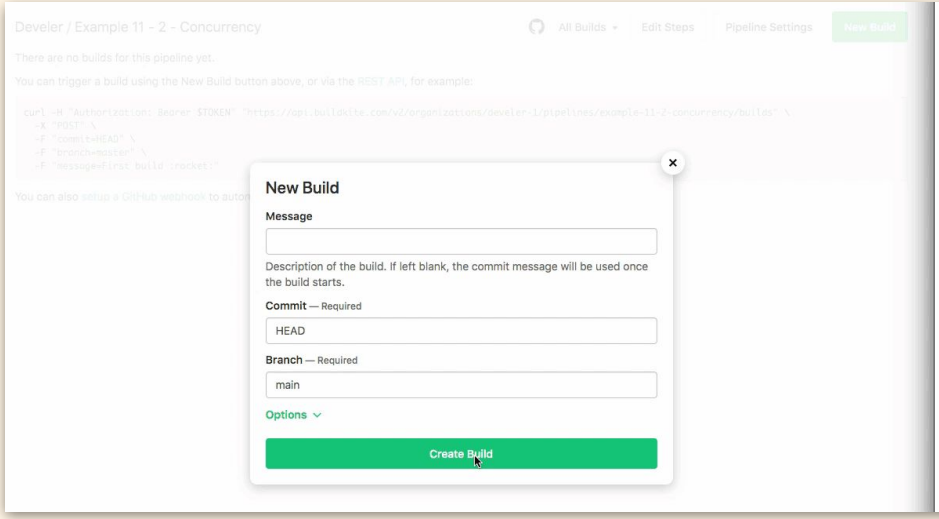
È anche possibile creare gruppi di step che dovranno rispettare regole di concorrenza (anche tra pipeline).

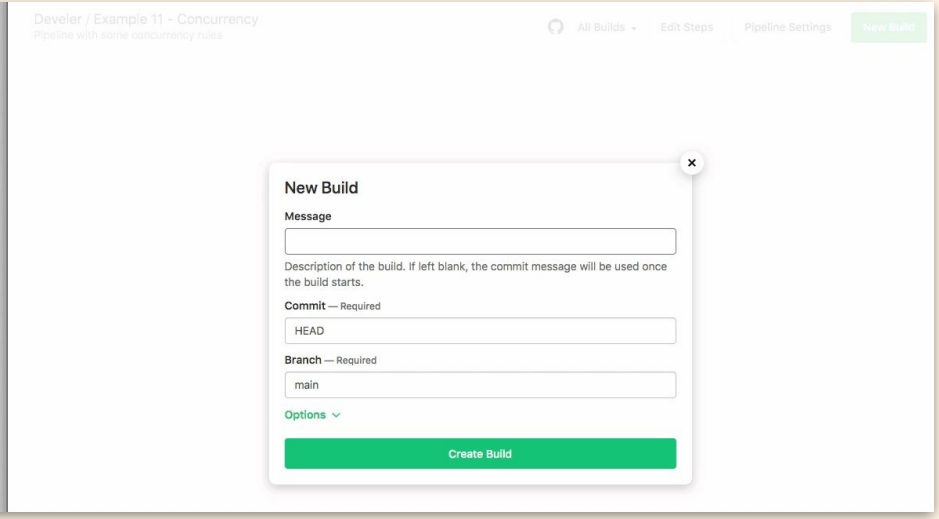
In questo caso abbiamo 2 parametri:

- concurrency_group
- concurrency

Esempio #11

Esempio 11: concurrency - Conditionals e dependencies

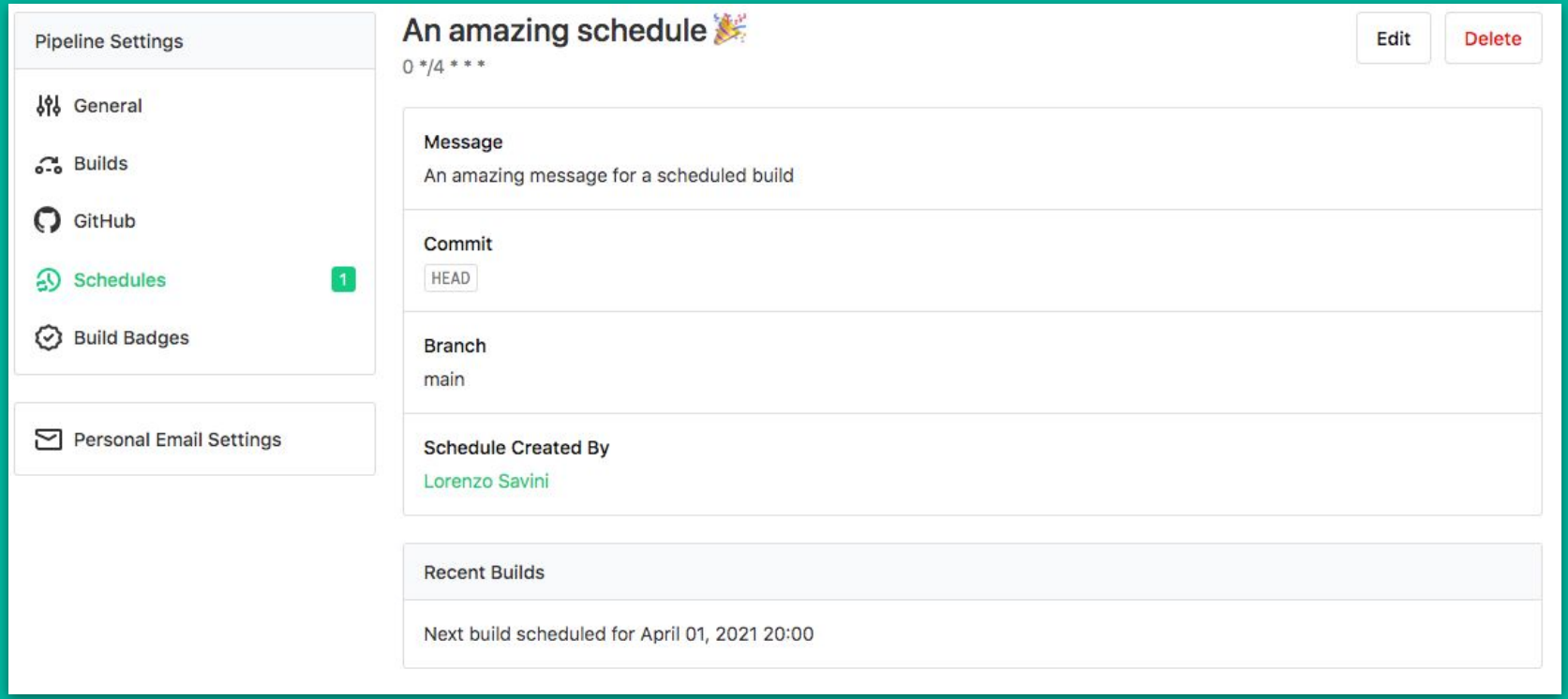




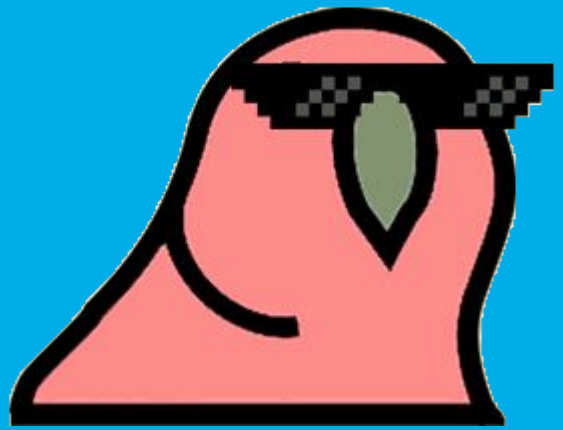
Pipeline schedulate

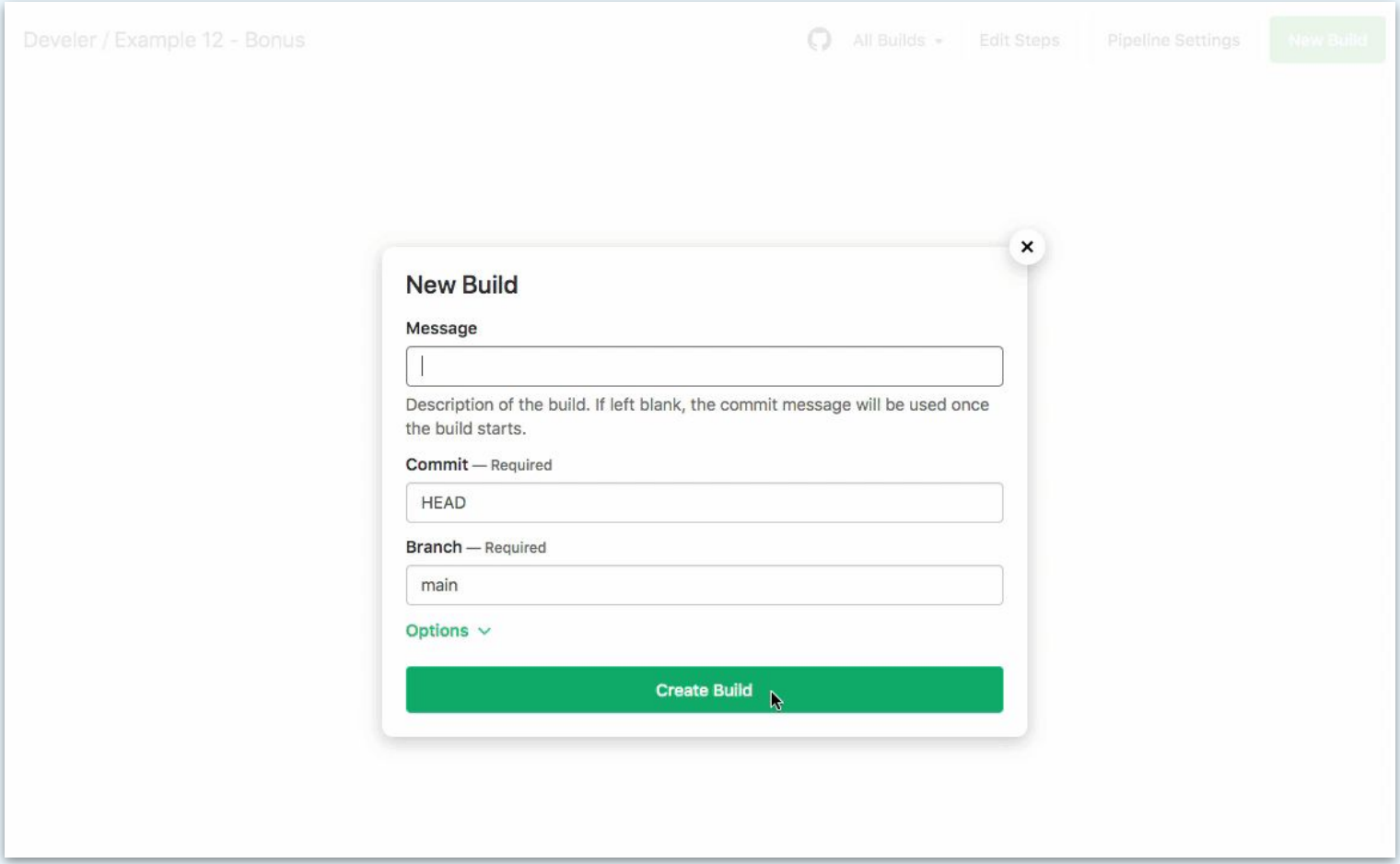
Buildkite permette di creare uno o più schedules per una pipeline, così da poterla eseguire a intervalli specifici

Pipeline schedulate



Bonus




Develer / Example 12 - Bonus
All Builds
Edit Steps
Pipeline Settings
New Build
New Build
Message
Description of the build. If left blank, the commit message will be used once the build starts.
Commit — Required
HEAD
Branch — Required
main
Options
Create Build

Lorenzo Savini

savo@develer.com

Vuoi rimanere aggiornato sugli eventi Develer?
Seguici nei nostri canali social:

develer

www.develer.com