

[Ruby_](https://cloud.google.com/ruby/) (<https://cloud.google.com/ruby/>) [Guides](#)

Using Pub/Sub with Ruby

Many apps need to do background processing outside of the context of a web request. In this sample, the Bookshelf app sends tasks to a separate background worker for execution. The worker gathers information from the [Google Books API](https://developers.google.com/books/) (<https://developers.google.com/books/>) and updates the book information in the database. This sample demonstrates how to set up separate services in App Engine, how to run a worker process in the App Engine flexible environment, and how to deal with lifecycle events.

This page is part of a multipage tutorial. To start from the beginning and read the setup instructions, go to [Ruby Bookshelf app](https://cloud.google.com/ruby/getting-started/tutorial-app) (<https://cloud.google.com/ruby/getting-started/tutorial-app>).

Installing dependencies

Go to the `getting-started-ruby/6-task-queueing` directory, and enter the following command.

```
bundle install
```



Creating a Pub/Sub topic and subscription

The Bookshelf app uses [Pub/Sub](https://cloud.google.com/pubsub) (<https://cloud.google.com/pubsub>) for a background processing queue of requests to get data from the [Google Books API](https://developers.google.com/books/) (<https://developers.google.com/books/>) for a book added to the Bookshelf.

1. Create a new Pub/Sub topic using the following Cloud SDK command where `[YOUR_PUBSUB_TOPIC]` represents your Pub/Sub topic name.

```
gcloud pubsub topics create [YOUR_PUBSUB_TOPIC]
```



2. Create a new Pub/Sub subscription for the topic created in the previous step. Replace `[YOUR_PUBSUB_SUBSCRIPTION]` with the name you want to give to this new Pub/Sub subscription.

```
gcloud pubsub subscriptions create --topic [YOUR_PUBSUB_TOPIC] [YOUR_PUBSUB_SUBSCRIPTION]
```



Configuring settings

1. Copy the example settings file.

```
cp config/settings.example.yml config/settings.yml
```



2. Open the `settings.yml` file for editing. Replace the `[YOUR_PROJECT_ID]` with your Google Cloud project ID.

[6-task-queueing/config/settings.example.yml](https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/steps/6-task-queueing/config/settings.example.yml)

(<https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/steps/6-task-queueing/config/settings.example.yml>)

`_ATFORM/GETTING-STARTED-RUBY/BLOB/STEPS/6-TASK-QUEUEING/CONFIG/SETTINGS.EXAMPLE.YML)`

```
default: &default
  project_id: [YOUR_PROJECT_ID]
  gcs_bucket: [YOUR_BUCKET_NAME]
  pubsub_topic: [YOUR_PUBSUB_TOPIC]
  pubsub_subscription: [YOUR_PUBSUB_SUBSCRIPTION]
  oauth2:
    client_id: [YOUR_CLIENT_ID]
    client_secret: [YOUR_CLIENT_SECRET]
```



3. Set the other variables to the same values you used in the [Authenticating users](https://cloud.google.com/ruby/getting-started/authenticate-users) (<https://cloud.google.com/ruby/getting-started/authenticate-users>) part of this tutorial.

For example, suppose your web app's client ID is `XYZCLIENTID` and your client secret is `XYZCLIENTSECRET`. Also suppose your project name is `my-project`, and your Cloud Storage bucket name is `my-bucket`. Then the default section of your `settings.yml` file would look like this:

```
default: &default
  project_id: my-project
  gcs_bucket: my-bucket
  pubsub_topic: your-pubsub-topic
  pubsub_subscription: your-pubsub-subscription
  oauth2:
    client_id: XYZCLIENTID
    client_secret: XYZCLIENTSECRET
```

4. Copy the example database configuration file.

```
cp config/database.example.yml config/database.yml
```

5. Configure the sample app to use the same database that you set up during the [Using structured data](https://cloud.google.com/ruby/getting-started/using-structured-data) (https://cloud.google.com/ruby/getting-started/using-structured-data) part of this tutorial.

CLOUD SQL

POSTGRES SQL

DATASTORE

- Edit `database.yml`. Uncomment the lines in the Cloud SQL portion of the file.

```
mysql_settings: &mysql_settings
  adapter: mysql2
  encoding: utf8
  pool: 5
  timeout: 5000
  username: [MYSQL_USER]
  password: [MYSQL_PASS]
  database: [MYSQL_DATABASE]
  socket: /cloudsql/[YOUR_INSTANCE_CONNECTION_NAME]
```

- Replace `[MYSQL_USER]` and `[MYSQL_PASS]` with your Cloud SQL instance username and password that you created previously.
- Replace `[MYSQL_DATABASE]` with the name of the database that you created previously.
- Replace `[YOUR_INSTANCE_CONNECTION_NAME]` with the **Instance Connection Name** of your Cloud SQL instance.

Note: You can retrieve the Cloud SQL instance connection name by running `gcloud beta sql instances describe [YOUR_INSTANCE_NAME]`.

- Run migrations.

```
bundle exec rake db:migrate
```



Running the app on your local machine

1. Start the local web server and two worker processes.

```
bundle exec foreman start --formation web=1,worker=2
```



2. In your web browser, enter `http://localhost:8080`.

Now add some books to the Bookshelf. You can watch the workers update the book information in the background.

The Foreman (<http://ddollar.github.io/foreman/>) RubyGem starts the Rails web server and runs two worker processes.

The worker establishes a Pub/Sub subscription to listen for events. After the subscription exists, events published to the topic are queued, even if there is no worker currently listening for events. When a worker comes online, Pub/Sub delivers any queued events.

When you're ready to move forward, press `Ctrl+C` to exit the local web server and worker processes.

Deploying the app to the App Engine flexible environment

1. Compile the JavaScript assets for production.

```
RAILS_ENV=production bundle exec rake assets:precompile
```



2. Deploy the worker.

```
gcloud app deploy worker.yaml
```



3. Deploy the sample app.

```
gcloud app deploy
```



4. In your web browser, enter the following address.

```
https://[YOUR_PROJECT_ID].appspot.com
```



If you update your app, you can deploy the updated version by entering the same command you used to deploy the app the first time. The new deployment creates a new version (<https://console.cloud.google.com/appengine/versions>) of your app and promotes it to the default version. The older versions of your app remain, as do their associated VM instances. Be aware that all of these app versions and VM instances are billable resources.

You can reduce costs by deleting the non-default versions of your app.

To delete an app version:


1. In the Cloud Console, go to the **Versions** page for App Engine.

GO TO THE VERSIONS PAGE ([HTTPS://CONSOLE.CLOUD.GOOGLE.COM/APPENGINE/VERSIONS](https://console.cloud.google.com/appengine/versions))

2. Select the checkbox for the non-default app version you want to delete.

Note: The only way you can delete the default version of your App Engine app is by deleting your project. However, you can [stop the default version in the Cloud Console](https://console.cloud.google.com/appengine/versions) (<https://console.cloud.google.com/appengine/versions>). This action shuts down all instances associated with the version. You can restart these instances later if needed.

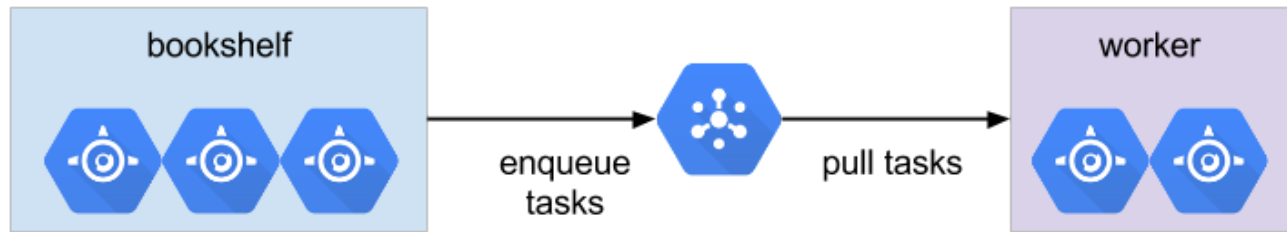
In the App Engine standard environment, you can stop the default version only if your app has manual or basic scaling.

3. Click **Delete**  to delete the app version.

For complete information about cleaning up billable resources, see the [Cleaning up](https://cloud.google.com/ruby/getting-started/using-pub-sub#clean-up) (<https://cloud.google.com/ruby/getting-started/using-pub-sub#clean-up>) section in the final step of this tutorial.

App structure

This diagram shows the app's components and how they fit together.



Understanding the code

This section walks you through the app's code and explains how it works.

Queue tasks

To gather information from the [Google Books API](https://developers.google.com/books/) (<https://developers.google.com/books/>) for books added to the Bookshelf, the `Book` class adds a task to the queue.

[6-task-queueing/app/models/book.rb](https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/6-task-queueing/app/models/book.rb)

(<https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/6-task-queueing/app/models/book.rb>)

E/CLOUDPLATFORM/GETTING-STARTED-RUBY/BLOB/STEPS/6-TASK-QEUEING/APP/MODELS/BOOK.RB)

```
after_create :lookup_book_details
```

```
def lookup_book_details
  if [author, description, published_on, image_url].any? {|attr| attr.blank? }
    LookupBookDetailsJob.perform_later self
  end
end
```

The preceding code creates an [Active Record callback](http://guides.rubyonrails.org/active_record_callbacks.html)

(http://guides.rubyonrails.org/active_record_callbacks.html) and specifies that after a book is created and saved in the database, `lookup_book_details` is called. If the book is missing any information, it adds the job, look up the book's details, to the queue.

`LookupBookDetailsJob` is an [Active Job](http://guides.rubyonrails.org/active_job_basics.html) (http://guides.rubyonrails.org/active_job_basics.html) job.

The code passes `self`, referencing the book, to `LookupBookDetailsJob.perform_later`. This adds a job to look up the book's details to the queue.

Pub/Sub Active Job backend

You can configure Active Job to use a custom [backend](#)

(http://guides.rubyonrails.org/active_job_basics.html#backends). For example, use [delayed_job](https://github.com/collectiveidea/delayed_job) (https://github.com/collectiveidea/delayed_job) or [resque](https://github.com/resque/resque) (<https://github.com/resque/resque>), to add tasks to the queue. The Bookshelf sample app has its own custom backend, which is specified in the `Application` class.

[6-task-queueing/config/application.rb](https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/6-task-queueing/config/application.rb)

(<https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/6-task-queueing/config/application.rb>)

`LOUDPLATFORM/GETTING-STARTED-RUBY/BLOB/STEPS/6-TASK-QUEUEING/CONFIG/APPLICATION.RB)`

```
config.active_job.queue_adapter = :pub_sub_queue
```



An Active Job backend, which is also called an adapter, must provide an `enqueue` method. When a job is enqueued using `perform_later`, the job is passed to the `enqueue` method of the configured Active Job backend.

The sample app adds a job to the queue by creating a subscription to a Pub/Sub topic, and then publishing the ID of a book to the topic. Once the subscription exists, messages are queued even if there is no worker currently listening. When a worker comes online, Pub/Sub delivers any queued events.

[6-task-queueing/lib/active_job/queue_adapters/pub_sub_queue_adapter.rb](https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/6-task-queueing/lib/active_job/queue_adapters/pub_sub_queue_adapter.rb)

(https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/6-task-queueing/lib/active_job/queue_adapters/pub_sub_queue_adapter.rb)

`Y/BLOB/STEPS/6-TASK-QUEUEING/LIB/ACTIVE_JOB/QUEUE_ADAPTERS/PUB_SUB_QUEUE_ADAPTER.RB)`

```
require "google/cloud/pubsub"
```



```
module ActiveJob
  module QueueAdapters
    class PubSubQueueAdapter

      def self.pubsub
        @pubsub ||= begin
          project_id = Rails.application.config.x.settings["project_id"]
          Google::Cloud::Pubsub.new project_id: project_id
        end
      end
    end
  end
end
```

```
def self.pubsub_topic
  @pubsub_topic ||= Rails.application.config.x.settings["pubsub_topic"]
end

def self.pubsub_subscription
  @pubsub_subscription ||= Rails.application.config.x.settings["pubsub_subscri
end

def self.enqueue job
  Rails.logger.info "[PubSubQueueAdapter] enqueue job #{job.inspect}"

  book = job.arguments.first

  topic = pubsub.topic pubsub_topic

  topic.publish book.id.to_s
end
```

The preceding code uses the [google-cloud-pubsub](https://googleapis.dev/ruby/google-cloud-pubsub/latest/Google/Cloud/PubSub.html)

(<https://googleapis.dev/ruby/google-cloud-pubsub/latest/Google/Cloud/PubSub.html>) RubyGem to interact with Pub/Sub. The Cloud client library is an idiomatic Ruby client for interacting with Google Cloud services.

[6-task-queueing/Gemfile](https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/6-task-queueing/Gemfile)

(<https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/6-task-queueing/Gemfile>)

B.COM/GOOGLECLOUDPLATFORM/GETTING-STARTED-RUBY/BLOB/STEPS/6-TASK-QUEUEING/GEMFILE)

```
gem "google-cloud-pubsub", "~> 0.30"
```



To process books added to a queue, a [Pub/Sub subscription](https://cloud.google.com/pubsub/subscriber)

(<https://cloud.google.com/pubsub/subscriber>) listens for messages published to the `lookup_book_details_queue` topic. This is covered in the [worker](#) (`#the_worker`) section.

Books API

The sample app uses the [Google API client](https://github.com/google/google-api-ruby-client/tree/v0.8.6)

(<https://github.com/google/google-api-ruby-client/tree/v0.8.6>) RubyGem to look up book details from the Books API.

[6-task-queueing/Gemfile](https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/6-task-queueing/Gemfile)

(<https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/6-task-queueing/Gemfile>)

B.COM/GOOGLECLOUDPLATFORM/GETTING-STARTED-RUBY/BLOB/STEPS/6-TASK-QUEUEING/GEMFILE)

```
gem "google-api-client", "~> 0.19"
```



When a job runs, the `LookupBookDetailsJob.perform` method retrieves a list of books, based on a book title, from the Books API.

[6-task-queueing/app/jobs/lookup_book_details_job.rb](https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/6-task-queueing/app/jobs/lookup_book_details_job.rb)

(https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/6-task-queueing/app/jobs/lookup_book_details_job.rb)

GETTING-STARTED-RUBY/BLOB/STEPS/6-TASK-QUEUEING/APP/JOBS/LOOKUP_BOOK_DETAILS_JOB.RB)

```
require "google/apis/books_v1"
```



```
class LookupBookDetailsJob < ActiveJob::Base
  queue_as :default

  def perform book
    Rails.logger.info "[BookService] Lookup details for book" +
      "#{book.id} #{book.title.inspect}"

    # Create Book API Client
    book_service = Google::Apis::BooksV1::BooksService.new

    # Lookup a list of relevant books based on the provided book title.
    book_service.list_volumes(book.title, order_by: "relevance") do |results, error|
      # Error occurred soft-failure
      if error
        Rails.logger.error "[BookService] #{error.inspect}"
        break
      end

      # Book was not found
      if results.total_items.zero?
        Rails.logger.info "[BookService] #{book.title} was not found."
        break
      end

      # List of relevant books
      volumes = results.items
    end
  end
end
```

If a book volume result includes a title, author, and book cover image, then it is selected as the best match. Otherwise the first result is used.

[6-task-queueing/app/jobs/lookup_book_details_job.rb](https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/6-task-queueing/app/jobs/lookup_book_details_job.rb)

(https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/6-task-queueing/app/jobs/lookup_book_details_job.rb)

GETTING-STARTED-RUBY/BLOB/STEPS/6-TASK-QUEUEING/APP/JOBS/LOOKUP_BOOK_DETAILS_JOB.RB)

```
# To provide the best results, find the first returned book that
# includes title and author information as well as a book cover image.
best_match = volumes.find {|volume|
  info = volume.volume_info
  info.title && info.authors && info.image_links.try(:thumbnail)
}

volume = best_match || volumes.first
```

If any relevant volume is found, the book details are updated and saved in the database.

[6-task-queueing/app/jobs/lookup_book_details_job.rb](https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/6-task-queueing/app/jobs/lookup_book_details_job.rb)

(https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/6-task-queueing/app/jobs/lookup_book_details_job.rb)

GETTING-STARTED-RUBY/BLOB/STEPS/6-TASK-QUEUEING/APP/JOBS/LOOKUP_BOOK_DETAILS_JOB.RB)

```
if volume
  info = volume.volume_info
  images = info.image_links

  publication_date = info.published_date
  publication_date = "#{info.published_date}-01-01" if publication_date =~ /^(\d{4})$/
  publication_date = Date.parse publication_date

  book.author = info.authors.join(", ") unless book.author.present?
  book.published_on = publication_date unless book.published_on.present?
  book.description = info.description unless book.description.present?
  book.image_url = images.try(:thumbnail) unless book.image_url.present?

  book.save
end
```

The worker

A worker process handles book lookup jobs. To run the worker, you can run the following command, as specified in `Procfile`.

```
bundle exec rake run_worker
```



The `run_worker` rake task calls `PubSubQueueAdapter` to start a worker.

[6-task-queueing/lib/tasks/run_worker.rake](https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/6-task-queueing/lib/tasks/run_worker.rake)

(https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/6-task-queueing/lib/tasks/run_worker.rake)

```
PLATFORM/GETTING-STARTED-RUBY/BLOB/STEPS/6-TASK-QUEUEING/LIB/TASKS/RUN_WORKER.RAKE)
```

```
desc "Run task queue worker"
task run_worker: :environment do
  ActiveJob::QueueAdapters::PubSubQueueAdapter.run_worker!
end
```



When the worker runs, it listens for messages on the Pub/Sub subscription to the `lookup_book_details_queue` topic defined in your `config/settings.yml` file. When a message is received, the associated book is retrieved from the database and the `LookupBookDetailsJob` runs immediately to update the book.

[6-task-queueing/lib/active_job/queue_adapters/pub_sub_queue_adapter.rb](https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/6-task-queueing/lib/active_job/queue_adapters/pub_sub_queue_adapter.rb)

(https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/6-task-queueing/lib/active_job/queue_adapters/pub_sub_queue_adapter.rb)

```
Y/BLOB/STEPS/6-TASK-QUEUEING/LIB/ACTIVE_JOB/QUEUE_ADAPTERS/PUB_SUB_QUEUE_ADAPTER.RB)
```

```
def self.run_worker!
  Rails.logger.info "Running worker to lookup book details"

  topic          = pubsub.topic pubsub_topic
  subscription   = topic.subscription pubsub_subscription

  subscriber = subscription.listen do |message|
    message.acknowledge!

    Rails.logger.info "Book lookup request (#{message.data})"

    book_id = message.data.to_i
    book    = Book.find_by_id book_id
```



```
    LookupBookDetailsJob.perform_now book if book
  end

  # Start background threads that will call block passed to listen.
  subscriber.start

  # Fade into a deep sleep as worker will run indefinitely
  sleep
end
```

Running on Google Cloud

The worker is deployed as a separate module within the same app. App Engine apps can have multiple, independent services. This means that you can independently deploy, configure, scale, and update pieces of your app. The frontend is deployed to the *default module*, and the worker is deployed to the *worker module*.

Even though the worker doesn't serve any web requests to users, or even run a web app, we strongly recommend that you provide an HTTP health check when running in the App Engine flexible environment to ensure that the service is running and responsive. It is, however, possible to disable health checking.

To provide a health check, the worker starts two processes instead of one. The first process is `worker` and the second process is `health_check`, which runs a simple Rack app that responds to HTTP requests with a successful response for health checks.

[6-task-queueing/health_check.ru](#)

(https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/steps/6-task-queueing/health_check.ru)

OGGLECLOUDPLATFORM/GETTING-STARTED-RUBY/BLOB/STEPS/6-TASK-QUEUEING/HEALTH_CHECK.RU)

```
# Respond to HTTP requests with non-500 error code
run lambda {|env| [200, {"Content-Type" => "text/plain"}, ["ok"]] }
```

The app uses [Foreman](http://ddollar.github.io/foreman/) (<http://ddollar.github.io/foreman/>) to manage multiple processes. The processes are configured in `Procfile`.

[6-task-queueing/Procfile](#)

(<https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/steps/6-task-queueing/Procfile>)

.COM/GOOGLECLOUDPLATFORM/GETTING-STARTED-RUBY/BLOB/STEPS/6-TASK-QUEUEING/PROCFILE)

```
web: bundle exec rackup -p 8080
worker: bundle exec rake run_worker
health_check: bundle exec rackup -p 8080 health_check.ru
```

Foreman is now used as the **entrypoint** for the docker container. This is specified in the **app.yaml** and **worker.yaml** files.

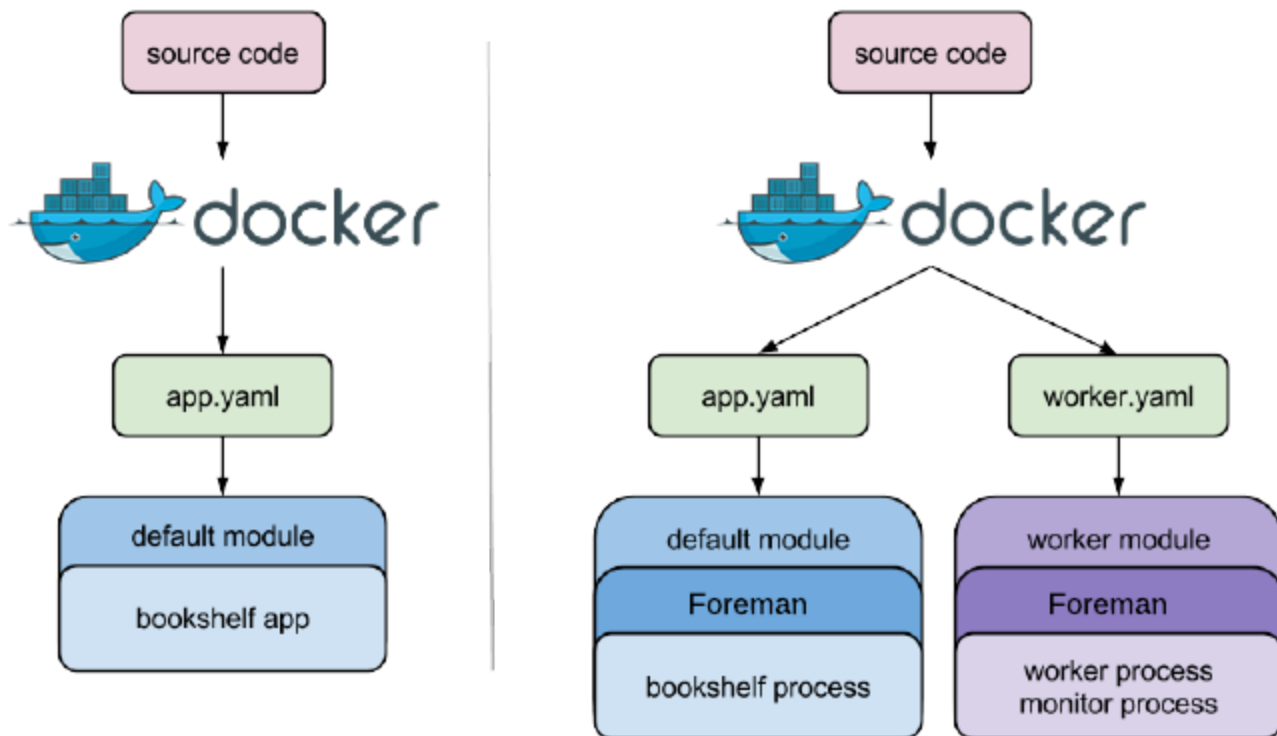
[6-task-queueing/app.yaml](#)

(<https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/steps/6-task-queueing/app.yaml>)

.COM/GOOGLECLOUDPLATFORM/GETTING-STARTED-RUBY/BLOB/STEPS/6-TASK-QUEUEING/APP.YAML)

```
entrypoint: bundle exec foreman start --formation "$FORMATION"
```

Notice that **Procfile** contains an entry for the **web** frontend to run the Bookshelf Rails app as well. Because the default (frontend) and worker services share the same codebase, the **FORMATION** environment variable controls which processes are started. The following diagram contrasts the single module deployment on the left with the multi-module deployment on the right.



The environment variables are set by the **app.yaml** and **worker.yaml** files.

[6-task-queueing/app.yaml](#)

(<https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/steps/6-task-queueing/app.yaml>)

.COM/GOOGLECLOUDPLATFORM/GETTING-STARTED-RUBY/BLOB/STEPS/6-TASK-QUEUEING/APP.YAML)

```
env_variables:  
  FORMATION: web=1
```



The worker is a separate module, so it needs its own YAML configuration file.

[6-task-queueing/worker.yaml](#)

(<https://github.com/GoogleCloudPlatform/getting-started-ruby/blob/steps/6-task-queueing/worker.yaml>)

1/GOOGLECLOUDPLATFORM/GETTING-STARTED-RUBY/BLOB/STEPS/6-TASK-QUEUEING/WORKER.YAML)

```
env_variables:  
  FORMATION: worker=5, health_check=1
```



This configuration is similar to the `app.yaml` file that is used for the frontend; the key differences are the `module: worker` setting, and the `FORMATION` environment variable, which configures Foreman to run five workers and the frontend for the health check instead of the Bookshelf web app.

Cleaning up

To avoid incurring charges to your Google Cloud Platform account for the resources used in this tutorial:

Delete the project

The easiest way to eliminate billing is to delete the project that you created for the tutorial.

To delete the project:

Caution: Deleting a project has the following effects:


- **Everything in the project is deleted.** If you used an existing project for this tutorial, when you delete it, you also delete any other work you've done in the project.

- **Custom project IDs are lost.** When you created this project, you might have created a custom project ID that you want to use in the future. To preserve the URLs that use the project ID, such as an **appspot.com** URL, delete selected resources inside the project instead of deleting the whole project.

If you plan to explore multiple tutorials and quickstarts, reusing projects can help you avoid exceeding project quota limits.

1. In the Cloud Console, go to the **Manage resources** page.

[GO TO THE MANAGE RESOURCES PAGE](https://console.cloud.google.com/iam-admin/projects) (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/IAM-ADMIN/PRO)

2. In the project list, select the project you want to delete and click **Delete**  .
3. In the dialog, type the project ID, and then click **Shut down** to delete the project.

Delete non-default versions of your app

If you don't want to delete your project, you can reduce costs by deleting the non-default versions of your app.

To delete an app version:


1. In the Cloud Console, go to the **Versions** page for App Engine.

[GO TO THE VERSIONS PAGE](https://console.cloud.google.com/appengine/versions) (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/APPENGINE/VERSIONS)

2. Select the checkbox for the non-default app version you want to delete.

Note: The only way you can delete the default version of your App Engine app is by deleting your project. However, you can [stop the default version in the Cloud Console](https://console.cloud.google.com/appengine/versions) (<https://console.cloud.google.com/appengine/versions>). This action shuts down all instances associated with the version. You can restart these instances later if needed.

In the App Engine standard environment, you can stop the default version only if your app has manual or basic scaling.


3. Click **Delete**  to delete the app version.

Delete your Cloud SQL instance

To delete a Cloud SQL instance:

1. In the Cloud Console, go to the **SQL Instances** page.

[GO TO THE SQL INSTANCES PAGE \(HTTPS://CONSOLE.CLOUD.GOOGLE.COM/SQL/INSTANCES\)](https://console.cloud.google.com/sql/instances)


2. Click the name of the SQL instance you want to delete.
3. Click **Delete**  to delete the instance.

Delete your Cloud Storage bucket

To delete a Cloud Storage bucket:

1. In the Cloud Console, go to the **Cloud Storage Browser** page.

[GO TO THE CLOUD STORAGE BROWSER PAGE \(HTTPS://CONSOLE.CLOUD.GOOGLE.COM/STORAGE\)](https://console.cloud.google.com/storage)

2. Click the checkbox for the bucket you want to delete.
3. Click **Delete**  to delete the bucket.

What's next

Learn how to [run the Ruby Bookshelf sample app on Compute Engine](https://cloud.google.com/ruby/getting-started/run-on-compute-engine)

(<https://cloud.google.com/ruby/getting-started/run-on-compute-engine>).

Try out other Google Cloud Platform features for yourself. Have a look at our [tutorials](https://cloud.google.com/docs/tutorials)

(<https://cloud.google.com/docs/tutorials>).

[PREV \(HTTPS://CLOUD.GOOGLE.COM/RUBY/GETTING-STARTED/LOGGING-APPLICATION-EVENTS\)](https://cloud.google.com/ruby/getting-started/logging-application-events)

NEXT >

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 4.0 License](https://creativecommons.org/licenses/by/4.0/) (<https://creativecommons.org/licenses/by/4.0/>), and code samples are licensed under the [Apache 2.0 License](https://www.apache.org/licenses/LICENSE-2.0) (<https://www.apache.org/licenses/LICENSE-2.0>). For details, see our [Site Policies](https://developers.google.com/terms/site-policies) (<https://developers.google.com/terms/site-policies>). Java is a registered trademark of Oracle and/or its affiliates.

Last updated November 19, 2019.