Getting started with .NET

This tutorial is intended for those new to building apps in the cloud, such as engineers and web developers, who want to learn key app development concepts as they apply to Google Cloud.

Objectives

- Learn basic Google Cloud tools, such as the Google Cloud Console (/cloud-console) and `gcloud` (/sdk/gcloud).
- Deploy your app to Cloud Run (/run/docs).
- Persist your data with Firestore (/firestore).
- Store file uploads in Cloud Storage (/storage).
- Monitor your app using Google Cloud's operations suite (/stackdriver).

For other language-specific tutorials for building your apps, see the following guides:

- Deploying an app to Google Kubernetes Engine (/kubernetes-engine/docs/quickstarts/deploying-a-language-specific-app).

Costs

This tutorial uses the following billable components of Google Cloud:

- Cloud Run (/run/pricing)
- Cloud Storage (/storage/pricing)
- Firestore (/firestore/pricing)
- Google Cloud's operations suite (/stackdriver/pricing)

The tutorial is designed to keep your resource usage within the limits of Google Cloud's Always Free (/free/docs/frequently-asked-questions#always-free) tier. To generate a cost estimate based on your projected usage, use the pricing calculator (/products/calculator). New Google Cloud users might be eligible for a free trial (/free-trial).

When you finish this tutorial, you can avoid continued billing by deleting the resources you created. For more information, see Cleaning up (#clean-up).

Before you begin

1. Sign in (https://accounts.google.com/Login) to your Google Account.

   If you don't already have one, sign up for a new account (https://accounts.google.com/SignUp).

2. In the Cloud Console, on the project selector page, select or create a Cloud project.

★ **Note**: If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

Go to the project selector page (https://console.cloud.google.com/projectselector2/home/dashboar

3. Make sure that billing is enabled for your Google Cloud project. Learn how to confirm billing is enabled for your project (/billing/docs/how-to/modify-project).

4. To create a Firestore database in Native mode, complete the following steps:

   a. In the Cloud Console, go to the **Firestore viewer** page.
   Go to the Firestore viewer (https://console.cloud.google.com/firestore/data)

   b. From the **Select a Firestore mode** screen, click **Select Native Mode**.

   c. Select a location (/firestore/docs/locations#types) for your Firestore database. This location setting is the default Google Cloud resource location for your Cloud project (/firestore/docs/locations#default-cloud-location). This location is used for Google Cloud services in your Cloud project that require a location setting, specifically, your default Cloud Storage (/storage/docs) bucket and your App Engine (/appengine/docs) app.

⚠ Warning: After you set the default resource location for your Cloud project, you cannot change it.

d. Click **Create Database**.

5. Enable the Cloud Run, Cloud Storage JSON, Cloud Logging, and Error Reporting APIs.

   Enable the APIs (https://console.cloud.google.com/flows/enableapi?apiid=run.googleapis.com,storag

6. In Cloud Shell, open the app's source code.

   Go to Cloud Shell (/console/cloudshell/open?git_branch=master&git_repo=https://github.com/Google

   Cloud Shell provides command-line access to your Google Cloud resources directly from the browser.

7. To download the sample code and change into the app directory, click **Proceed**.

8. In Cloud Shell, configure the `gcloud` tool to use your new Google Cloud project:

```
# Configure gcloud for your project
gcloud config set project PROJECT_ID
```

   Replace ***PROJECT_ID*** with the Google Cloud project ID that you created using the Cloud Console.

   The `gcloud` command-line tool (/sdk/gcloud) is the primary way you interact with your Google Cloud resources from the command line. In this tutorial, you use the `gcloud` tool to deploy and monitor your app.

Running your app

1. Run the app:

```
GOOGLE_CLOUD_PROJECT=PROJECT_ID dotnet run
```

   Replace ***PROJECT_ID*** with the Google Cloud project ID that you created.

2. In Cloud Shell, click **Web preview**, and select **Preview on port 8080**. This opens a new window with your running app.

Deploying your app to Cloud Run

Google Cloud offers several options for running your code (/hosting-options). For this example, you use Cloud Run (/run/docs) to deploy a scalable app to Google Cloud. With zero server management, Cloud Run lets you focus on writing code. Plus, Cloud Run automatically scales to support sudden traffic spikes.

The Dockerfile tells Cloud Run how to run your app:

Bookshelf/Dockerfile
(https://github.com/GoogleCloudPlatform/getting-started-dotnet/blob/master/Bookshelf/Dockerfile)

tHub (https://github.com/GoogleCloudPlatform/getting-started-dotnet/blob/master/Bookshelf/Dockerfile)

```
FROM mcr.microsoft.com/dotnet/core/aspnet:2.1
COPY . /app
WORKDIR /app
ENTRYPOINT ["dotnet", "Bookshelf.dll"]
```

Dockerfiles (https://docs.docker.com/engine/reference/builder/) can be richer, but this configuration works for a lot of apps.

Cloud Run tells your app which port to listen to by setting the `PORT` environment variable. Bookshelf's `Program.cs` contains code to observe the `PORT` variable and listen on that port:

Bookshelf/Program.cs
(https://github.com/GoogleCloudPlatform/getting-started-dotnet/blob/master/Bookshelf/Program.cs)

łub (https://github.com/GoogleCloudPlatform/getting-started-dotnet/blob/master/Bookshelf/Program.cs)

```
using Google.Cloud.Diagnostics.AspNetCore;
using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;
using System;

namespace Bookshelf
{
    public class Program
    {
```

```
        public static void Main(string[] args)
        {
            CreateWebHostBuilder(args).Build().Run();
        }

        public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseGoogleDiagnostics(Startup.GetProjectId(), "Bookshelf", "0.01")
                .UseStartup<Startup>().UsePortEnvironmentVariable();
    }

    static class ProgramExtensions
    {
        // Google Cloud Run sets the PORT environment variable to tell this
        // process which port to listen to.
        public static IWebHostBuilder UsePortEnvironmentVariable(
            this IWebHostBuilder builder)
        {
            string port = Environment.GetEnvironmentVariable("PORT");
            if (!string.IsNullOrEmpty(port))
            {
                builder.UseUrls($"http://0.0.0.0:{port}");
            }
            return builder;
        }
    }
}
```

In your terminal window, deploy the app to Cloud Run using the `gcloud` tool:

1. Build the app locally.

   ```
   dotnet publish -c Release
   ```

2. Use Cloud Build to build a Docker container and publish to Container Registry.

   ```
   gcloud builds submit --tag gcr.io/PROJECT_ID/bookshelf \
       bin/Release/netcoreapp2.1/publish
   ```

3. Run the container with Cloud Run (fully managed).

```
gcloud run deploy bookshelf --region us-central1 --platform managed \
    --image gcr.io/PROJECT_ID/bookshelf --allow-unauthenticate
```

Your app is now viewable at the URL displayed in the output of `gcloud run`:

```
Service [bookshelf] revision [bookshelf-00001] has been deployed and is serving
https://bookshelf-lwuhslogjlnpofsxugoc.a.run.app
```

4. Copy the URL into your web browser to view the app.

★ **Note**: This SSL-protected domain is created automatically, and is useful for development. You can set up a custom domain (/run/docs/mapping-custom-domains) with Cloud Run as well.

For more information on deploying to Cloud Run, see the Cloud Run documentation (/run/docs).

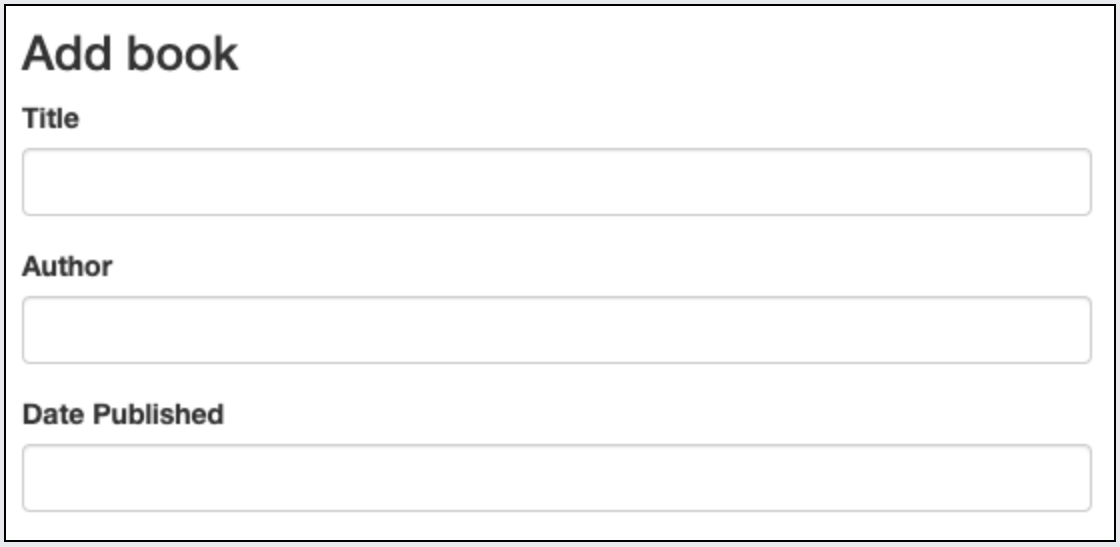
Persisting your data with Firestore

You cannot store information on your Cloud Run instances, because it is lost if the instance is restarted, and doesn't exist when new instances are created. Instead, you use a database that all your instances read from and write to.

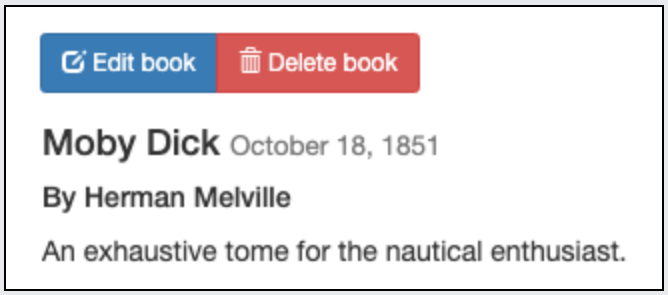
Google Cloud offers several options for storing your data (/products/storage). In this example, you use Firestore to store the data for each book. Firestore is a fully managed, serverless,

NoSQL document database that lets you store and query data. Firestore auto scales to meet your app needs, and scales to zero when you're not using it. Add your first book now.

1. To create a book for your deployed app, click **Add book**.

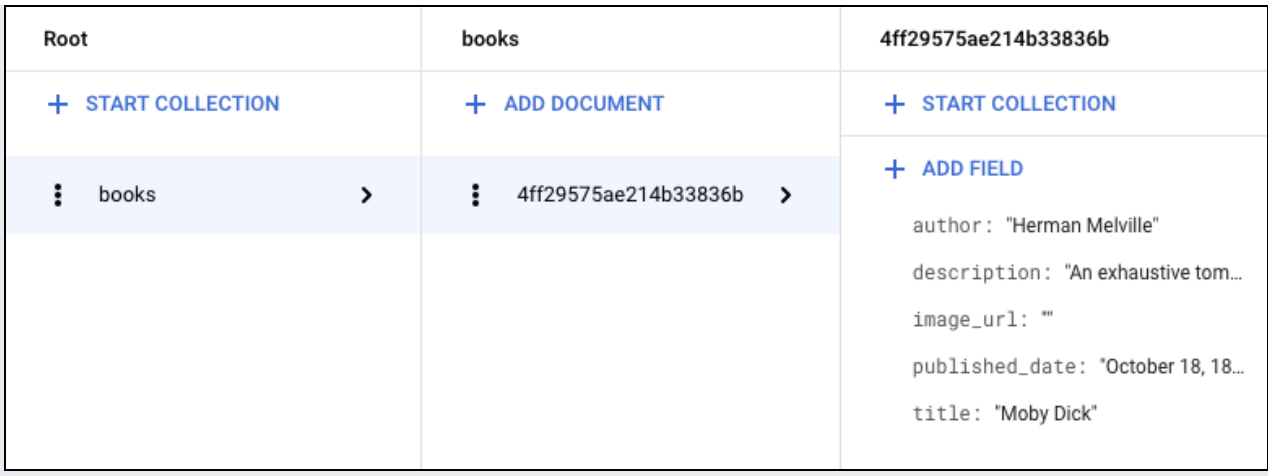


2. In the **Title** field, enter `Moby Dick`.
3. In the **Author** field, enter `Herman Melville`.
4. Click **Save**. There is now an entry to your Bookshelf app.



5. In the Cloud Console, go to **Cloud Firestore**. Go to Cloud Firestore (/console/firestore/data)

   The data appears in Firestore. The Bookshelf app stores each book as a Firestore document (/firestore/docs/data-model#documents) with a unique ID, and all these documents are stored in a Firestore collection (/firestore/docs/data-model#collections). For the purposes of this tutorial, the collection is called books.



Firestore stores the books by using the Firestore Client Library (http://googleapis.github.io/google-cloud-dotnet/#/docs/google-cloud/latest/firestore/readme). Here is an example of fetching a Firestore document:

Bookshelf/Models/FirestoreBookStore.cs (https://github.com/GoogleCloudPlatform/getting-started-dotnet/blob/master/Bookshelf/Models/FirestoreBookStore.cs)

.com/GoogleCloudPlatform/getting-started-dotnet/blob/master/Bookshelf/Models/FirestoreBookStore.cs)

```
using Google.Cloud.Firestore;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Bookshelf.Models
{
    class FirestoreBookStore : IBookStore
    {
        private FirestoreDb _firestore;
        private CollectionReference _books;

        public FirestoreBookStore(string projectId)
        {
            _firestore = FirestoreDb.Create(projectId);
            _books = _firestore.Collection("Books");
        }
```

For more information on using Firestore, see Adding data to Firestore (/firestore/docs/manage-data/add-data).

Storing file uploads in Cloud Storage

Now that you've added a book, it's time to add the book cover image. You cannot store files on your instances. A database isn't the right choice for image files. Instead, you use Cloud Storage.

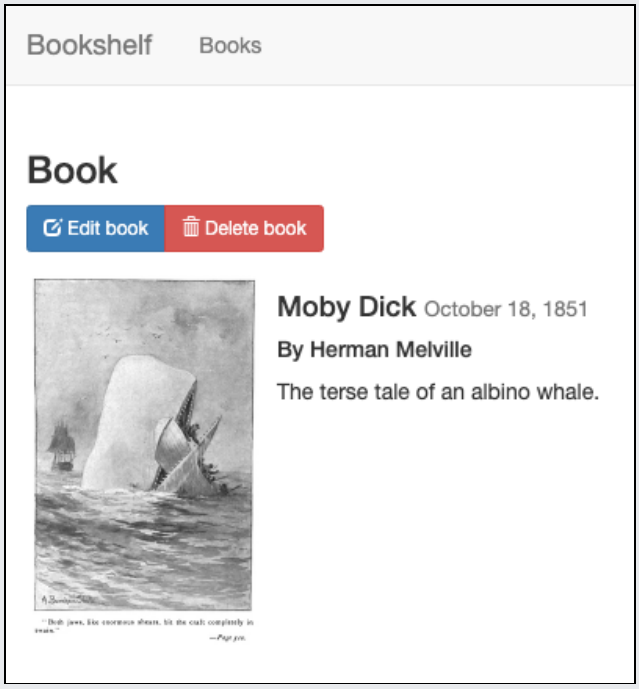
Cloud Storage (/storage/docs/creating-buckets) is the primary blob store for Google Cloud. You can use Cloud Storage to host app assets that you want to share across Google Cloud. To use Cloud Storage, you need to create a Cloud Storage bucket (/storage/docs/key-terms#buckets), a basic container to hold your data.

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

   Go to the Cloud Storage Browser page (/console/storage/browser)

2. Click **Create bucket**.
3. In the **Create bucket** dialog, enter a name for your bucket by appending your Google Cloud project ID to the string `_bucket` so the name looks like `YOUR_PROJECT_ID_bucket`. This name is subject to the bucket name requirements (/storage/docs/bucket-naming#requirements). All other fields can remain at their default values.
4. Click **Create**.
5. After your bucket is created, click **Edit book**, and select an image to upload as your book's cover. For example, you can use this public domain image:

6. Click **Save**. You're redirected to the homepage, where there is an entry to your Bookshelf app.



The bookshelf app sends uploaded files to Cloud Storage by using the Cloud Storage Client Library (http://googleapis.github.io/google-cloud-dotnet/#/docs/google-cloud/latest/storage/readme).

Bookshelf/Services/ImageUploader.cs (https://github.com/GoogleCloudPlatform/getting-started-dotnet/blob/master/Bookshelf/Services/ImageUploader.cs)

ıub.com/GoogleCloudPlatform/getting-started-dotnet/blob/master/Bookshelf/Services/ImageUploader.cs)

```
using Google.Cloud.Storage.V1;
using Microsoft.AspNetCore.Http;
using System;
using System.Threading.Tasks;
using System.Web;

namespace Bookshelf.Services
{
    public class ImageUploader
    {
        private readonly string _bucketName;
        private readonly StorageClient _storageClient;

        public ImageUploader(string bucketName)
```

```
        {
            _bucketName = bucketName;
            _storageClient = StorageClient.Create();
        }
```

Note: You can change the bucket name at any time by setting the **`$bucketId`** variable in the preceding code to a different name and redeploying your app by using **`gcloud app deploy`**.

For more information on using Cloud Storage, see the list of how-to guides (/storage/docs/how-to).

Monitoring your app using Google Cloud's operations suite

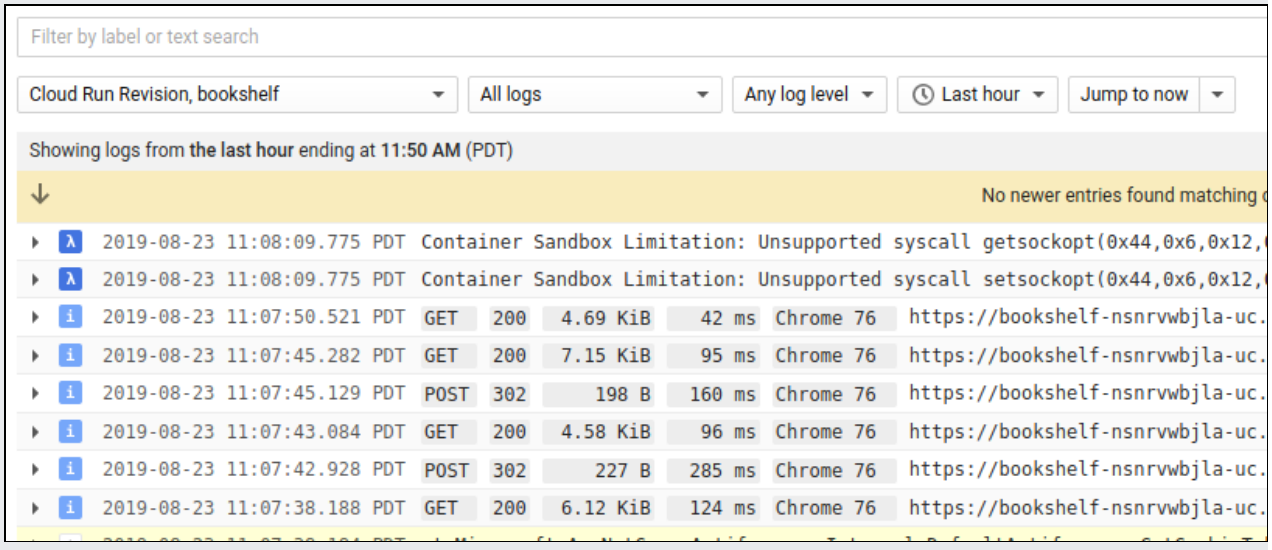
You've deployed your app and created and modified books. To monitor these events for your users, use Application Performance Management.

Monitor logs with Cloud Logging

1. In the Google Cloud, go to the **Logs Viewer**

   Go to Logs Viewer (/console/logs/viewer?resource=cloud_run_revision%2Fservice_name%2Fbookshe

   You can monitor your app in real time. If you have any issues with your app, this is one of the first places to look.



2. In the **Resource** drop-down list, select **Cloud Run Revision, bookshelf**.

Monitor errors with Error Reporting

1. In the Cloud Console, go to the **Error Reporting** page.

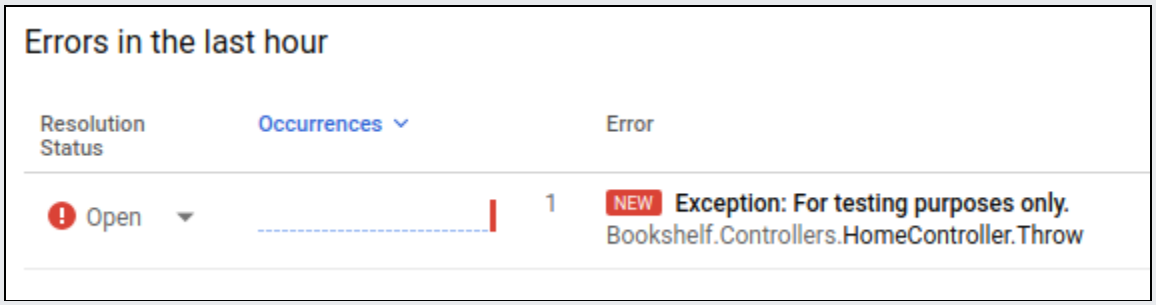
   Go to Error Reporting page (/console/errors)

   Error Reporting highlights errors and exceptions in your app and lets you set up alerting around them.

2. In your browser, go to the `/Home/Throw` URL in your app.
   For example, if your app is hosted at https://bookshelf-lwuhslogjlnpofsxugoc.a.run.app, then go to https://bookshelf-lwuhslogjlnpofsxugoc.a.run.app/Home/Throw.

   This generates a new test exception and sends it to Google Cloud's operations suite.

3. In the Cloud Console, return to the **Error Reporting** page, and in a few moments the new error is visible. Click **Auto Reload** so you don't need to manually refresh the page.



Note: Application Performance Management contains many tools to help debug and monitor your apps. For more information, see the tutorials (/stackdriver/docs/tutorials).

Cleaning up

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

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)

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

What's next

- Deploying an app to Google Kubernetes Engine (/kubernetes-engine/docs/quickstarts/deploying-a-language-specific-app)
- Handling sessions with Firestore (/dotnet/docs/getting-started/session-handling-with-firestore)

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/), and code samples are licensed under the Apache 2.0 License (https://www.apache.org/licenses/LICENSE-2.0). For details, see the Google Developers Site Policies (https://developers.google.com/site-policies). Java is a registered trademark of Oracle and/or its affiliates.

Last updated 2020-06-23 UTC.