Migrating to Microservices from a Monolithic App

[Python](https://cloud.google.com/appengine/docs/standard/python/microservice-migration) |[Java](https://cloud.google.com/appengine/docs/standard/java/microservice-migration) |**PHP** |[Go](https://cloud.google.com/appengine/docs/standard/go/microservice-migration)

When starting with a legacy, monolithic application, you must find parts that can be carved off and moved to separate microservices. Often, a well-structured monolithic app will have very natural divisions, and a service class will already function as an interface to a layer of data storage and business logic. Such classes are the ideal place to connect the client calls to the microservice.

Separating functionality in a monolithic app

You could take a couple approaches for how you separate your app:

* Look for business logic in your application that can be separated.
* Find code that is naturally isolated, such as by using static code analysis tools to identify the sections.
* Examine your application for logic that you might benefit from different scaling configuration settings or memory requirements than the rest of your application. This could possibly result in cost savings that might lead to better resource utilization.

You might need to refactor your code to remove unnatural dependencies. We recommend that you perform refactoring within your legacy code and deploy it to production before attempting to separate the app into separate services.

Common areas for microservices include the following:

* User or account information
* Authorization and session management
* Preferences or configuration settings
* Notifications and communications services
* Photos and media, especially metadata
* Task queue workers

Steps for Migrating an App

After a set of classes has been identified as a candidate to become a microservice, the next steps include:

* Leaving the existing code in place and operational in the legacy application to facilitate rollback.
* Creating a new code repository, or at least a sub-directory in your existing repository.
* Copying the classes into the new location.
* Writing a view layer that provides the HTTP API hooks and formats the response documents in the correct manner.
* Formulating the new code as a separate application (create an app.yaml).
* Deploying your new microservice as a service or separate project.
* Testing the code to ensure that it is functioning correctly.
* Migrating the data from the legacy app to the new microservice. See below for a discussion.
* Altering your existing legacy application to use the new microservices application.
* Deploying the altered legacy application
* Verifying that everything works as expected and that you don't need to roll back to the legacy application.
* Removing any dead code from the legacy application.

Migrating data on a live application

Data migration on a live application can be tricky and highly dependent on your situation. Often, to facilitate roll-forward and rollback, you will need to write code that populates both the old and new Cloud Datastore entities, possibly by using a temporary API on the microservice, and then write code that migrates the existing set of data, for example as a MapReduce. This process will usually involve some amount of temporary code and redundant data. Depending on the specifics of your situation, you may also need to execute a catch-up data migration after you release. Be careful not to overwrite newer data with older data.

While this seems like a lot of work, it's a common occurrence and is important to allow for rolling forward and rolling back in the event that the cutover to the new microservice does not succeed. You can remove your temporary code and delete the data from the old storage location only after you have verified that everything is migrated correctly and everything is operating as expected. Be sure to make backups along the way.

What's next

* Get an overview of [microservice architecture on App Engine](https://cloud.google.com/appengine/docs/standard/php/microservices-on-app-engine).
* Understand how to [create and name dev, test, qa, staging, and production environments with microservices in App Engine](https://cloud.google.com/appengine/docs/standard/php/creating-separate-dev-environments).
* Learn the [best practices for designing APIs to communicate between microservices](https://cloud.google.com/appengine/docs/standard/php/designing-microservice-api).
* Learn the [best practices for microservice performance](https://cloud.google.com/appengine/docs/standard/php/microservice-performance).

https://cloud.google.com/appengine/docs/standard/python/microservices-on-app-engine