Hello, world!

View on GitHub

This tutorial demonstrates some core features of Modal:

- You can run functions on Modal just as easily as you run them locally.
- Running functions in parallel on Modal is simple and fast.
- Logs and errors show up immediately, even for functions running on Modal.

Importing Modal and setting up

We start by importing `modal` and creating a `App`. We build up from our `App` to define our application.

```
import sys

import modal

app = modal.App("example-hello-world")
```

Defining a function

Modal takes code and runs it in the cloud.

So first we've got to write some code.

Let's write a simple function: log `"hello"` to standard out if the input is even or `"world"` to standard error if it's not, then return the input times itself.

To make this function work with Modal, we just wrap it in a decorator from our application `app`, `@app.function`.

```
@app.function()
def f(i):
    if i % 2 == 0:
        print("hello", i)
    else:
        print("world", i, file=sys.stderr)

    return i * i
```

Running our function locally, remotely, and in parallel

Now let's see three different ways we can call that function:

1. As a regular `local` call on your computer, with `f.local`
2. As a `remote` call that runs in the cloud, with `f.remote`
3. By `map` ping many copies of `f` in the cloud over many inputs, with `f.map`

We call `f` in each of these ways inside a `main` function below.

```
@app.local_entrypoint()
def main():
    # run the function locally
    print(f.local(1000))

    # run the function remotely on Modal
    print(f.remote(1000))

    # run the function in parallel and remotely on Modal
    total = 0
    for ret in f.map(range(20)):
        total += ret

    print(total)
```

Enter `modal run hello_world.py` in a shell and you'll see a Modal app initialize. You'll then see the `print`ed logs of the `main` function and, mixed in with them, all the logs of `f` as it is run locally, then remotely, and then remotely and in parallel.

That's all triggered by adding the `@app.local_entrypoint` decorator on `main`, which defines it as the function to start from locally when we invoke `modal run`.

What just happened?

When we called `.remote` on `f`, the function was executed **in the cloud**, on Modal's infrastructure, not locally on our computer.

In short, we took the function `f`, put it inside a container, sent it the inputs, and streamed back the logs and outputs.

But why does this matter?

Try doing one of these things next to start seeing the full power of Modal!

You can change the code and run it again

For instance, change the `print` statement in the function `f` to print `"spam"` and `"eggs"` instead and run the app again. You'll see that that your new code is run with no extra work from you — and it should even run faster!

Modal's goal is to make running code in the cloud feel like you're running code locally. That means no waiting for long image builds when you've just moved a comma, no fiddling with container image pushes, and no context-switching to a web UI to inspect logs.

You can map over more data

Change the `map` range from `20` to some large number, like `1170`. You'll see Modal create and run even more containers in parallel this time.

And it'll happen lightning fast!

You can run a more interesting function

The function `f` is obviously silly and doesn't do much, but in its place imagine something that matters to you, like:

- Running language model inference or fine-tuning
- Manipulating audio or images
- Collecting financial data to backtest a trading algorithm.

Modal lets you parallelize that operation effortlessly by running hundreds or thousands of containers in the cloud.