Fetching stock prices in parallel

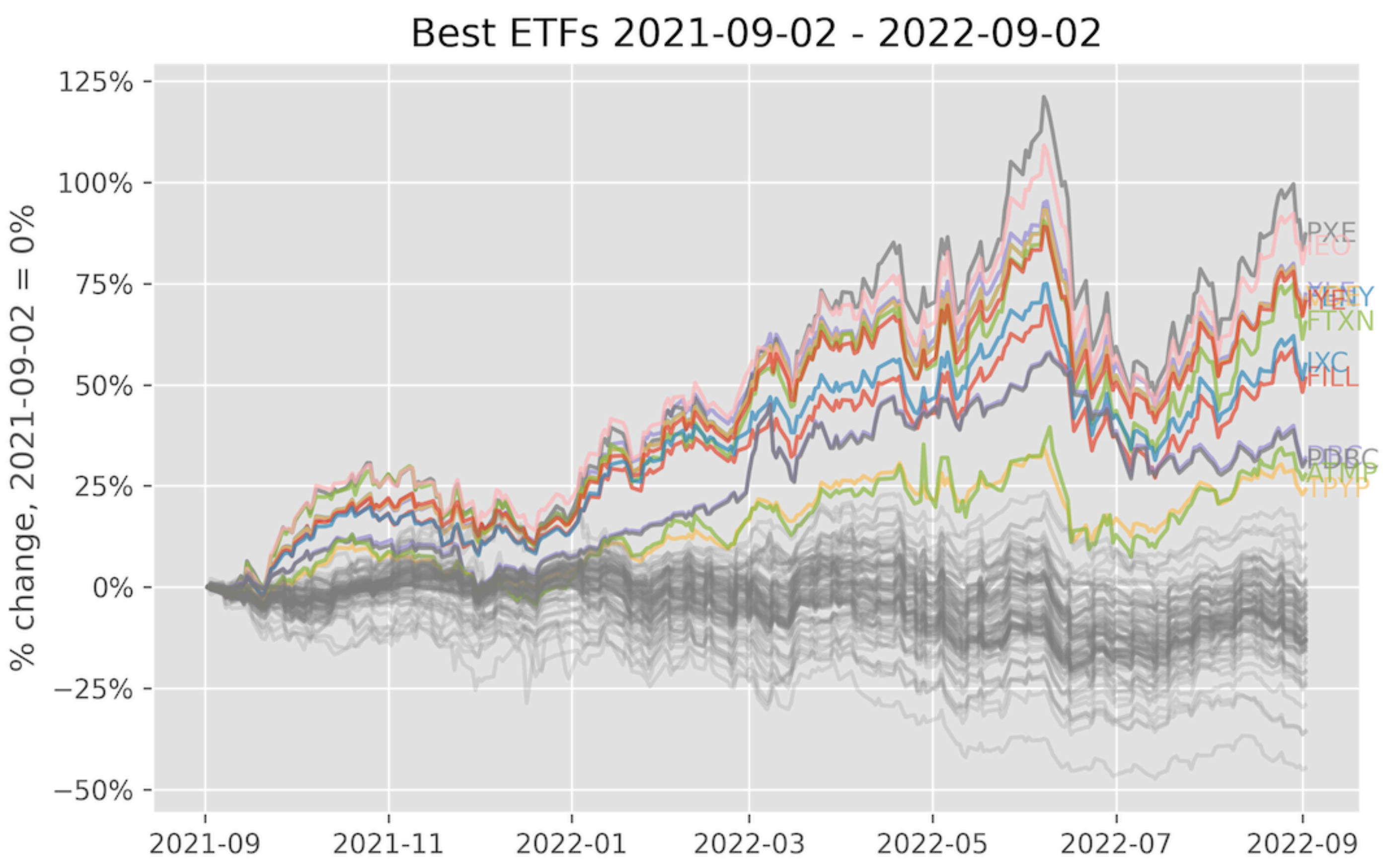
View on GitHub

This is a simple example that uses the Yahoo! Finance API to fetch a bunch of ETFs We do this in parallel, which demonstrates the ability to map over a set of items In this case, we fetch 100 stocks in parallel

You can run this script on the terminal with

```
modal run 03_scaling_out/fetch_stock_prices.py
```

If everything goes well, it should plot something like this:



Setup

For this image, we need

- `httpx` and `beautifulsoup4` to fetch a list of ETFs from a HTML page
- `yfinance` to fetch stock prices from the Yahoo Finance API
- `matplotlib` to plot the result

```
import io
import os

import modal

app = modal.App(
    "example-fetch-stock-prices",
    image=modal.Image.debian_slim().pip_install(
        "httpx~=0.24.0",
        "yfinance~=0.2.31",
        "beautifulsoup4~=4.12.2",
        "matplotlib~=3.7.1",
    ),
)
```

Fetch a list of tickers

The `yfinance` package does not have a way to download a list of stocks. To get a list of stocks, we parse the HTML from Yahoo Finance using Beautiful Soup and ask for the top 100 ETFs.

```
@app.function()
def get_stocks():
    import bs4
    import httpx

    headers = {
        "user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML
        "referer": "https://finance.yahoo.com/",
    }
    url = "https://finance.yahoo.com/etfs?count=100&offset=0"
    res = httpx.get(url, headers=headers)
    res.raise_for_status()
    soup = bs4.BeautifulSoup(res.text, "html.parser")
    for td in soup.find_all("td", {"aria-label": "Symbol"}):
        for link in td.find_all("a", {"data-test": "quoteLink"}):
            symbol = str(link.next)
            print(f"Found symbol {symbol}")
            yield symbol
```

Fetch stock prices

Now, let's fetch the stock data. This is the function that we will parallelize. It's fairly simple and just uses the `yfinance` package.

```
@app.function()
def get_prices(symbol):
    import yfinance

    print(f"Fetching symbol {symbol}...")
    ticker = yfinance.Ticker(symbol)
    data = ticker.history(period="1Y")["Close"]
    print(f"Done fetching symbol {symbol}!")
    return symbol, data.to_dict()
```

Plot the result

Here is our plotting code. We run this in Modal, although you could also run it locally. Note that the plotting code calls the other two functions. Since we plot the data in the cloud, we can't display it, so we generate a PNG and return the binary content from the function.

```python
@app.function()
def plot_stocks():
    from matplotlib import pyplot, ticker

    # Setup
    pyplot.style.use("ggplot")
    fig, ax = pyplot.subplots(figsize=(8, 5))

    # Get data
    tickers = list(get_stocks.remote_gen())
    if not tickers:
        raise RuntimeError("Retrieved zero stock tickers!")
    data = list(get_prices.map(tickers))
    first_date = min((min(prices.keys()) for symbol, prices in data if prices))
    last_date = max((max(prices.keys()) for symbol, prices in data if prices))

    # Plot every symbol
    for symbol, prices in data:
        if len(prices) == 0:
            continue
        dates = list(sorted(prices.keys()))
        prices = list(prices[date] for date in dates)
        changes = [
            100.0 * (price / prices[0] - 1) for price in prices
        ]  # Normalize to initial price
        if changes[-1] > 20:
            # Highlight this line
            p = ax.plot(dates, changes, alpha=0.7)
            ax.annotate(
                symbol,
                (last_date, changes[-1]),
                ha="left",
                va="center",
                color=p[0].get_color(),
                alpha=0.7,
            )
        else:
            ax.plot(dates, changes, color="gray", alpha=0.2)

    # Configure axes and title
    ax.yaxis.set_major_formatter(ticker.PercentFormatter())
    ax.set_title(f"Best ETFs {first_date.date()} - {last_date.date()}")
    ax.set_ylabel(f"% change, {first_date.date()} = 0%")

    # Dump the chart to .png and return the bytes
    with io.BytesIO() as buf:
        pyplot.savefig(buf, format="png", dpi=300)
        return buf.getvalue()
```

Entrypoint

The entrypoint locally runs the app, gets the chart back as a PNG file, and saves it to disk.

```
OUTPUT_DIR = "/tmp/"


@app.local_entrypoint()
def main():
    os.makedirs(OUTPUT_DIR, exist_ok=True)
    data = plot_stocks.remote()
    filename = os.path.join(OUTPUT_DIR, "stock_prices.png")
    print(f"saving data to {filename}")
    with open(filename, "wb") as f:
        f.write(data)
```