Snakes of the World, Unite!

Modern Python Workers Seizing Production

Dr Maciej Gryka
RainforestQA
PyData Berlin, 17th Oct 2018

he World, Unite!

Modern Wor ers Seizing $\underbrace{}_{\text{the means of}}$ Production

Dr Maciej Gryka
RainforestQA
PyData Berlin, 17th Oct 2018

So what’s the problem?

1. You want to run long-lived (minutes, hours) Python workers.
2. Bursting should be easy: periods with no traffic and bursts of e.g. 1000 simultaneous jobs, without unnecessary infrastructure costs.
3. Use non-trivial code and things like C extensions, compiling some stuff yourself etc. Maybe you want some ruby in your Python?
4. The jobs are stateless (almost).

How would you make this happen?

Our setup

1. Dockerize the app
2. Use a bunch of AWS: ECS, SQS, Batch
3. CI (CircleCI in our case) for deployment

Workflow

1. Something happens in the web app,

Workflow

1. Something happens in the web app,
2. …which triggers the jobs using `boto` (the details of the jobs are already defined in Batch at that point).

Workflow

1. Something happens in the web app,
2. …which triggers the jobs using `boto` (the details of the jobs are already defined in Batch at that point).
3. AWS Batch launches Dockerized workers on ECS. It automatically up- and down-scales EC2 instances to host the containers as needed.

Workflow

1. Something happens in the web app,

2. …which triggers the jobs using `boto` (the details of the jobs are already defined in Batch at that point).

3. AWS Batch launches Dockerized workers on ECS. It automatically up- and down-scales EC2 instances to host the containers as needed.

4. The jobs are stateless in our case (almost: we store some artifacts on S3) and don’t know about each other. If you want state, use eternal storage.

Workflow

1. Something happens in the web app,

2. …which triggers the jobs using `boto` (the details of the jobs are already defined in Batch at that point).

3. AWS Batch launches Dockerized workers on ECS. It automatically up- and down-scales EC2 instances to host the containers as needed.

4. The jobs are stateless in our case (almost: we store some artifacts on S3) and don’t know about each other. If you want state, use eternal storage.

5. The job definition and the container registry is updated as part of CI when deploying the worker (we use CircleCI and CloudFormation).

Our use case

1. Rainforest provides functional testing as a service.
2. Some parts of testing are really boring for humans, so we want to automate them, while leaving space for humans in the loop.
3. The simplest approach is record-replay (with a twist: many people to record!).
4. Our Python workers replay testers’ actions - so they’re human-speed, i.e. slow.
5. This is structurally similar to what OpenAI Gym does for Reinforcement Learning - down to the vnc driver :)

Advantages

1. No cost when not in use.

2. Effectively infinite scalability (for us - totally removed “not having enough capacity” from our list of worries).

3. Workers are spawned fresh when needed and killed once done. (i.e. you can go wild with your long-running memory leaks ;))

4. The OpenAI Gym-inspired setup is awesome for our scenario (experimenting with new agents is trivial).

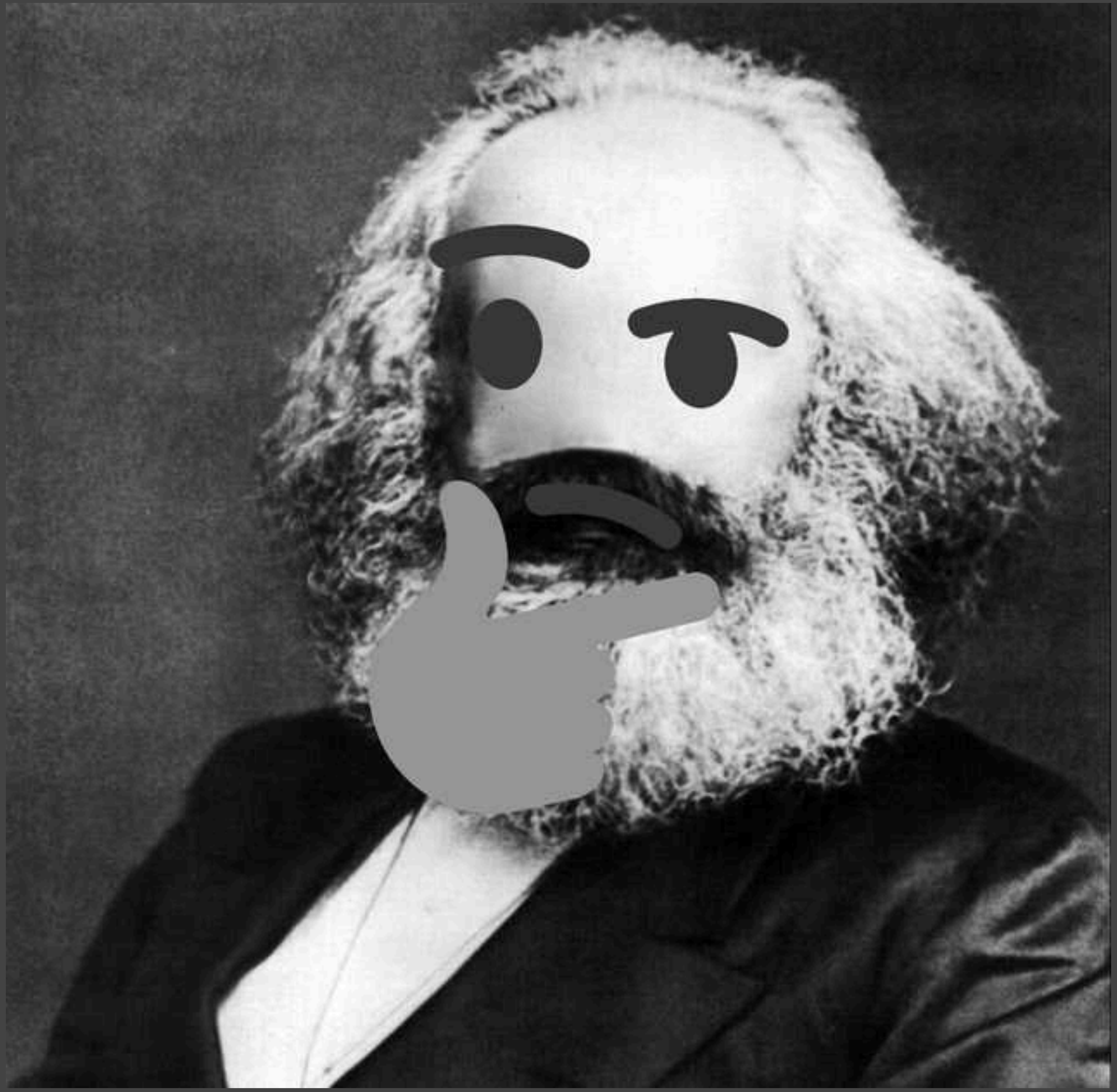
Downsides

1. Increased complexity from our previous solution: a single worker running constantly and doing work serially.

2. Setting up logging takes a bit more effort (to stay sane, you pretty much have to pipe out to somewhere else; shoutout to LogDNA).

3. Harder to debug, especially on the system boundaries.

Dare I do a demo?



Bonus: cool tools ❤️

1. Docker 🐳 is pretty great (I wasn’t kidding about the ruby 💎 thing).
2. pyenv https://github.com/pyenv/pyenv
3. Pipenv ✨🍰✨ https://pipenv.readthedocs.io/
4. black 🖤 https://github.com/ambv/black

Also, we're hiring!

rainforest

https://www.rainforestqa.com/careers/

We mostly do not-infrastructure work :)

If you're interested in ML generally and especially Computer Vision applied to non-photographs and Reinforcement Learning, ask me!

We're fully remote - I live in Postdam and get to work from home, but also travel to San Francisco a couple of times per year.