در اینجا توضیحات مختصری در مورد نحوه پیاده‌سازی و تقسیم کار پروژه‌ی رایانش ابری می‌دهیم.

تقسیم کار)

اگر پروژه را به دو بخش:

- dockerize کردن و آماده‌سازی image و بالا آوردن containerها
- پیاده‌سازی load balancer و client و server

تقسیم کنیم؛ قسمت اول بیشتر توسط ماهان زنده‌دل انجام شده و قسمت دوم توسط علی نظری انجام شده است. البته در تمامی بخش‌ها همکاری زیادی صورت گرفته و یادگیری و هم‌افزایی صورت گرفته است.

پیاده‌سازی)

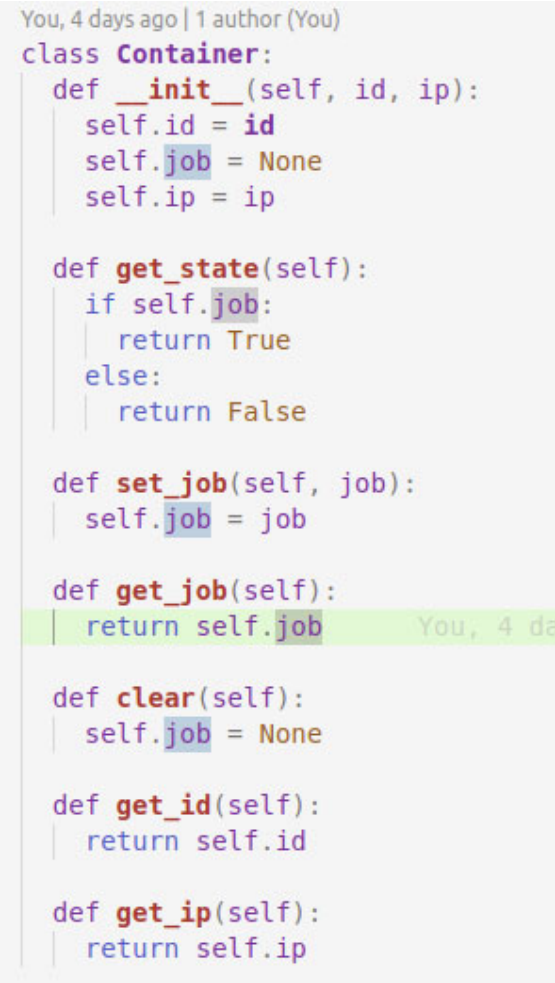
قسمت‌های مختلفی در این پروژه در نظر گرفته شده است. ابتدا در مورد client موجود در کامپیوتر میزبان صحبت می‌کنیم.

clientی در کامپیوتر میزبان وجود دارد که دستورات کاربر را دریافت می‌کند و jobهای لازم بین containerهای موجود برای اجرا تقسیم می‌کند.

هر job به صورت زیر است:

```
You, 4 days ago | 1 author (You)
class Job:
  def __init__(self, id, code, inp, out):
    self.id = id
    if not code.endswith(".py"):
      code = f"programs/{code}.py"
    self.code = os.path.join(WORKING_DIRECTORY, code)
    self.input = os.path.join(WORKING_DIRECTORY, inp)
    self.output = os.path.join(WORKING_DIRECTORY, out)

  def __repr__(self):
    return str(self.id)

  def get_code(self):
    return self.code

  def get_input(self):
    return self.input

  def get_output(self):
    return self.output
```

containerها نیز دارای مشخصات زیر هستند:



در ادامه به جزئیات استفاده از این کلاس‌ها در پیاده‌سازی می‌پردازیم.

jobها با کمک load balancer زیر به containerهای مختلف dispatch می‌شوند:

```
def dispatch(container):
  global containers
  job = container.get_job()
  req = requests.get(f"http://{container.get_ip()}:50051/", data=json.dumps({
      "code": job.get_code(),
      "input": job.get_input(),
      "output": job.get_output()
  }))
  print(
      f"• job-{container.get_job()} completed with result {req.json()['status']}")
  container.clear()


def load_balance():
  global containers, jobs
  while not stop:
    for container in containers:
      if len(jobs) != 0 and not container.get_state():
        job = jobs.pop()
        print(f"○ Assigning job-{job} to container-{container.get_id()}")
        container.set_job(job)
        threading.Thread(target=dispatch, args=(container, )).start()
```

در نهایت یک سرور flask پیاده‌سازی شده که درون containerها قرار می‌گیرد و jobهای مختلف به آن برای اجرا سپرده می‌شود:

```
@app.route("/", methods=["GET"])
def req():
  try:
    body = json.loads(request.data)
    process = subprocess.Popen(["python", f"/home/ali{body['code']}", f"/home/ali{body['input']}"],
                               stdout=subprocess.PIPE,
                               stderr=subprocess.PIPE)
    stdout, stderr = process.communicate()
    if stdout:
      file = open(f"/home/ali{body['output']}", "w")
      file.write(stdout.decode("utf-8"))
      file.close()
      return {"status": "200"}
    else:
      return {"status": "500"}

  except Exception as e:
    print("error", e)
    return json.dumps({"error": str(e)})
```

سپس این سرور flask به صورت زیر dockerize و آماده deploy می‌شود:

```
You, 4 days ago | 1 author (You)
FROM python:3.8-slim

ENV DockerHome=/home

COPY requirements.txt ${DockerHome}

WORKDIR ${DockerHome}

RUN pip install --upgrade pip      You, 4 d
RUN pip install -r requirements.txt

COPY . ${DockerHome}

EXPOSE 50051

CMD cd ${DockerHome} && python server.py
```

```
version: "3"
services:
    baba:
        build:
            context: .
            dockerfile: Dockerfile
        ports:
            - 50051-50053:50051
        volumes:
            - /:/home/ali      You, 4 da
        image: baba-img
        networks: [default]
networks:
    default:
        driver: bridge
        ipam:
            config:
              - subnet: 172.18.0.0/24
```

همان‌طور که در فایل docker-compose قابل مشاهده است، امکان دسترسی به storage کامپیوتر به containerها داده‌شده که بتوانند ورودی‌ها را بخوانند و خروجی را نیز در آن ذخیره کنند.