به نام خدا

تحقیق میان ترم برنامه نویسی وب

علی احمدی کافشانی

حامد عبدی

در این مقاله بر آنیم که در اوبونتو یک اپلیکیشن نود را به همراه سلری[1] و Node-schedule به همراه پایگاه داده ربیت پیاده‌سازی کنیم.

هدف ما

فرض کنیم که برنامه ما پنج کار مختلف دارد که باید هر کدام از این موارد را در زمان های خاصی یا به صورت دوره ای مثلاً هر ده دقیقه انجام دهد. برای انجام این مورد و برنامه‌ریزی این کارها میتوان از پکیج Node-schedule استفاده کرد.
مورد بعدی انجام این کارها است. آیا خود برنامه باید این مورد را انجام دهد؟
اگر این اتفاق بیافتد سرعت خود برنامه بشدت کم می‌شود و چون برنامه ما باید در لحظه سرویس دهد و از طریق api ها هم اطلاعات می‌دهد نمیتوان به این سؤال جواب مثبت داد.
در این مورد ما به سراغ celery می‌رویم که به طور ساده یک صف و تعدادی کارگر است. با استفاده از کارگرهای سلری ما کنترل بیشتر روی انجام تسک ها داریم و به طور مثال در صورت پیاده‌سازی صحیح میتوان با افزایش تعداد کارگرها سرعت برنامه را بهبود داد.

سلری

سلری یک صف غیرهمزمان[3]برای انجام کارهای ما است. به عبارتی دیگر ما یک سری کار[7] به سلری میدهیم و این برنامه آن‌ها را به صف تبدیل کرده و یکی پس از دیگری انجام می دهد. با این روش می‌توان چند برنامه کارگر[5] را همزمان اجرا کرد و این کارگرها هر کدام به ترتیب یک کار از صف بخوانند و آن را انجام بدهند. منظور از کارگر صرفاً یک برنامه پردازشگر برای انجام یک کار است.

حال یک برنامه نود می نویسیم که بتوانیم رفتار سلری را مشاهده کنیم و کارهای غیرهمزمان را در آن وارد کنیم.
در مرحله اول ابتدا برنامه را ایجاد می‌کنیم و کتابخانه‌های مورد نیاز را به آن اضافه میکنیم.

```
$ npm init
$ npm install express
$ npm install celery-node
$ npm install node-schedule
```

1. Celery
2. rabbitmq server
3. asynchronous
4. task
5. worker

توجه شود از اکسپرس فقط برای راه اندازی یک وب سرور ساده و دیدن بهتر نتایج استفاده خواهد شد.
حالا در فایل package.json در بدنه Scripts یک خط به صورت زیر اضافه می کنیم تا بتوانیم برنامه را اجرا کنیم.

```
"develop": "node index.js"
```

می‌توان خطوط دیگر scripts را حذف نمود.
حال فایل index.js را تعریف میکنیم و یک وب سرور ساده می نویسیم.

```
const express = require('express')
const app = express()
const port = 3000

app.get('/add-task', (req, res) => {
res.send("Success")
})

app.listen(port, () => {
console.log("Node Started at port " + port)
})
```

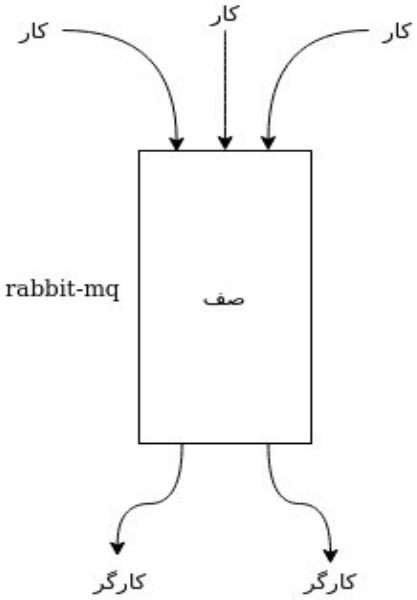
حالا میتوان با دستور

npm run develop

و رفتن به آدرس `localhost:3000` برنامه را اجرا کرد و از نصب درست آن اطمینان حاصل کرد.

اتصال به rabbit-mq

حال به اتصال سلری با برنامه نوشته شده می پردازیم.
ابتدا باید در سیستم خود سرور ربیت را راه اندازی کنیم. ربیت به عنوان پایگاه داده ما برای نگهداری کارهای موجود و در صورت نیاز نگهداری نتایج این کارها استفاده می شود.
شکل زیر نحوه جاگیری و چرایی وجود ربیت در معماری ما را توضیح می‌دهد.



به طور کلی ربیت یک پایگاه داده است به صورت صف است. یعنی هر دیتایی که زودتر وارد شود زودتر خارج میشود(FIFO). از این قابلیت ربیت برای ایجاد یک صف و اولویت دادن به کارهای ورودی میتوان استفاده کرد.
میتوان با استفاده از داکر یک سرور ربیت را راه اندازی کرد

```
$ docker run -it --rm --name rabbitmq -p 5672:5672 -p 15672:15672
rabbitmq:3.9-management
```

با اینکه اشاره شد که ربیت را با داکر راه اندازی کنیم ولی میتوان آن را بر روی اوبنتو خود نیز نصب کرد که در اینجا نحوه راه اندازی آن توضیح داده شده است.
حال میتوان با استفاده از پکیج node-celery که نصب کردیم با ربیت ارتباط بگیریم.

```
const celery = require('celery-node');

const celery_client = celery.createClient(
"amqp://",
"amqp://"
);
```

سپس میتوان مسیری که برای ایجاد تسک درست کرده‌ایم را تکمیل کنیم:

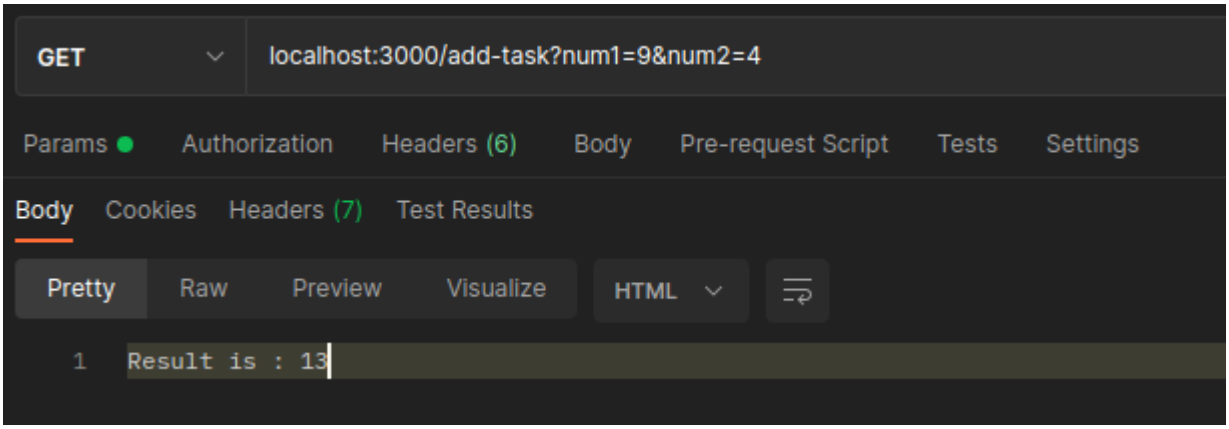
```
app.get('/add-task', (req, res) => {
const num1 = Number(req.query.num1);
const num2 = Number(req.query.num2);

const task = celery_client.createTask("tasks.add");
const result = task.applyAsync([num1, num2]);
result.get().then(data => {
res.send("Result is : " + data);
});
})
```

در این مسیر ابتدا از پارامترهای num1 و num2 که در درخواست GET می‌آید دو مقدار موردنظر را میخوانیم و با استفاده از صف سلری این دو مقدار را با هم جمع می کنیم.

حال باید یک اتصال برای کارگر برقرار کرد تا تسک را از ربیت بخواند و کار مورد نظر را انجام دهد:

```
const worker = celery.createWorker(
"amqp://",
"amqp://"
);
worker.register("tasks.add", (a, b) => a + b);
worker.start();
```

حالا میتوان با استفاده از مسیر add-task یک کار جدید به سلری اضافه نمود و کارگر از ربیت این کار را میخواند و انجام می دهد.



توجه شود که در این پروژه یک کار ساده مانند انجام یک جمع مورد نظر ما بود و میتوان بسته به پروژه شما کارهایی که با سلری انجام می‌شود پیچیده‌تر شود.
از این ویژگی میتوان برای انجام کارهای سنگین استفاده شود به طوری که کاربر درگیر زمان طولانی پردازش نشود. سؤالی که مطرح می‌شود این است که میتوان از پردازش موازی نود استفاده کرد و دیگر نیاز به استفاده از کتابخانه جدید در کد نباشد اما باید توجه کرد برای انجام بعضی از کارها به صورت ترتیبی و همچنین اولویت دادن به کارها(که با صف بودن سلری این کار انجام میشود) از این کتابخانه میتوان استفاده کرد.
در ادامه کار ما می خواهیم این کار را به صورت دوره انجام دهیم و هر 30 ثانیه دو عدد 2 و 5 را جمع بزنیم.

```
const schedule = require('node-schedule');

const job = schedule.scheduleJob("*/30 * * * * *", function () {
const task = celery_client.createTask("tasks.add");
const result = task.applyAsync([2, 5]);

result.get().then(data => {
console.log("Result is : " + data);
});
});
```

در قسمت `"*/30 * * * * *"` ما مشخص کردیم که این کار هر 30 ثانیه انجام شود که میتوان از اینجا این مقدار را به هر مقدار موردنظر و حتی تاریخ موردنظر تغییر داد.
از این ویژگی میتوان برای کارهای متعددی استفاده کرد. به طور مثال اگر ما یک در یک بازی جدول امتیازاتی داشته باشیم که نخواهیم کاملاً به روز باشد و مثلاً هر یک ساعت یکبار بروزرسانی شود از این ویژگی نود میتوان استفاده کرد.

کد نهایی در اینجا موجود هست.