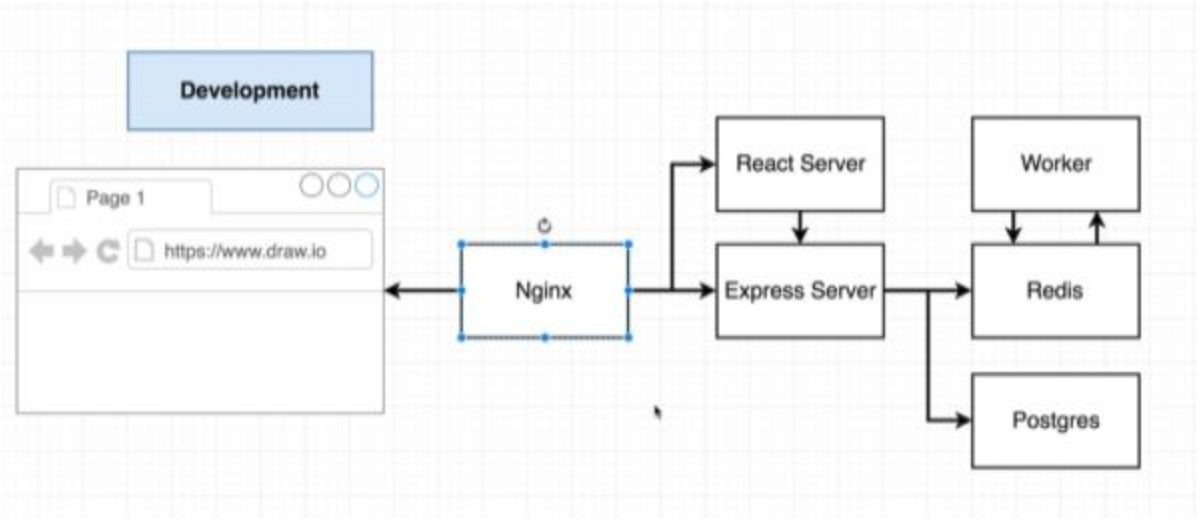
توضیح مختصری در مورد پروژه کارشناسی

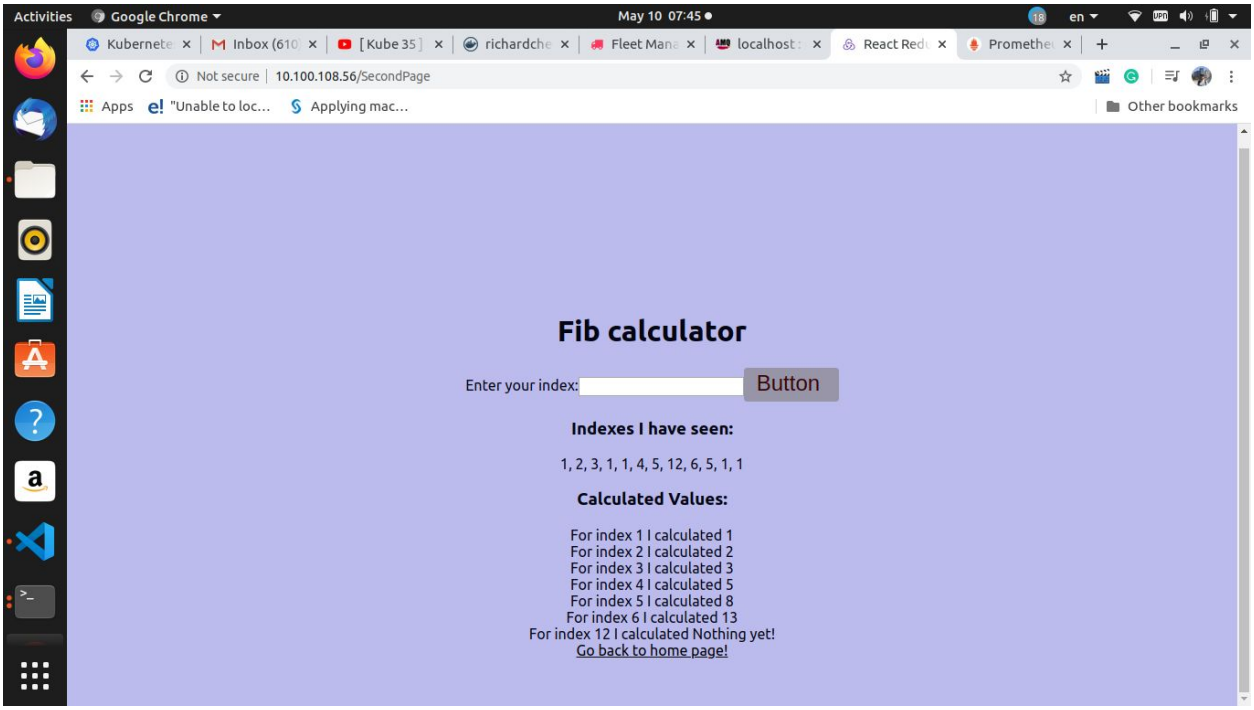
محمد پویا خرسندی

بخش ۱ : توضیح سرویس ها

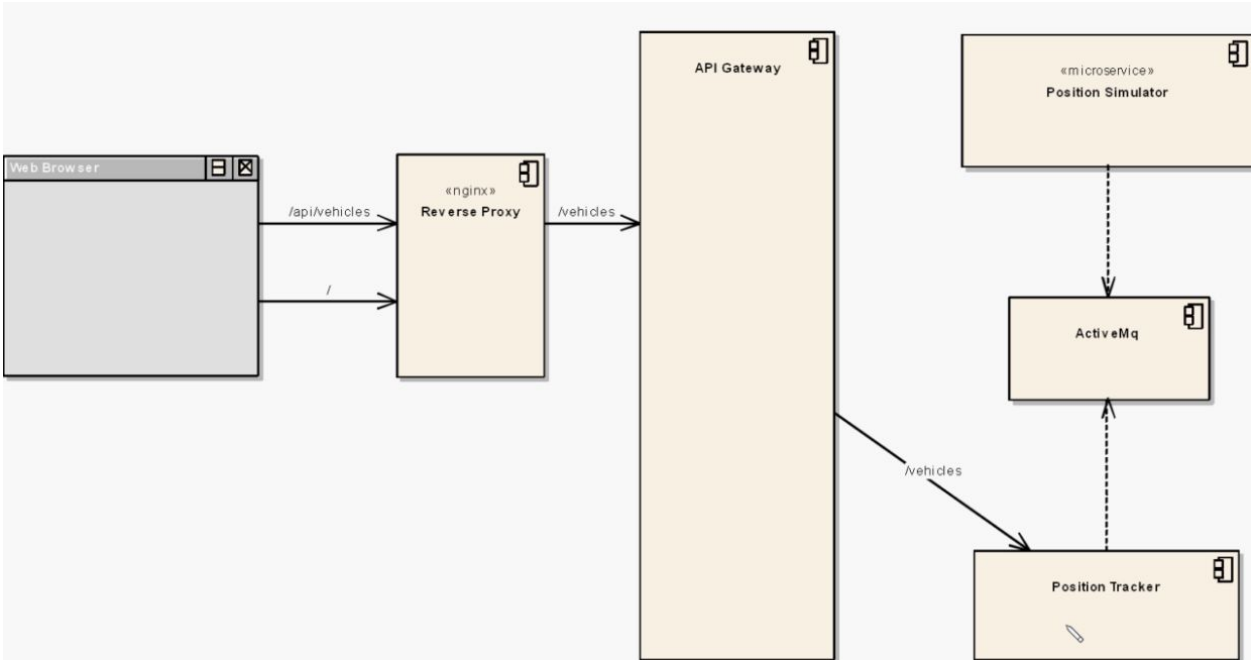
برای این پروژه من دو سرویس را مد نظر دارم که اولی سرویس بسیار ساده ایست که فیبوناچی را محاسبه میکند و برمیگرداند و ما این سرویس را به چند مایکرو سرویس تقسیم میکنیم . یک فرانت اند و بک اند وجود دارد که فرانت اند با زبان react است و بک اند آن با استفاده از فریم ورکی از node js به نام express نوشته شده است. به علاوه بر این دو یک مایکرو سرویس دیگر هم داریم که کار محاسبه کردن فیبوناچی را انجام میدهد. به این صورت که داده که در فرانت اند کاربر وارد کرده است سرور در پایگاه داده redis میگذارد و این مایکرو سرویس از آن پایگاه داده بر میدارد و میخواند و محاسبه میکند و نتیجه را به کاربر نمایش میدهد. ما در این جا از دو پایگاه داده استفاده کرده ایم به این منظور که بتوانیم مایکرو سرویس های بیشتری داشته باشیم. پایگاه داده دیگر mongo db است که عدد هایی که برای محاسبه میخواهیم به مایکرو سرویس worker بدهیم؛ در آنجا ذخیره میکنیم. سپس در فرانت اند نشان میدهیم که چه اعدادی کاربر وارد کرده است. شمای کلی این سرویس به صورت زیر است. و در اخر با استفاده از nginx روتینگ را انجام میدهیم که هر در خواست به کدام مایکرو سرویس انتقال پیدا کند.



و شمای فرانت اند این سرویس به این شکل است.

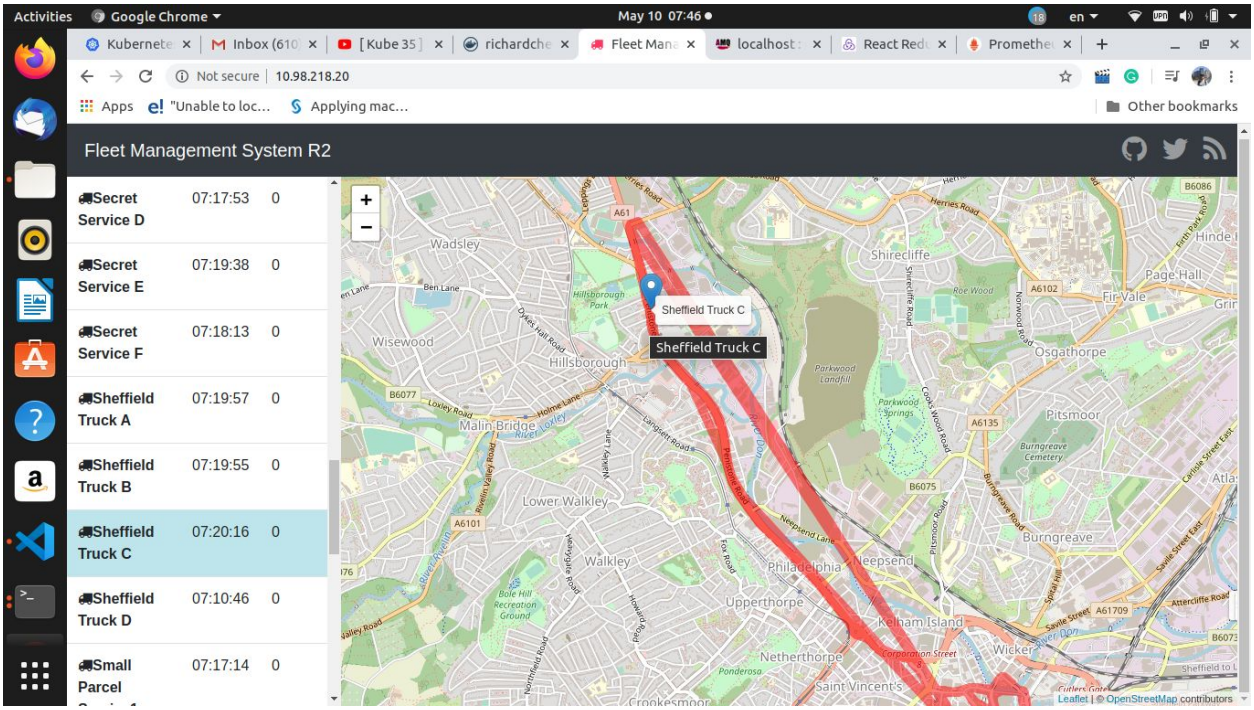


سرویس بعدی که ما پیاده سازی کردیم سرویس نشان دادن مسیر های طی شده توسط کامیون ها است که این سرویس را هم ما به چند مایکرو سرویس تقسیم کرده ایم. ابتدا شمای کلی مایکرو سرویس ها را ببینیم سپس توضیح بدهیم.



خب همون طور که مشاهده میکنیم ۵ مایکرو سرویس وجود دارد به علاوه یک پایگاه داده که به مایکرو سرویس position tracker متصل میشود برای نگه داری مسیر های کامیون ها. اولین مایکرو سرویس position simulator هست که مسیر های کامیون ها را شبیه سازی میکند و از یک فایلی که در مسیر های کامیون ها شبیه سازی شده است میخواند و به مایکرو سرویس ActivrMq میدهد و این مایکرو سرویس نقاط تولید شده برای کامیون ها را نگه میدارد و سپس مایکرو سرویس position tracker محاسباتی بر روی این نقاط انجام میدهد و مسیر های طی شده توسط کامیون ها را ذخیره و بازیابی میکند. فرانت اند ما یک مایکرو سرویس دیگری است که از طریق مایکرو سرویس api-gateway به سایر مایکرو سرویس ها متصل میشود و به این صورت از هم مجزا میشوند.

شمای این سرویس هم به صورت زیر است.



همانطور که مشاهده میشود لیستی از کامیون ها در سمت چپ امده است و با کلیک کردن هر کامیون مسیر طی شده کامیون مشخص میشود.

بعد از پیاده سازی این مایکرو سرویس ها با استفاده از داکر فایل ها ایمج های هر مایکرو سرویس را تولید کردیم و با نوشتن فایل های yaml برای deployment و نتورکینگ کانتینتر ها این مایکرو سرویس ها را در پلتفرم کوبرنتیز پیاده سازی کردیم که شمای کلی این مایکرو سرویس ها به این گونه است.

```
root@mpouyakh-m:/home/mpouyakh# kubectl get pods -n default
NAME                                              READY   STATUS    RESTARTS   AGE
api-gateway-5df59ddbc8-ntkh4                      1/1     Running   0          4h37m
client-deployment-74b64b89cb-7sm6f                1/1     Running   0          25h
mongodb-7dc4596644-p64z9                          1/1     Running   0          29h
nfs-client-provisioner-1586887262-868646f4b-jw5mr 1/1     Running   29         9d
position-simulator-7f4d479d95-54xvv               1/1     Running   0          4h39m
position-tracker-54fb5494bf-sc4nd                 1/1     Running   0          4h38m
postgres-deployment-75f7b55b5f-68nhr              1/1     Running   0          9d
pouya-nginx-ingress-controller-748d848f7d-zwtkv   1/1     Running   0          5h2m
pouya-nginx-ingress-default-backend-95b66b7b-cjgkk 1/1     Running   0          3d6h
prometheus-alertmanager-77ccc9cf68-559m9          2/2     Running   0          9d
prometheus-kube-state-metrics-6756bbbb8-ssbd6     1/1     Running   0          9d
prometheus-node-exporter-cnnzs                    1/1     Running   0          9d
prometheus-node-exporter-ldwg9                    1/1     Running   26         9d
prometheus-pushgateway-987dc77f-tb9c4             1/1     Running   0          9d
prometheus-server-5498c9576f-vk58m                2/2     Running   1          5h1m
queue-ff85789bd-m2l6x                             1/1     Running   0          4h38m
redis-deployment-5f458546b8-qkhfj                 1/1     Running   0          9d
server-deployment-ccf98c97d-xl2v6                 1/1     Running   0          24h
webapp-785d5b86bf-8j8j8                           1/1     Running   0          4h38m
worker-deployment-6f5674d765-mqqv8                1/1     Running   0          20h
```

بخش ۲ : auto scaling

در این بخش ما از metrics-server برای برای پیدا کردن مصرف cpu , memory استفاده کردیم و این package را با استفاده از Helm package manager که به کلاستر ما متصل شده است نصب کرده ایم. Metrics-server برای هر پاد میزان مصرفی cpu و memory را نشان میدهد. که به این صورت است.

Pods

Name	Namespac	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Age
api-gateway-5df59ddbc8-ntkh4	default	app: api-gateway pod-template-hash: 5df59ddbc8	mpouyakh	Running	0	24.00m	193.83Mi	4 hours
position-tracker-54fb5494bf-sc4nd	default	app: position-tracker pod-template-hash: 54fb5494bf	mpouyakh	Running	0	28.00m	239.49Mi	4 hours
webapp-785d5b86bf-8j8j8	default	app: webapp pod-template-hash: 785d5b86bf	mpouyakh	Running	0	1.00m	6.47Mi	4 hours
queue-ff85789bd-m2l6x	default	app: queue pod-template-hash: ff85789bd	mpouyakh	Running	0	84.00m	233.64Mi	4 hours
position-simulator-7f4d479d95-54xvv	default	app: position-simulator pod-template-ha	mpouyakh	Running	0	2.00m	169.44Mi	4 hours

حال برای اینکه بتوانیم به گونه ای عمل کنیم که بر اساس این metric ها پاد های ما به صورت خودکار تولید شوند یا نابود شوند از قابلیت Horizontal pod autoscaling (HPA) استفاده میکنیم و بر اساس threshold هایی که خودمان تعریف میکنیم؛ کوبرنتیز شروع به تولید یا نابود کردن پاد ها میکند. برای این کار باید بگوییم که اگر یک پاد از یک حد بیشتر مصرف کرد تعداد replica رابیشتر کند تا کیفیت پاسخ دهی افزایش یابد و تاخیر کمتر باشد. باری اینکه HPA بتواند کار کند باید میزان مصرف پیشنهادی برای هر deployment را مشخص کنیم به منظور اینکه HPA بر اساس ان بتواند کار autoscaling را انجام دهد. که به طور مثال برای چند پاد این کار را انجام داده ام.

```
root@mpouyakh-m:/home/mpouyakh# kubectl get hpa
NAME                                REFERENCE                                          TARGETS         MINPODS   MAXPODS   REPLICAS   AGE
client-deployment                   Deployment/client-deployment                       1%/60%          1         5         1          26h
pouya-nginx-ingress-controller      Deployment/pouya-nginx-ingress-controller          11%/50%         1         5         1          7d16h
pouya-nginx-ingress-default-backend Deployment/pouya-nginx-ingress-default-backend     11%/50%         1         5         1          7d16h
prometheus-server                   Deployment/prometheus-server                       <unknown>/80%   1         5         1          25h
server-deployment                   Deployment/server-deployment                       <unknown>/80%   1         5         1          25h
worker-deployment                   Deployment/worker-deployment                       0%/80%          1         5         1          25h
root@mpouyakh-m:/home/mpouyakh#
```

به طور مثال برای اولین deployment که clinet-deployment است اگر از ۶۰ درصد میزانی که برای این پاد مشخص کردیم بگذرد شروع به اضافه کردن این پاد میکند و تعداد replica را افزایش میدهد تا بار تقسیم شود و پاد crash کند. میزان درخواستی برای این deployment را همانطور که در شکل زیر مشاهده میکنیم 70m گذاشتیم که هر یک cpu مساوی 1000m است. در این مثال اگر پاد مد نظر ما از ۶۰ درصد 70m عبور کند؛ HPA شروع به افزایش پاد ها میکند و اگر از این مقدار پایین تر بیاید پاد ها شروع به کاهش میکنند. پس برای اینکار ما ابتدا باید بررسی کنیم یک پاد در شرایط معمول جقدر cpu و memory نیاز دارد و بر اساس آن مقدار درخواستی برای آن پاد را مشخص کنیم تا کوبرنتیز بتواند در مواقعی که بار بر روی آن پاد زیاد شد؛ تعداد را افزایش دهد.

سوال ها :

۱. ایا این سرویس ها کافی است یا باید سرویس های پیچیده تر و بیشتری اضافه یا جایگزین کنم؟

۲. ایا برای autoscaling معیار های مصرفی cpu و memory کافی است یا معیار های بیشتری باید اضافه شود و بر اساس آن ها autoscaling را انجام دهیم؟