به نام خدا

پروژه نهایی

رایانش ابری

گروه سوم
علیرضا زارع زین آبادی ۹۹۳۱۰۲۲
امیرفاضل کوزه گر کالجی ۹۹۳۱۰۹۹

ترم پاییز ۱۴۰۲

گام اول

پیاده سازی سرویس ها با پایتون، جنگو، سلری و پستگره انجام شده است.

سرویس api:

1. مدیریت درخواست ها:

```python
@api_view(["POST", "GET"])
def submit_server_or_check_server(request):
    if request.method == "GET":
        server_id = request.query_params.get('id')
        try:
            server = Server.objects.get(id=server_id)
            serializer = ServerSerializer(server)
            return Response(serializer.data)
        except Server.DoesNotExist:
            return Response(status=404)
    else:
        try:
            address = request.data['address']
            server = Server(address=address)
            server.save()
            return Response(status=201)
        except:
            return Response(status=400)



@api_view(["GET"])
def check_all_servers(request):
    servers = Server.objects.all()
    serializer = ServerSerializer(servers, many=True)
    return Response(serializer.data)
```

2. مدل مورد استفاده:

```python
class Server(models.Model):
    id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
    address = models.CharField(max_length=255, verbose_name="Server Address")
    success_count = models.PositiveIntegerField(default=0, verbose_name="Success Count")
    failure_count = models.PositiveIntegerField(default=0, verbose_name="Failure Count")
    last_failure = models.DateTimeField(null=True, blank=True, verbose_name="Last Failure Time")
    created_at = models.DateTimeField(auto_now_add=True, verbose_name="Creation Time")

    class Meta:
        verbose_name_plural = "Servers"
        ordering = ['-created_at']

    def __str__(self):
        return f"Server {self.address} - Success: {self.success_count}, Failures: {self.failure_count}"


```

3. مدیریت خواندن و نوشتن برای statefullset:

```python
PSQL_READ = {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': os.environ.get("POSTGRES_NAME", ""),
        'USER': os.environ.get("POSTGRES_USER", ""),
        'PASSWORD': os.environ.get("POSTGRES_PASSWORD", ""),
        'HOST': os.environ.get("POSTGRES_HOST_READ"),
        'PORT': int(os.environ.get("POSTGRES_PORT", "1")),
        }
PSQL_WRITE = {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': os.environ.get("POSTGRES_NAME", ""),
        'USER': os.environ.get("POSTGRES_USER", ""),
        'PASSWORD': os.environ.get("POSTGRES_PASSWORD", ""),
        'HOST': os.environ.get("POSTGRES_HOST_WRITE"),
        'PORT': int(os.environ.get("POSTGRES_PORT", "1")),
        }
SQLITE = {
        "ENGINE": "django.db.backends.sqlite3",
        "NAME": BASE_DIR / "db.sqlite3",
    }

DATABASES = {
    "read": SQLITE if os.environ.get("DEBUG", "1") == "1" else PSQL_READ,
    "default":  SQLITE if os.environ.get("DEBUG", "1") == "1" else PSQL_WRITE,
}


DATABASE_ROUTERS = ['healthServiceApi.models.PrimaryReplicaRouter']
# Password validation
```

```python
class PrimaryReplicaRouter:
    def db_for_read(self, model, **hints):
        return 'read'

    def db_for_write(self, model, **hints):

        return 'default'

    def allow_relation(self, obj1, obj2, **hints):

        return True

    def allow_migrate(self, db, app_label, model_name=None, **hints):

        return True
```

سرویس زمانبند:

```
CELERY_BROKER_URL = os.environ.get("CELERY_BROKER_URL", "redis://localhost:6379");
CELERY_RESULT_BACKEND = os.environ.get("CELERY_RESULT_BACKEND", "redis://localhost:6379");
CELERY_BEAT_SCHEDULE = {
    'check-server-health-every-hour': {
        'task': 'healthServiceApi.tasks.check_server_health',
        'schedule': int(os.environ.get("TESTCYCLE", "10")),
    },

}
```

```
@shared_task
def check_server_health():
    for server in Server.objects.all():
        _address = server.address if "http://" in server.address or "https://" in server.address else f"http://{server.address}"
        try:
            response = requests.get(_address)
            if response.status_code == 200:
                print(f"Server {_address} Is Alive")
                server.success_count += 1
                server.save()
            else:
                print(f"Server {_address} Is Dead [Status = {response.status_code}]")
                server.failure_count += 1
                server.last_failure = now()
                server.save()
        except requests.RequestException as e:
            print(f"Server {_address} Is Dead [Error = {str(e)}]")
            server.failure_count += 1
            server.last_failure = now()
            server.save()

```

در اینجا به یک مسیج بروکر نیاز داشتیم که برای راحتی از ردیس استفاده شد.

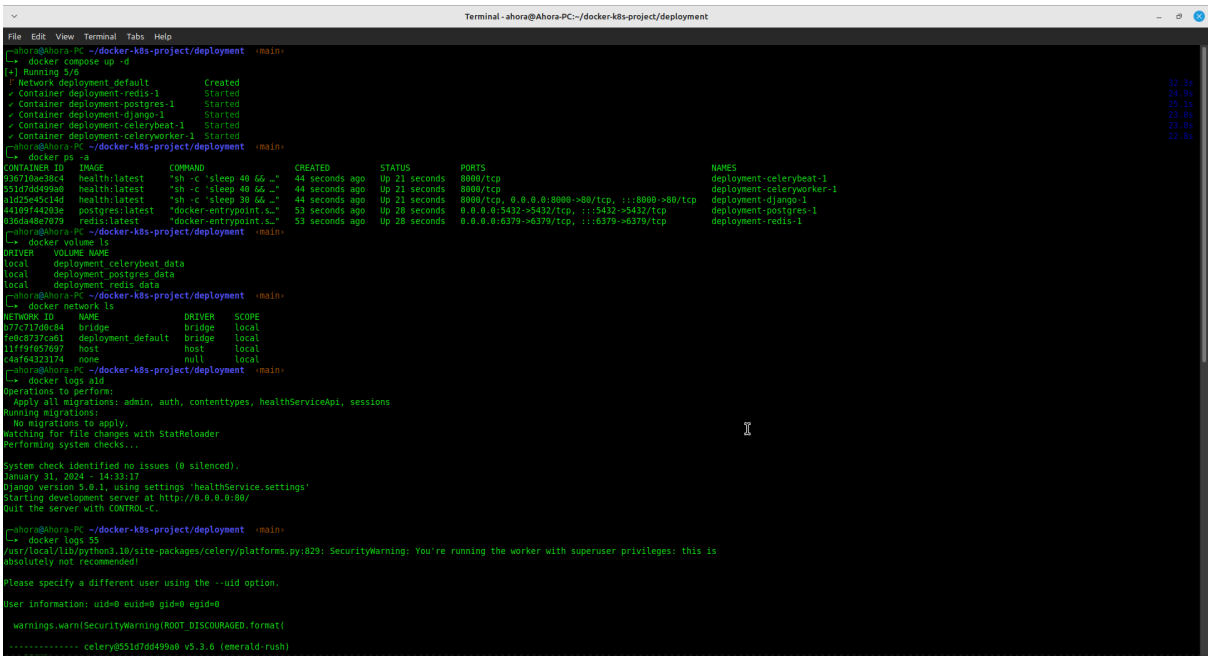
با توجه به اینکه از پایتون برای پیاده سازی استفاده شد میتوان آنرا در یک لایه داکرایز کرد و نیاز به بیلد ندارد. همچنین این ایمیج برای کانتینر های ورکر سلری و بیت سلری نیز استفاده می شود پس کامند را در داکر کامپوز و مانیفست کوبرنیز مینویسیم.

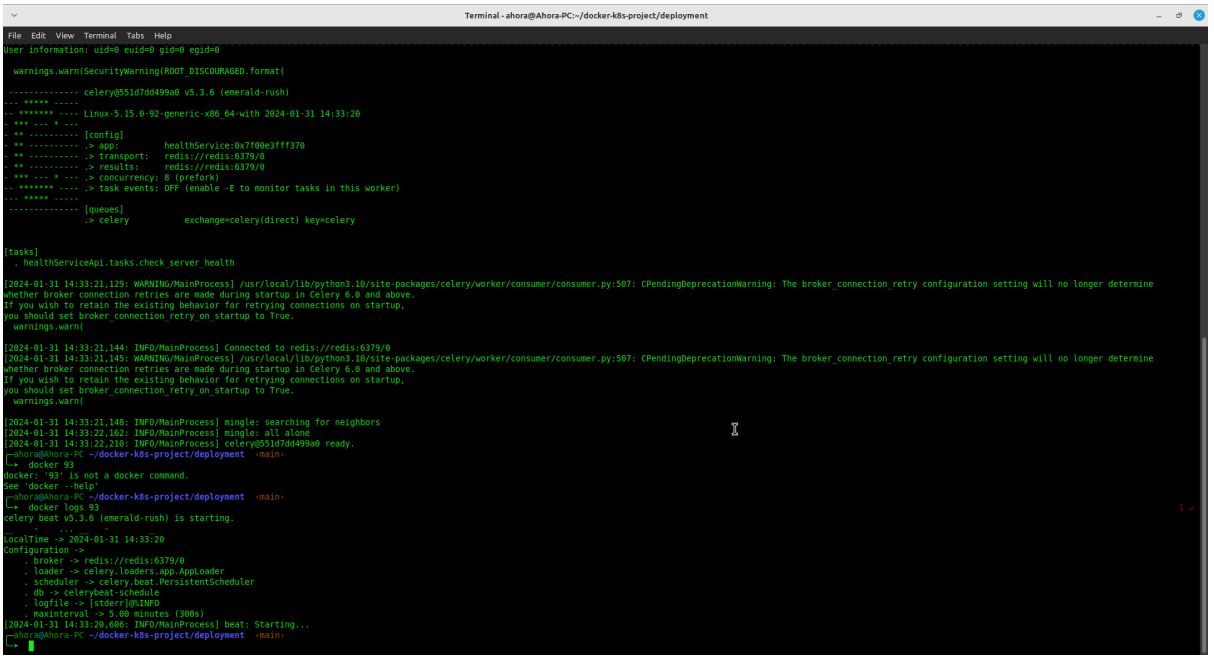
```
FROM python:3.10.2-alpine
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1
WORKDIR /code
COPY ./  .
RUN apk add --no-cache --virtual .build-deps \
    ca-certificates gcc postgresql-dev linux-headers musl-dev \
    libffi-dev jpeg-dev zlib-dev


RUN pip install -r requirements.txt

EXPOSE 80

```

در داکر کامپوز نیز این متغیر های محلی برای کانفیگ پذیر بودن برنامه استفاده شد
پورت برای جنگو چون باید در کامند آن بیاید میتواند کانفیگ پذیر باشد.

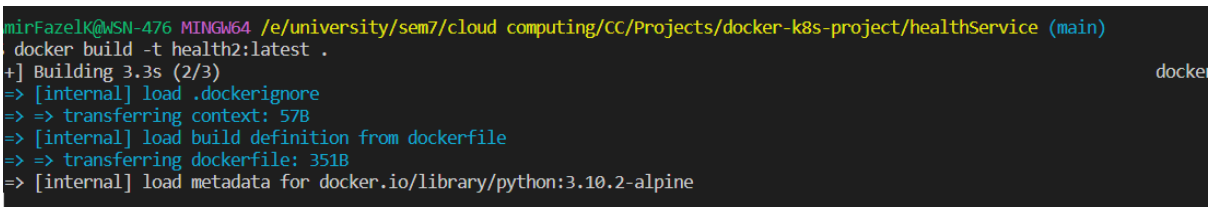
گام دوم

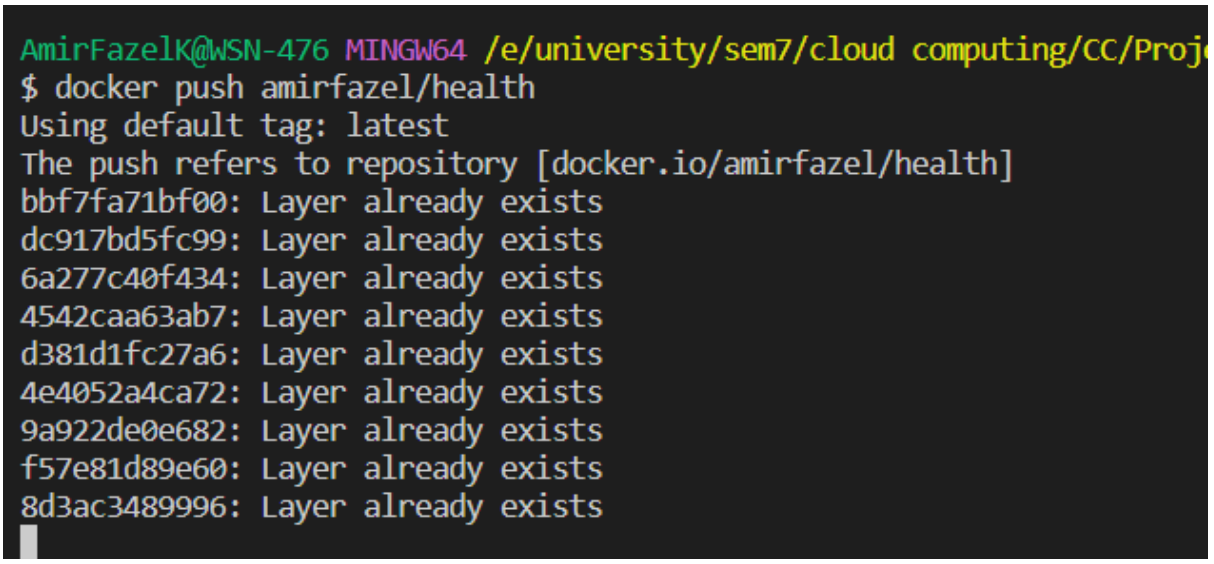
صحت اجرای داکر کامپوز:

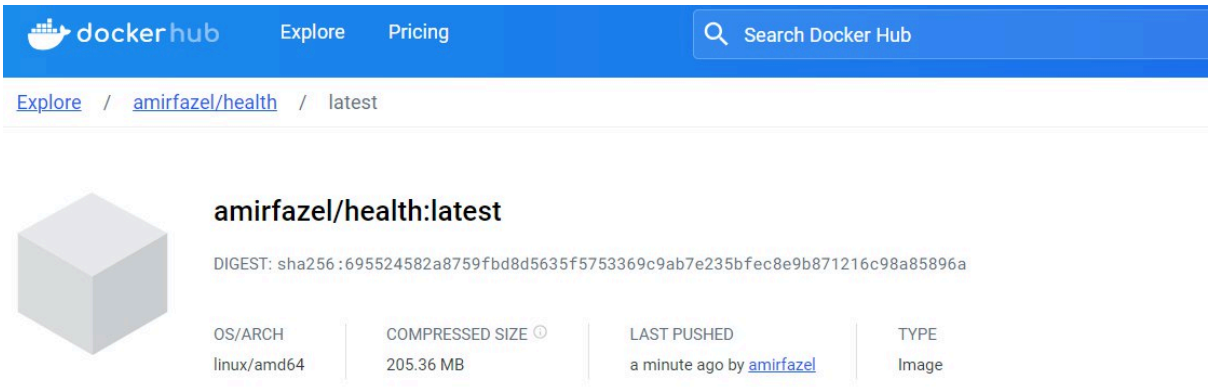




در ادامه با دستور docker build -t از داکر فایل خود یک خروجی ایمیج تهیه می کنیم و با دستور docker push ایمیج تولید شده را وارد داکرهاب می کنیم.



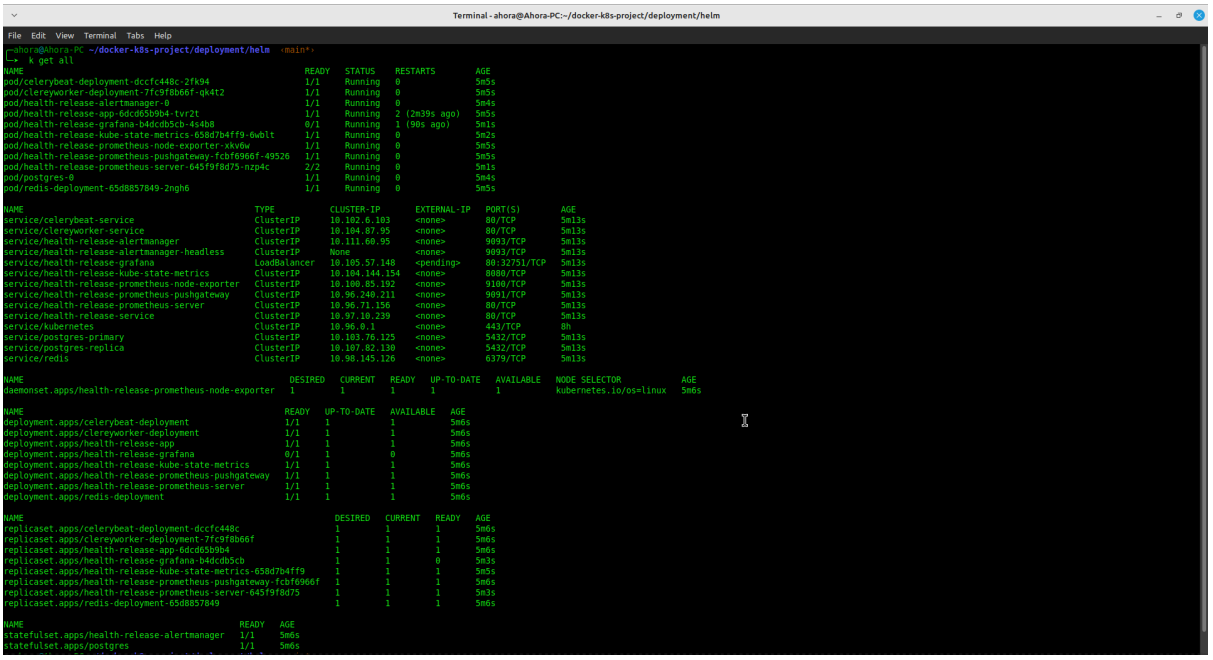




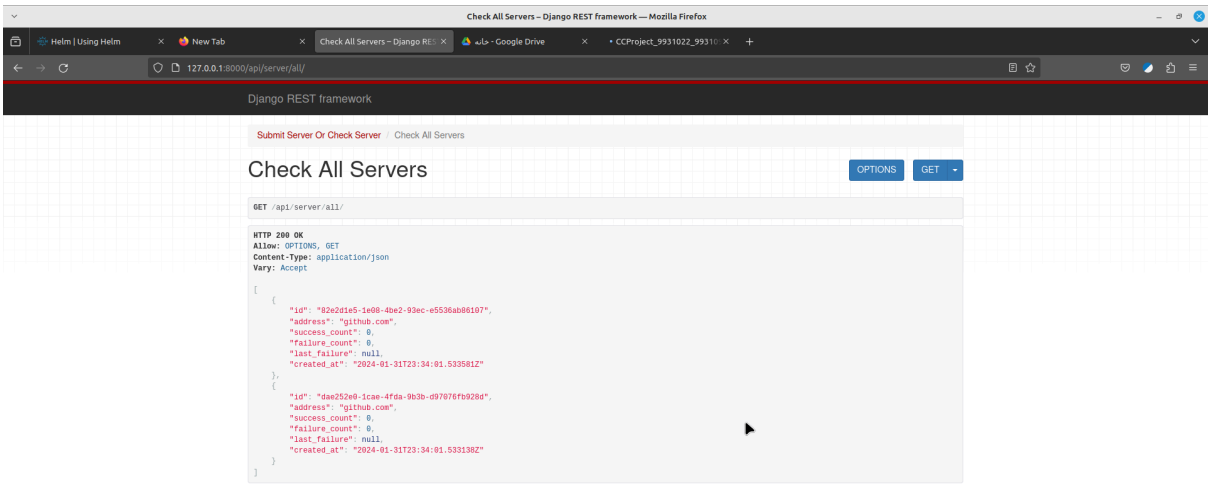
گام سوم

در ادامه با نوشتن فایل های توصیف کوبرنتیز و بالا آوردن پاد ها، موارد زیر را که شامل سرویس ها و پاد ها می باشد را داریم:

k get all:



after port forward:

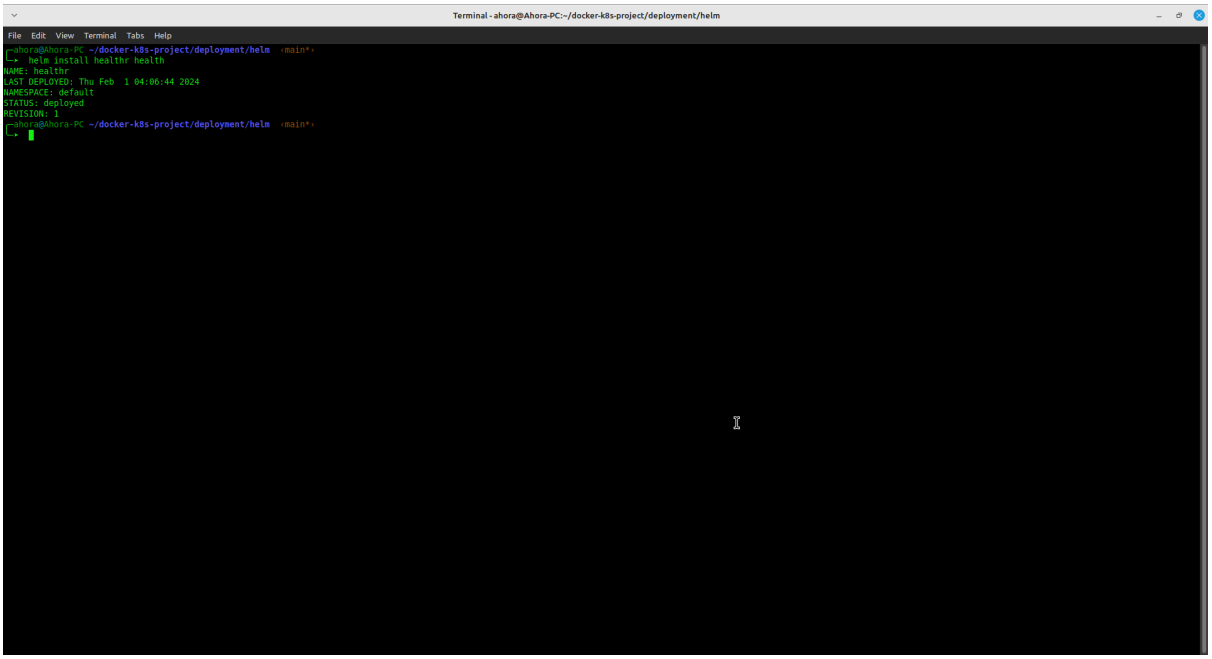


سوال سوم) در بخش stateful ، دو پاد تولید شد. یکی مربوط به خواندن یا slave و دیگری مربوط به نوشتن روی دیتابیس یا primary.

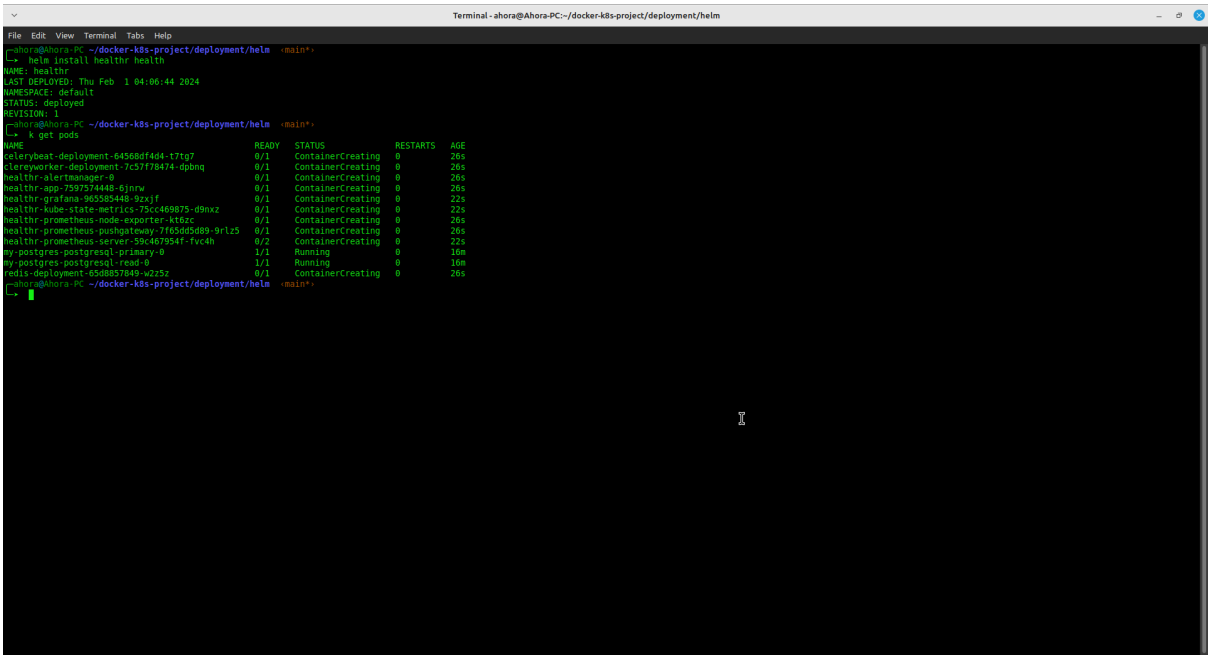
گام چهارم

در اینجا با وارد کردن دستور helm install کانتینر های کوبری خود را بالا می اوریم.

helm install

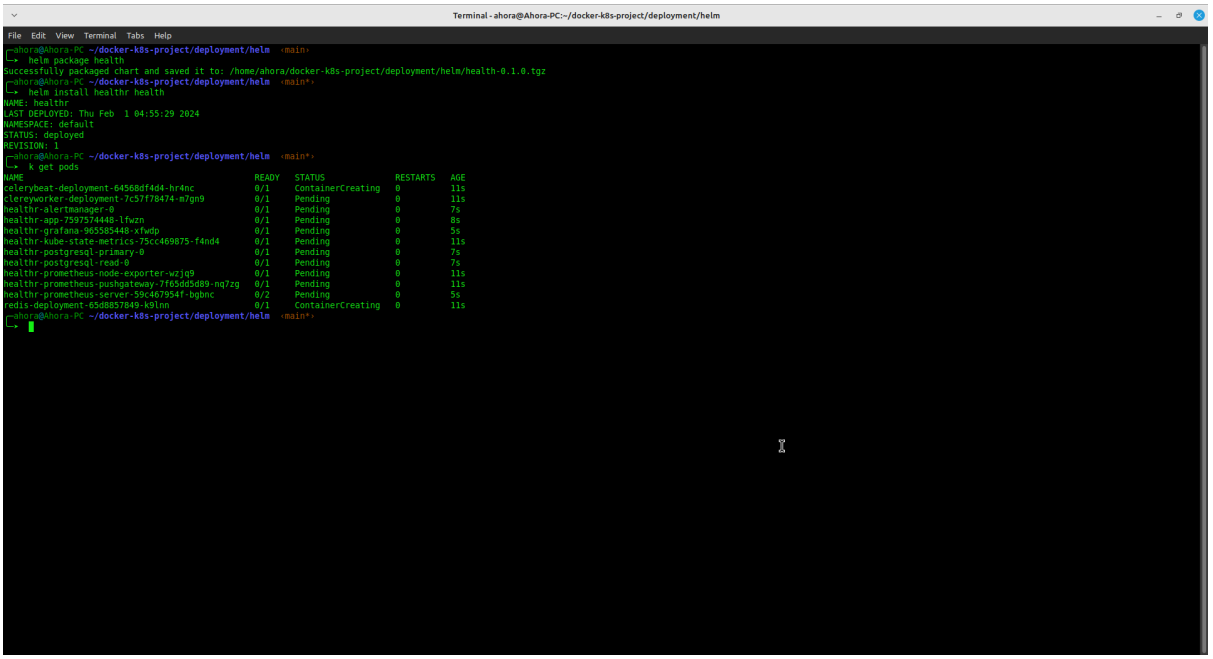


k get pods after helm install:

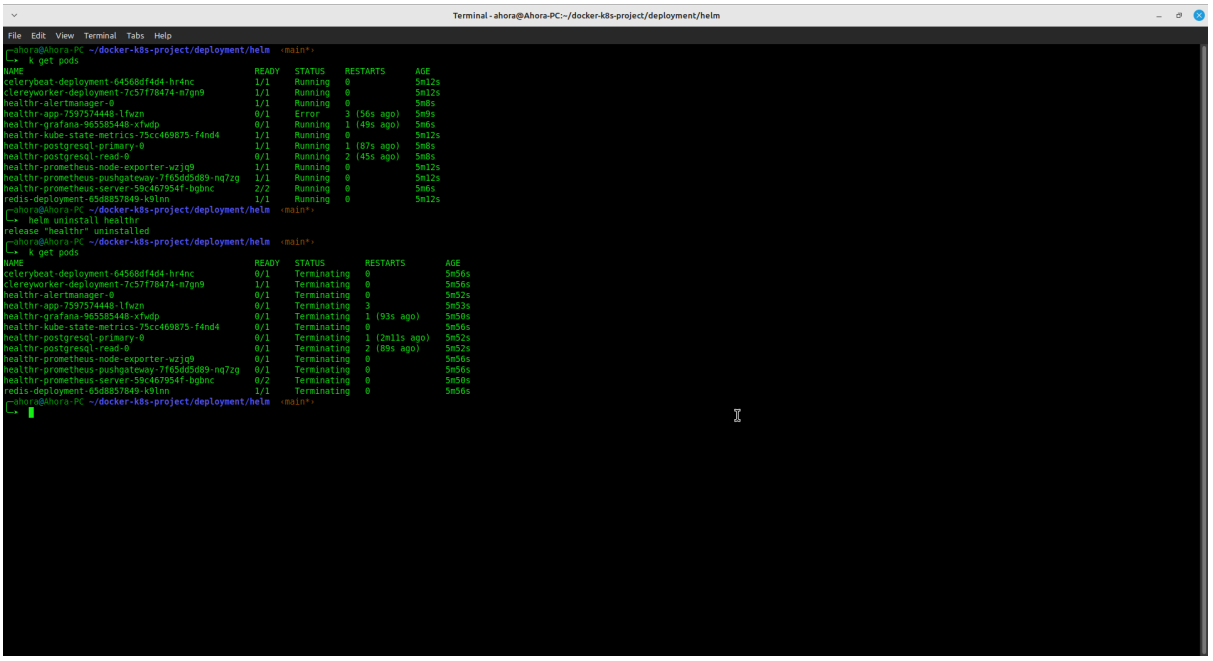


در ادامه با زدن دستور helm uninstall یک اپ را به کلی پاک کرده و پاد ها و موارد مربوطه به آن حذف می شوند.

helm install final:



helm uninstall:



بخش امتیازی

probes
1)
liveness:

چک می کند تا ببیند که یک کانتینر به درستی اجرا می شود یا نه

readiness:

چک میکند تا ببیند ایا یک کانتینر اماده دریافت پهنای باند هست یا نه

startup:

چک می کند تا ببیند یک برنامه درون یک کانتینر به درستی اجرا می شود یا نه

2)

برای دو پروب liveness و startup به دلیل داشتن یک اندپوینت پینگ و دانستن این موضوع که برای ایجاد پایگاه داده عملیات migrate انجام می شود پس می توان گفت تا بالا آمدن کانتینر برنامه هم بالا می اید هر دوی این پروب ها را با اندپوینت ping اطمینان می دهیم.
پروب readiness هم مرتبط با دریافت پهنای باند می شد. در این برنامه هرگاه که قابلیت خواندن از دیتابیس را داشته باشیم می توانیم readiness را نیز اطمینان دهیم.

prometheus:

متریک ها:

- یک شمارنده برای تعداد ریکوئست ها
- یک شمارنده برای ریکوئست های ناموفق
- یک هیستوگرام برای زمان پاسخگویی ریکوئست ها
- یک gauge برای نگهداری از تعداد سرور های دیتا بیس