فریم ورک Django

فریم ورک Django یک چارچوب وب رایگان و متن باز است که در **Python** نوشته شده و از الگوی معماری **MVT** یا همون **model-view-template** پیروی می کند. هدف اصلی **Django** این است که ایجاد وب سایت های پیچیده و مبتنی بر پایگاه داده را آسان تر کند. این چارچوب بر قابلیت استفاده مجدد، **Pluggability** اجزای سازنده، کد کمتر، اتصال کم و توسعه سریع تاکید می کند. در این فریم ورک از **Python** در سراسر برنامه حتی برای فایل های تنظیمات و مدل های داده استفاده می شود.

اجزا

علی رغم نامگذاری خاص خود مانند نامگذاری اشیا قابل خواندن توسط پاسخ های **HTTP** که **View** نام دارد، چارچوب هسته **Django** را می توان به عنوان یک معماری **MVC** نیز یاد کرد. همچنین موارد زیر در چارچوب این هسته گنجانده شده اند:

- **یک وب سرور سبک و مستقل برای توسعه و آزمایش.**
- **یک سیستم قالب که از مفهوم ارث برده شده از برنامه نویسی شی گرا استفاده می کند.**
- **یک چارچوب ذخیره سازی که می تواند از هر یک از چندین روش Cache استفاده کند.**
- **یک سیستم بین المللی سازی شامل ترجمه های اجزای سازنده Django به زبان های مختلف**
- **یک سیستم برای گسترش قابلیت های موتور قالب**

برنامه های کاربردی همراه

توزیع اصلی **Django** همچنین تعدادی از برنامه های موجود در بسته **Contrib** خود را شامل می شود که می توان به موارد زیر اشاره کرد:

- **یک سیستم تایید هویت extensible**
- **رابط کاربری اداری پویا**
- **ابزار برای تولید خوراک RSS و Atom syndication**
- **ابزار برای تولید نقشه های سایت گوگل**
- **یک چارچوب برای ایجاد برنامه های کاربردی GIS**

مقدمات و طبقه بندی سرور

فریم ورک Django را می توان در رابطه با آپاچی یا **Ngixn** با استفاده از **WSGI** یا کوکی با استفاده از فلوت اجرا کرد. **Django** نیز شامل توانایی راه یک سرور**FastCGI** می شود که امکان استفاده از پشت هر وب سروری که از **FastCGI** پشتیبانی می کند را می دهد (مانند **Lighttpd** یا **Hiawatha**). این چارچوب همچنین می تواند در رابطه با **python** در هر سرور درخواست **Java EE** مانند **GlassFish** یا **JBoss** اجرا شود.

ویژگی ها

این چارچوب برای کمک به توسعه دهندگان طراحی شده است تا برنامه ها را هر چه سریع تر و در اسرع وقت به اتمام رساند.

این چارچوب امنیت را به طور جدی ایفا می کند و به توسعه دهندگان کمک می کند تا از بسیاری از خطا های امنیتی مشترک جلوگیری کنند.

برخی از شلوغ ترین سایت ها در وب، توانایی این چارچوب را به سرعت و انعطاف پذیری در مقیاس می گیرند.

داکر داکر (Docker) یک پلتفرم متن باز است که بر مبنای سیستم عامل لینوکس راه‌اندازی شده است. در پاسخ به سوال داکر چیست، خیلی ساده می‌توان گفت، ابزاری است که می‌تواند فرایند ایجاد، پیاده سازی و اجرای برنامه‌ها را با استفاده از Container ها بسیار ساده کند.

Docker، نوعی ماشین مجازی است و این امکان را برای برنامه‌ها فراهم می‌کند تا از یک Kernel واحد لینوکس استفاده کرده و از امکاناتی بهره مند شوند که در سیستم عامل میزبان ارائه نشده است. به این ترتیب می‌توانند به صورت مستقل از پیش نیازها و امکانات مازاد بهره برداری کنند. این موضوع باعث می‌شود سرعت و عملکرد برنامه بهبود قابل ملاحظه‌ای پیدا کند و حجم آن نیز کاهش یابد.

کانتینر (Container)

نگهداری برنامه‌ها در محیطی ایزوله و به صورت مستقل، از اهداف سیستم‌های جدید توسعه نرم افزار است. به این ترتیب فعالیت آن‌ها بر روی یکدیگر تاثیری نداشته و کاملا مستقل از هم کار می‌کنند. یکی از راه‌های پیاده کردن این تکنولوژی استفاده از ماشین مجازی (Virtual Machine) است که برنامه‌ها را روی یک سخت افزار اما جدا از هم نگهداری می‌کند. در این حالت component ها با هم تداخل نداشته و رقابت برای استفاده از منابع سخت افزاری به حداقل می‌رسد.

اما کانتینر چیست؟ در مقابل ماشین‌های مجازی، کانتینرها(Container) قرار دارند، آن‌ها می‌توانند جایگزین مناسبی برای ماشین‌های مجازی باشند. کانتینرها محیط‌های اجرایی را جدا کرده و هسته سیستم عامل را به اشتراک می‌گذارد. کانتینرها نسبت به ماشین‌های مجازی از منابع کمتری استفاده می‌کنند و همچنین خیلی سریع قابلیت اجرا پیدا می‌کنند.

می‌توان کانتینرها را به **3 بخش** تقسیم کرد که عبارتند از:

سازنده (Builder):

فناوری مورد استفاده برای ساخت کانتینر

موتور (Engine):

فناوری مورد استفاده برای راه‌اندازی کانتینر

تنظیم (Orchestration):

فناوری مورد استفاده برای تنظیمات و مدیریت کانتینر

کانتینر داکر

هر چند مفهوم container از مدت‌ها قبل در حوزه IT مطرح بوده، اما معرفی و ارائه داکر به عنوان یک پروژه متن باز باعث شد استفاده از container ها دوباره فراگیر شود.

کانتینر (Container) این امکان را برای توسعه دهندگان فراهم می‌کند تا بسته کاملی از برنامه‌های خود همراه تمامی بخش‌های مورد نیاز آن ایجاد کرده و آن را در قالب یک بسته واحد ارسال کنند.

با وجود کانتینرها، توسعه دهندگان می‌توانند اطمینان داشته باشند که برنامه داکر در هر ماشین، با سیستم عامل لینوکس بدون توجه به تنظیمات سفارشی قابل اجرا و استفاده است. ماشین جدید می‌تواند تنظیماتی متفاوت با ماشینی که برنامه روی آن طراحی شده، داشته باشد.

برای ساخت یک برنامه داکر و همچنین کار با داکر باید از کامپوننت‌های مختلف استفاده کنیم. در ادامه این کامپوننت‌ها را معرفی و بررسی می‌کنیم.

Dockerfile

هر کانتینر داکر به وسیله یک فایل داکر شروع به کار می‌کند. در پاسخ به سوال داکر فایل چیست، به سادگی می‌توان گفت Dockerfile ها در واقع فایل‌های تنظیمات داکر هستند که با استفاده از آن‌ها می‌توانیم به داکر بگوییم که یک container را چگونه بالا بیاورد و تنظیم کند. به عنوان مثال، چه سرویس‌هایی را فعال کند و چطور به آن‌ها اجازه دسترسی دهد. در واقع داکر فایل مشخص می‌کند که پشت Container ما چه سیستم عاملی قرار بگیرد، همینطور از چه زبان ها، متغیرهای محلی، پورت‌های شبکه یا غیره استفاده شود. و مهم‌تر از همه اینکه مشخص کند Container ما بعد از اینکه واقعا اجرا شد قرار است چه کاری انجام دهد.

مکانیزم عملکرد داکر

تا اینجا دانستیم داکر چیست و کانتینرها به چه شکل عمل می‌کنند، در این بخش قصد داریم مکانیزم عملکرد داکر را بررسی کنیم.

داکر (Docker) یک لایه واسطه بین سیستم عامل اصلی و بسته نرم افزاری ایجاد می‌کند. در واقع با استفاده از این لایه، نرم افزارها را از یکدیگر تفکیک می‌کند. در سیستم عامل لینوکس قابلیت‌هایی برای تفکیک و ایزوله کردن منابع وجود دارد که هم هسته سیستم عامل و هم گروه‌ها و منابع سخت افزاری و نرم افزاری سیستم عامل را به صورت ایزوله و تفکیک شده در اختیار نرم افزارها قرار می‌دهد، که سیستم داکر نیز از آنها استفاده می‌کند.

CI/CD گیت لب، ابزاری ست که در گیت لب ساخته شده و از طریق یک سری روش های مداوم نرم افزار ها را توسعه می دهد.

- Continuous Integration یا (CI)
- Continuous Delivery یا (CD)
- Continuous Deployment یا (CD)

CI مخفف عبارت Continuous Integration، به معنی یکپارچه سازی مداوم میباشد. CI ها، با انتقال تکه کدهای کوچک به پایگاه کد برنامه های میزبانی یا هاست شده در یک مخزن Git، کار میکنند و در هر بار انتقال یک Pipeline از اسکریپت ها برای ساخت، تست و معتبرسازی تغییرات کد قبل از merge کردن آنها به branch اصلی، راه اندازی و اجرا میشود.

این متدها به شما اجازه میدهند که باگ و ارور های موجود در چرخه توسعه را شناسایی کنید. سپس مطمئن شوید که تمام کدهای توسعه یافته با کدهای استاندارد ایجاد شده در برنامه، مطابقت داشته باشند.

همچنین گیت لب با استفاده از DevOps خودکار، می تواند به صورت خودکار برنامه ها را شناسایی، ایجاد و تست کند و سپس توسعه دهد.

مفاهیم و اصطلاحات گیت لب CI/CD

گیت لب CI/CD برای توصیف، اجرا و توسعه محصولات کاربران از اصطلاحات و مفاهیم زیر استفاده میکند:

Pipelines : ساختار دهی پروسه های CI/CD از طریق Pipeline ها.

CI/CD variables : استفاده مجدد از متغیرهای مبتنی بر یک جفت کلید variable/value.

Environments : توسعه برنامه های ساخته شده در محیط های مختلفی مثل staging و production.

Job artifacts : خروجی، کاربرد و استفاده مجدد از job artifact ها.

Cache dependencies : کش کردن dependency ها برای داشتن یک اجرای سریع تر.

GitLab Runner : پیکربندی Runner های شخصی برای اجرای اسکریپت ها.

PipeLine efficiency : پیکربندی PipeLine ها برای اجرای سریع و کارآمد.

Test cases : پیکربندی PipeLine ها برای اجرای سریع و کارآمد.

پایگاه داده پستگرس (PostgreSQL)

این بانک اطلاعاتی که در برخی منابع با نام Postgres معرفی شده است یکی از قدرتمندترین بانک های اطلاعاتی متن باز دنیا با تاکید بر انعطاف پذیری و انطباق با استانداردها می باشد. پستگرس‌کیوال توسط گروه توسعه سراسری پستگرس‌کیوال توسعه داده می‌شود، که شامل تعداد زیادی از افراد داوطلب است. پایگاه داده پستگرس یک object-relational database management system می باشد که برای انواع مختلف سیستم های عامل بهینه شده است و می تواند بر روی مدل های مختلفی از سیستم های عامل مانند ویندوز ، لینوکس و macOS نصب شده و سرویس دهی نماید. یکی از نکات قابل توجه دیتابیس پستگرس این است که در سرورهای مک (macOS Server) به صورت دیتابیس پیش فرض سرور در نظر گرفته شده است.

یکی از ویژگی های دیتابیس پستگرس متن باز بودن این بانک اطلاعاتی می باشد که سبب شده تا توابع، نوع داده ها و عملگرهای بسیار زیادی به آن افزوده شده که کار با این بانک اطلاعاتی را بسیار ساده نموده است. همچنین این مهم ، انعطاف و قابلیت های بسیار زیادی را برای برنامه نویسان و توسعه دهندگان بوجود آورده است.

مستندات استقرار:

پیش‌نیازها

تنها پیش‌نیاز این سیستم، اکانت آروان و دسترسی به پلتفرم ابری آروان است.
ابتدا به وب‌سایت آروان به نشانی arvancloud.com بروید و یک اکانت آروان بسازید و یا وارد اکانت‌تان شوید.
سپس به بخش پروفایل رفته و در سربرگ API KEYS برای خود یک API KEY جدید بسازید و آن را در جایی ذخیره کنید.

معماری سیستم

در این مقاله موارد زیر را بررسی می‌کنیم:

- اجزای سازنده‌ی یک اپلیکیشن نوشته‌شده در فریم‌ورک Django
- پیاده‌سازی و ساخت اجزای مورد نیاز در پلتفرم ابری آروان
- ساخت کانتینر از اپلیکیشن برای استقرار برنامه‌ی نوشته‌شده در پلتفرم ابری
- استقرار کامل اپلیکیشن بر روی پلتفرم ابری آروان

یک اپلیکیشن نوشته‌شده در فریم‌ورک Django حداقل شامل کدهای برنامه، یک پایگاه داده و یک وب‌سرور، برای سروکردن فایل‌های استاتیک است. هم‌چنین می‌توان از Redis و Celery نیز برای بهبود عمل‌کرد سیستم استفاده کرد.
این‌جا فقط به شیوه‌ی ساده‌ی اجرای یک اپلیکیشن Django خارج از ساختار سنتی، و در قالب کانتینر پرداخته می‌شود که نمای کلی زیر را دارد:
در این مدل پیاده‌سازی، بدون نیاز به پیکربندی و راه‌اندازی سرورهای مجزا می‌توان به‌سادگی با افزایش تعداد Podها از HA بودن سرویس مطمئن شد. هم‌چنین بارگذاری فایل‌های ایستا بر روی CDN می‌تواند باعث افزایش سرعت و بهبود عمل‌کرد اپلیکیشن شود. این‌جا به نحوه‌ی اجرای یک اپلیکیشن Django بر روی پلتفرم ابری آروان می‌پردازیم.

برای اجرای این سناریو به دو کانتینر نیاز داریم. یک کانتینر پایگاه داده همراه با دیسک، برای ذخیره‌سازی اطلاعات برنامه، و یک کانتینر که شامل کدهای برنامه و کتاب‌خانه‌های مورد نیاز برای اجرای آن است.

برای ساخت و اجرای درست دیتابیس به چهار مانیفست نیاز داریم:

1. Deployment
2. Service
3. Route
4. VolumeClaim

Deployment در واقع خود مدیریت Podهای پایگاه داده، Service پلی برای برقراری ارتباط بین پایگاه داده و اپلیکیشن Django و Route برای دسترسی به پایگاه داده خارج از پلتفرم ابری آروان و VolumeClaim دیسکی است که به پایگاه داده برای ذخیره‌سازی اطلاعات به آن متصل شده است.

هر اپلیکیشن برای اجرا روی پلتفرم ابری آروان باید در قالب کانتینر درآید.
برای ساخت کانتینر اپلیکیشن خود، لازم است ابتدا فایل settings.py را تغییر دهید. سپس در مرحله‌ی بعد با نوشتن یک Dockerfile برای پروژه، آن را به‌صورت داکرایزشده بر dockerhub یا یک repository شخصی فرستاده و پوش کنید. - این فایل‌ها داخل پوشه deployment قرارگرفته اند.

Dockerfile

```
FROM python:3.8

ENV PYTHONUNBUFFERED 1

RUN mkdir /code
WORKDIR /code

ADD requirements.txt /code/
RUN pip install -r requirements.txt

ADD . /code/
```

در این فایل ما از نسخه 3.8 پایتون استفاده میکنیم و در نهایت دپندسی‌های مورد نظر خود در فایل requirements.txt را در داخل ایمیج خود نصب میکنیم .

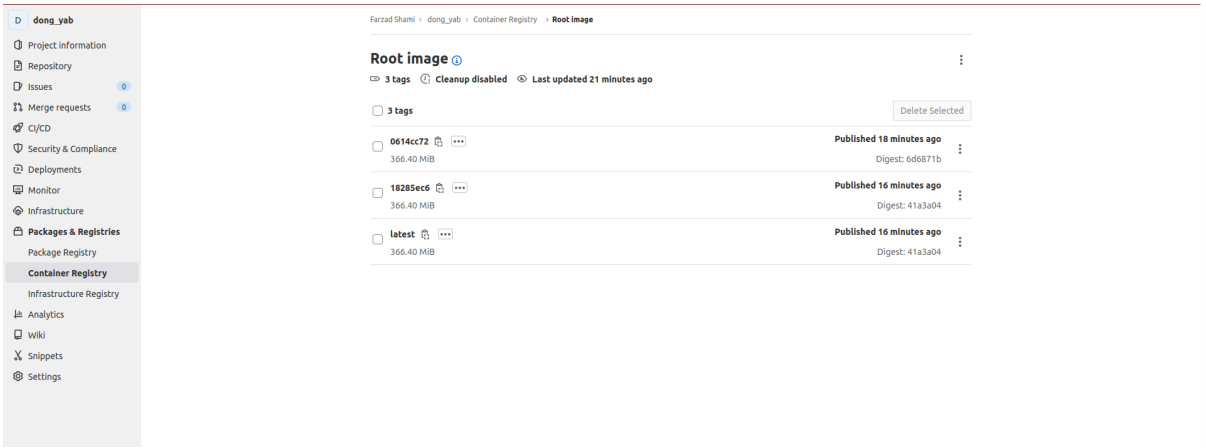
.gitlab-ci.yml

```
stages:
 - build

variables:
 REGISTRY: "registry.gitlab.com"
 LATEST_TAG: "$REGISTRY/farzad-845/dong_yab:latest"
 TAG: "$REGISTRY/farzad-845/dong_yab:$CI_COMMIT_SHORT_SHA"

build:production:
 stage: build
 image:
   name: gcr.io/kaniko-project/executor:debug
   entrypoint: [ "" ]
 script:
   - echo
"{\"auths\":{\"$CI_REGISTRY\":{\"username\":\"$CI_REGISTRY_US
ER\",\"password\":\"$CI_REGISTRY_PASSWORD\"}}}" >
/kaniko/.docker/config.json
   - /kaniko/executor --context $CI_PROJECT_DIR --dockerfile
$CI_PROJECT_DIR/Dockerfile --destination $TAG --destination
$LATEST_TAG --cache=true
 rules:
   - if: '$CI_COMMIT_BRANCH == "master"'
     when: always
   - when: manual
```

با استفاده از امکاناتی که گیتلب به صورت رایگان در اختیار ما قرار میدهد پایپلاین‌های خودمان را مینویسیم، این پایپلاین در صورت اینکه روی برنچ مستر پروژه کدی پوش بشود، آن را بیلد کرده و در داخل کانتینر ریجستری خود قرار میدهد.



نکته مهم این است که در صورت تغییر ریپوزیتوری به ریپوی شخصی و قطع دسترسی‌ها، همواره باید متغییرهای این فایل به روز شوند. در اینجا ما از kaniko برای بیلد کردن استفاده کرده‌ایم که مستندات استفاده از این ابزار را میتوان از لینک زیر مشاهده نمود.

https://docs.gitlab.com/ee/ci/docker/using_kaniko.html

برای مستقر شدن در روی سرور، از k8s یا کوبرنتیس استفاده شده است، این ابزار امکان اسکیل کردن پروژه را به ما میدهد و ما همواره میتوانیم بر حسب نیاز خود با در نظر گرفتن بودجه و تعداد کاربران اقدام به خرید زیرساخت نماییم.
تمامی فایل‌های مرتبط با استقرار در داخل پوشه deployment قرار گرفته اند. نحوی راه‌اندازی پروژه به این شکل است :

Deployment

credentials secret:

fill the `deployment/django/.dockerconfigjson` file with your own private registry credentials

The general format of this file is :

```
{
   "auths": {
       "https://registry.gitlab.com":{
           "username":"REGISTRY_USERNAME",
           "password":"REGISTRY_PASSWORD",
           "email":"REGISTRY_EMAIL",
           "auth":"BASE_64_BASIC_AUTH_CREDENTIALS (see
below)"
       }
   }
```

```
}
```

The base 64 basic credentials mentioned above are the username and password in basic credentials format `{username}:{password}`, encoded with base64 format.

To achieve this simply run :

```
echo -n "{REGISTRY_USERNAME}:{REGISTRY_PASSWORD}" | base64
```

Create a file with the above-mentioned JSON format, and then base64 encode it for the Kubernetes secret.

```
cat .dockerconfigjson | base64
```

Create a file called `registry-credentials.yml` and add the following content

```
apiVersion: v1
kind: Secret
metadata:
 name: registry-credentials
 namespace: default
type: kubernetes.io/dockerconfigjson
data:
 .dockerconfigjson: BASE_64_ENCODED_DOCKER_FILE
```

Now we can create the secret in our cluster.

```
kubectl apply -f registry-credentials.yml
```

Specifying on a deployment

```
imagePullSecrets:
   - name: registry-credentials
```

services:

for PostgreSQL database

```
kubectl create -f deployment/db/
```

for Django instance

```
kubectl create -f deployment/django/service.yaml
kubectl create -f deployment/django/route.yaml
kubectl create -f deployment/django/migration.yaml
```

if migration is successfully done, you cab delete job, after all ...

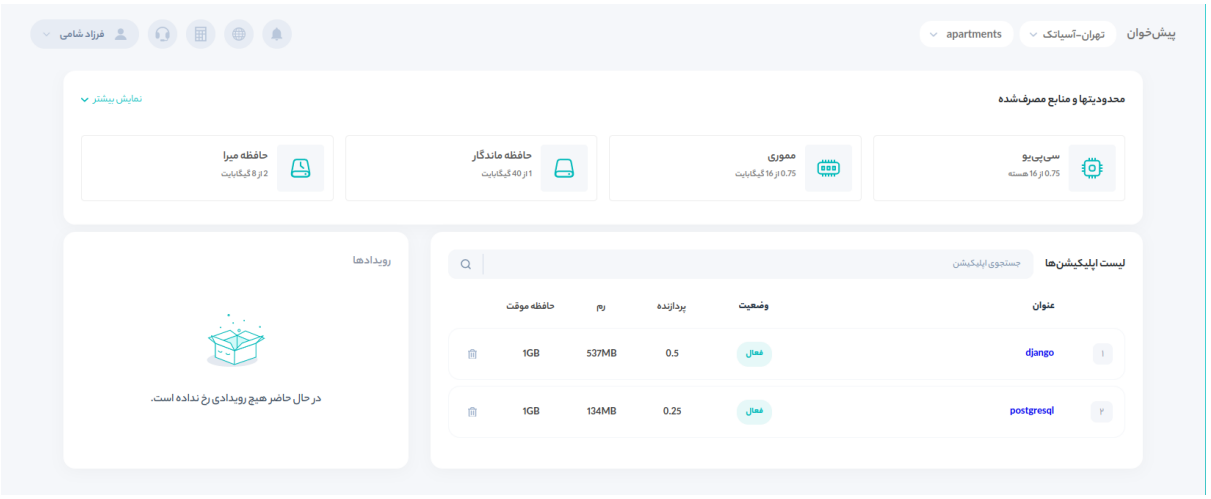
```
kubectl create -f deployment/django/deployment.yaml
```

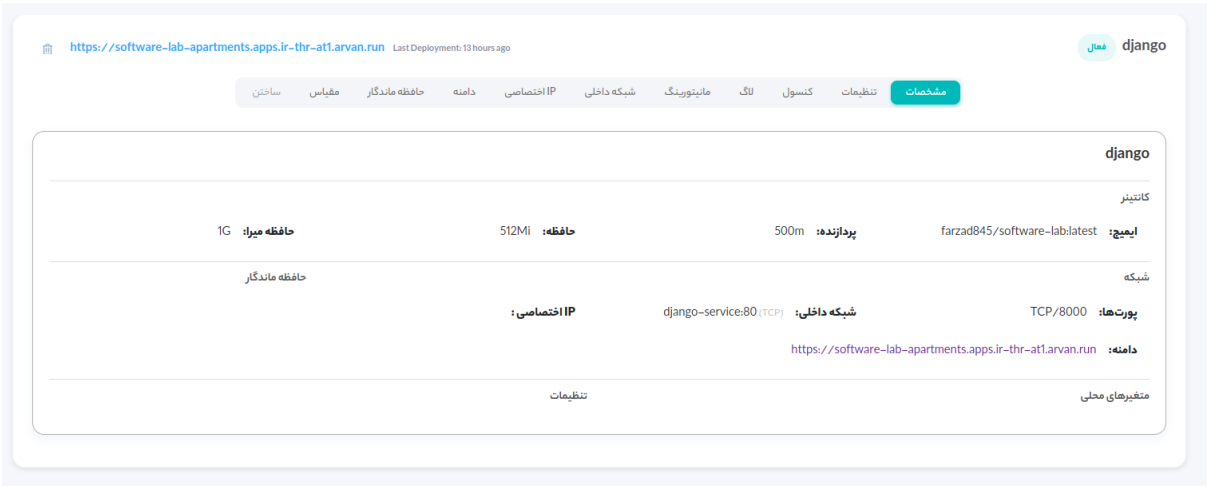
for auto-scale

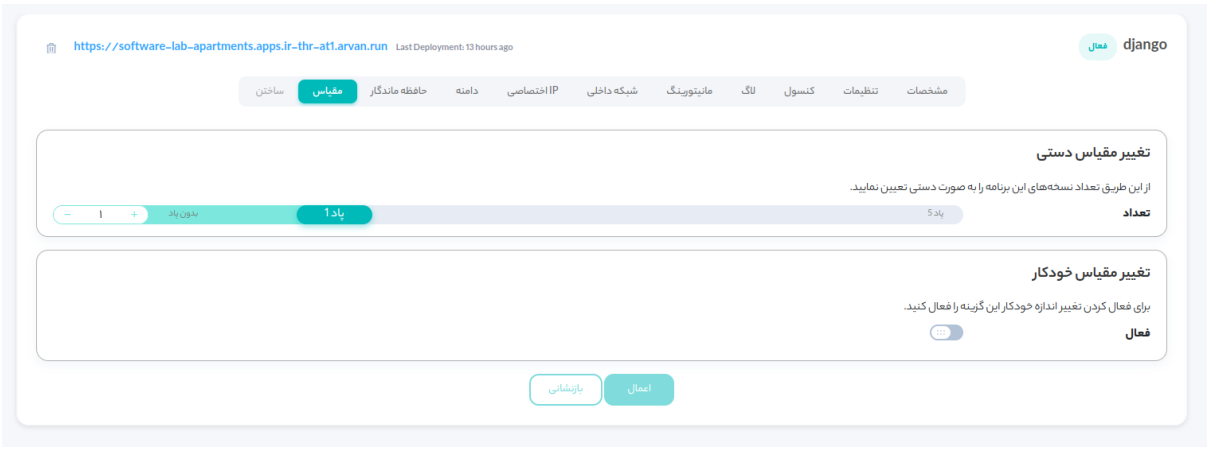
```
kubectl autoscale deploy django --max 10 --min=1
--cpu-percent=50
```

ما برای استقرار از زیرساخت ابری آروان استفاده کردیم که به ما این امکان را می‌دهد که به زیرساخت کوبرنتیس به صورت ابری دسترسی داشته باشیم.

در نهایت پس از استقرار کدها روی آروان کارفرما به راحتی می‌تواند حتی با استفاده رابط گرافیکی هم فرایندهایی نظیر اسکیل اپ را انجام دهد.




django
فعال
https://software-lab-apartments.apps.ir-thr-at1.arvan.run
Last Deployment: 13 hours ago
مشخصات
تنظیمات
کنسول
لاگ
مانیتورینگ
شبکه داخلی
IP اختصاصی
دامنه
حافظه ماندگار
مقیاس
ساختن
django
کانتینر
ایمیج:
farzad845/software-lab:latest
پردازنده:
500m
حافظه:
512Mi
حافظه میرا:
1G
شبکه
پورت‌ها:
TCP/8000
شبکه داخلی:
django-service:80 (TCP)
IP اختصاصی:
دامنه:
https://software-lab-apartments.apps.ir-thr-at1.arvan.run
متغیرهای محلی
تنظیمات
حافظه ماندگار


django
فعال
https://software-lab-apartments.apps.ir-thr-at1.arvan.run
Last Deployment: 13 hours ago
مشخصات
تنظیمات
کنسول
لاگ
مانیتورینگ
شبکه داخلی
IP اختصاصی
دامنه
حافظه ماندگار
مقیاس
ساختن
تغییر مقیاس دستی
از این طریق تعداد نسخه‌های این برنامه را به صورت دستی تعیین نمایید.
تعداد
پاد 1
بدون پاد
پاد 5
تغییر مقیاس خودکار
برای فعال کردن تغییر اندازه خودکار این گزینه را فعال کنید.
فعال
اعمال
بازنشانی