مقدمه

هدف این پروژه، استقرار یک برنامه وب جنگو (Django) بر روی کلاستر کوبرنیتز (Kubernetes) و پیاده‌سازی CI/CD (Continuous Integration/Continuous Deployment) با استفاده از گیت‌لب (GitLab) بود. این گزارش به مراحل اصلی، چالش‌ها و راه‌حل‌ها می‌پردازد.

۱ مراحل اجرای پروژه

۱.۱ نصب Docker و Kubernetes

اولین گام، نصب Docker Desktop بود که شامل کلاستر Kubernetes داخلی است. این ابزار بستر لازم برای کانتینرسازی و تست محلی را فراهم می‌کند.

برای بررسی نصب Docker، دستورات زیر اجرا شدند:

```
docker --version
docker run hello-world
```

نصب kubectl از طریق Homebrew نیز برای تعامل با کلاستر انجام شد. برای بررسی نصب kubectl و وضعیت کلاستر:

```
brew install kubectl
kubectl cluster-info
kubectl get nodes
```

چالش‌ها: اطمینان از سازگاری معماری (ARM64 برای macOS) و مدیریت مصرف منابع Docker Desktop از چالش‌های اولیه بود.

۲.۱ راه‌اندازی پایگاه‌داده Postgres

ابتدا به صورت عادی و بدون docker و kubernetes برنامه پایگاه داده را اجرا کردیم و از طریق پورت آن تعامل با برنامه وب را بررسی کردیم. (برنامه وب را نیز به صورت عادی اجرا کردیم.) سپس image پایگاه داده را دریافت کردیم و حال با docker آن را اجرا کردیم.

```
docker pull postgres:14
docker run postgres:14
```

سپس اتصال برنامه جنگو به دیتابیس در این حالت نیز انجام شد. در آخر برنامه جنگو را نیز داکرایز کرده و به همراه دیتابیس در بستر kubernetes اجرا شد. سپس، منابع کوبرنیتز شامل PersistentVolumeClaim (PVC)، Secret و Deployment/Service برای Postgres تعریف و اعمال شدند.

چالش: اتصال برنامه جنگو به دیتابیس با استفاده از نام سرویس کوبرنیتز (postgres-service) و تزریق امن اعتبارنامه‌ها از طریق Kubernetes Secret، به جای استفاده از hardcode کردن آن‌ها در Deployment بود.

۳.۱ توسعه اپلیکیشن Django

یک برنامه جنگو با قابلیت‌های زیر پیاده‌سازی کردیم :

- مشاهده لیست پادها (Pods) و نودها (Nodes) در کوبرنیتز
- ساخت منابع ساده (ConfigMap) برای تست دسترسی Kubernetes API
- شبیه‌سازی متریک‌های بار (CPU) برای تست Horizontal Pod Autoscaler (HPA)
- ثبت و لیست پیام‌ها در پایگاه‌داده PostgreSQL

چالش‌ها:

- **دسترسی به Kubernetes API و RBAC:** خطای 403 Forbidden به دلیل عدم مجوز کافی پاد جنگو برای تعامل با Kubernetes API. که این مسئله را با استفاده از rbac حل کردیم.
- **پیدا کردن api برای دریافت متریک‌ها چالش زیادی داشت:** حل کردن خطای مجوز برای این api با اضافه کردن یک option در دستور آن حل شد.

۴.۱ Dockerize کردن اپلیکیشن

برنامه جنگو با استفاده از یک Dockerfile چندمرحله‌ای (multi-stage Dockerfile) به ایمیج داکر تبدیل شد.

چالش: پیدا نشدن Gunicorn در زمان اجرا و مدیریت فایل‌های استاتیک جنگو با دستور python manage.py collectstatic –noinput درون Dockerfile.

۵.۱ ساخت مخزن GitLab و راه‌اندازی GitLab Runner

یک مخزن خصوصی در GitLab برای کد پروژه ایجاد شد. سپس GitLab Runner به صورت محلی روی macOS نصب و پیکربندی شد.

نمونه دستور ثبت Runner:

```
sudo gitlab-runner register \
  --url "https://hamgit.ir/" \
  --registration-token "YOUR_TOKEN" \
  --description "My MacOS Runner" \
  --tag-list "macos,docker" \
  --executor "docker" \
  --run-untagged="true"
```

چالش‌ها:

- مشکل zsh: killed با دانلود نسخه صحیح ARM64
- عدم اتصال Runner به GitLab

۲ تست بار با k6

برای بررسی عملکرد برنامه تحت بار و تأیید کارایی Horizontal Pod Autoscaler (HPA)، از ابزار k6 برای شبیه‌سازی ترافیک استفاده شد.

۱.۲ پیکربندی تست بار

اسکریپت تست بار با مشخصات زیر پیاده‌سازی شد:

```
// k6_load_test.js
import http from 'k6/http';
import { sleep } from 'k6';

export const options = {
  stages: [
    { duration: '1m', target: 50 },  // Ramp up to 50 virtual users
    { duration: '3m', target: 100 }, // Maintain 100 virtual users
    { duration: '1m', target: 0 },   // Ramp down gradually
  ],
  thresholds: {
    http_req_duration: ['p(95)<500'], // 95% requests < 500ms
    http_req_failed: ['rate<0.01'],  // <1% failed requests
  },
};

export default function () {
  http.get('http://django.local/');
  http.get('http://django.local/messages/list/');
  sleep(1);
}
```

۲.۲ اجرای تست و نتایج

تست با دستور زیر اجرا شد:

```
k6 run k6_load_test.js
```

نتایج کلیدی مشاهده شده: با زیاد شدن بار روی برنامه hpa به صورت خودکرا برنامه را scale می‌کند و با کاهش بار پاد های برنامه از بین می‌روند. تعداد پادها بین ۱ تا ۵ تنظیم شده‌است.

۳.۲ مانیتورینگ

در طول تست، وضعیت منابع با دستورات زیر مانیتور شد:

```
kubectl get pods -w
```

چالش‌ها و راهکارها:

- **تنظیم آستانه‌های HPA:** مقدار اولیه CPU target بسیار پایین بود که پس از تست‌های مقدماتی به ۸۰% افزایش یافت
- **مشکل DNS Resolution:** با افزودن hostAliases به Deployment حل شد
- **تأخیر در مقیاس‌دهی:** با کاهش scaleDownDelay در HPA بهبود یافت

۴.۲ ایجاد فایل .gitlab-ci.yml

این فایل قلب خط لوله CI/CD است و مراحل ساخت ایمیج داکر، پوش به Docker Hub و استقرار در کوبرنیتز را تعریف می‌کند.

۵.۲ استقرار بر روی Kubernetes

تمامی منابع کوبرنیتز لازم برای برنامه جنگو و PostgreSQL با استفاده از فایل‌های YAML تعریف و اعمال شدند.

۶.۲ بررسی صحت عملکرد

پس از استقرار، وضعیت Pipeline در GitLab UI و منابع کوبرنیتز بررسی شد:

```
kubectl get pods -n hamravesh-project
```

۳ چالش‌ها و راهکارها

- اجرای GitLab Runner روی macOS
- اتصال Django به Postgres در Kubernetes: نیاز به وارد شدن به پاد و اجرای دستور migrate
- مشکلات RBAC برای دسترسی به Kubernetes API
- مشکل اسنفاده از django.local به عنوان domain که با اضافه کردن آن به فایل /etc/hosts حل شد.

۴ جمع‌بندی

در این پروژه، یک اپلیکیشن جنگو توسعه داده شده و با استفاده از CI/CD، به صورت کامل روی یک کلاستر کوبرنیتز مستقر گردید. چالش‌های فنی متعددی در طول مسیر برطرف شد که درک عمیقی از DevOps و kubernetes و docker فراهم شد.

مهرشاد دهقانی
تابستان ۱۴۰۴