Scrapy 학습 전략

**1. Scrapy 학습 전략**

1. **첫날 목표**:
   * Scrapy 설치 및 간단한 Spider 작성.
   * 데이터를 JSON, CSV로 저장하는 기본 파이프라인 구축.
2. **추가 학습 목표**:
   * 네트워크 최적화(속도 제한, User-Agent 설정).
   * 대규모 크롤링 워크플로우 구축.
   * Splash 또는 Selenium을 사용한 동적 데이터 처리.
3. **추천 자료**:
   * Scrapy 공식 튜토리얼: 단계별 학습에 유용.  
     <https://docs.scrapy.org/en/latest/intro/tutorial.html>
   * Scrapy GitHub 프로젝트: 실제 사용 예제.  
     <https://github.com/scrapy/scrapy>

**2. 학습 난이도와 시간 분포**

**1일(6~8시간):**

* Scrapy 설치 및 기본 프로젝트 작성.
* 간단한 Spider로 데이터 크롤링 및 저장.

**2~3일:**

* 프로젝트 구조 이해.
* CSS/XPath 선택자 심화 학습.
* 데이터 저장 파이프라인 구현.

**3~5일 이상:**

* Splash 또는 Selenium 통합.
* 네트워크 요청 최적화(속도 조절, 프록시 사용 등).
* 동적 데이터 처리 및 고급 설정.

**3. 하루 내 학습 가능 범위**

하루(6~8시간) 정도의 학습 시간으로 **기본적인 웹 크롤러를 작성하고 데이터를 추출하는 과정**을 익힐 수 있습니다. 이 과정에서는 다음과 같은 작업이 포함됩니다:

1. **Scrapy 설치 및 프로젝트 생성**:
   * pip install scrapy로 Scrapy 설치.
   * scrapy startproject myproject로 프로젝트 생성.
2. **기본 Spider 작성**:
   * URL 리스트를 설정하고, HTML 페이지에서 데이터(예: 제목, 링크 등)를 추출하는 Spider 작성.
3. **데이터 추출**:
   * CSS 선택자 또는 XPath를 사용하여 필요한 데이터 추출.
   * 데이터를 JSON 또는 CSV 파일로 저장.
4. **명령어 실행**:
   * scrapy crawl spider\_name으로 크롤러 실행.
   * -o output.json을 사용해 데이터를 파일로 저장.

**예제 코드**

scrapy startproject example\_project

# example\_project/spiders/example\_spider.py

import scrapy

class ExampleSpider(scrapy.Spider):

name = "example"

start\_urls = ["https://example.com"]

def parse(self, response):

for title in response.css('h1::text'):

yield {"title": title.get()}

scrapy crawl example -o output.json

**4. 하루 이상 소요되는 학습 범위**

Scrapy는 기본적인 웹 크롤링 이상의 기능을 제공합니다. 아래와 같은 고급 기능을 배우는 데는 추가 시간이 필요할 수 있습니다.

**2.1 Scrapy 프로젝트 구조 이해**

* Scrapy는 프로젝트 구조가 고정되어 있으며, 각 파일(items.py, middlewares.py, pipelines.py)의 역할을 이해해야 합니다.
  + 예: pipelines.py는 데이터를 후처리하거나 저장하는 데 사용.

**2.2 데이터 저장과 처리**

* Scrapy의 파이프라인을 사용해 데이터를 데이터베이스(MySQL, MongoDB 등)에 저장하거나 데이터를 가공.
  + **학습 시간**: 2~4시간 추가 소요.

**2.3 동적 콘텐츠 처리**

* Scrapy는 기본적으로 정적 HTML 데이터를 처리하지만, JavaScript로 생성된 콘텐츠는 추가 설정이 필요.
  + Splash와 같은 JavaScript 렌더링 도구와의 통합.
  + **학습 시간**: 4~6시간 추가 소요.

**2.4 고급 설정**

* 네트워크 요청 관련 설정:
  + 요청 속도 조절(DOWNLOAD\_DELAY).
  + User-Agent 변경.
  + IP 차단 방지(프록시 사용).
  + **학습 시간**: 2~3시간 추가 소요.

**2.5 비동기 처리와 확장**

* Scrapy는 비동기적으로 작동하며, 이를 이해하면 성능을 극대화할 수 있음.
  + **학습 시간**: 3~5시간 추가 소요.

HTTP 상태 코드는 **클라이언트가 서버에 보낸 요청**에 대한 서버의 응답 상태를 나타내는 표준화된 숫자입니다. **각 숫자**는 특정한 응답 상태를 의미합니다. 아래에 주요 HTTP 상태 코드를 정리했습니다:

**1. 상태 코드 구조**

HTTP 상태 코드는 **3자리 숫자**로 구성되며, 첫 번째 자릿수에 따라 응답의 카테고리를 구분합니다:

* **1xx**: 정보 응답 (서버가 요청을 처리 중)
* **2xx**: 성공 (요청이 성공적으로 처리됨)
* **3xx**: 리디렉션 (다른 URL로 이동 필요)
* **4xx**: 클라이언트 오류 (요청에 문제가 있음)
* **5xx**: 서버 오류 (서버에서 문제 발생)

**2. 자주 사용하는 상태 코드**

**2.1 2xx (성공)**

* **200 OK**:
  + 요청이 성공적으로 처리됨.
  + 데이터 크롤링 시, 서버에서 요청된 데이터를 정상적으로 반환했다는 의미.
* **201 Created**:
  + 요청이 성공적으로 처리되었으며, 서버가 새 리소스를 생성했음을 의미.
  + 일반적으로 POST 요청으로 데이터 생성 후 반환.

**2.2 3xx (리디렉션)**

* **301 Moved Permanently**:
  + 요청된 리소스가 영구적으로 다른 URL로 이동.
  + Scrapy는 자동으로 새 URL을 따라가지만, 크롤링 정책에 따라 이를 처리해야 할 수도 있음.
* **302 Found**:
  + 요청된 리소스가 임시적으로 다른 URL로 이동.

**2.3 4xx (클라이언트 오류)**

* **400 Bad Request**:
  + 잘못된 요청(예: 파라미터 오류)으로 인해 서버가 요청을 처리할 수 없음.
  + 크롤링 코드나 요청 URL에 문제가 있을 가능성이 있음.
* **401 Unauthorized**:
  + 인증이 필요한 요청에 인증 정보가 없거나 잘못된 경우.
  + API를 사용하는 크롤러는 적절한 인증 토큰을 제공해야 함.
* **403 Forbidden**:
  + 서버가 요청을 거부.
  + 크롤링이 금지된 리소스에 접근하거나, IP 차단 또는 User-Agent 차단이 원인일 수 있음.
* **404 Not Found**:
  + 요청한 리소스를 찾을 수 없음.
  + 잘못된 URL이거나, 삭제된 리소스일 가능성이 있음.
* **429 Too Many Requests**:
  + 클라이언트가 짧은 시간 내에 너무 많은 요청을 보냄.
  + 서버가 크롤링 속도가 너무 빠르다고 판단하여 요청을 일시적으로 차단.
  + **해결 방법**:
    - 요청 속도를 줄임(DOWNLOAD\_DELAY 증가).
    - 동시 요청 수 제한(CONCURRENT\_REQUESTS 감소).

**2.4 5xx (서버 오류)**

* **500 Internal Server Error**:
  + 서버에서 요청 처리 중 알 수 없는 문제가 발생.
  + 서버의 문제이므로 요청을 반복하거나 일정 시간 후 다시 시도.
* **502 Bad Gateway**:
  + 서버가 다른 서버로부터 잘못된 응답을 받음.
* **503 Service Unavailable**:
  + 서버가 과부하 상태이거나 유지보수 중으로 요청을 처리할 수 없음.
  + 크롤링 속도를 줄이고 일정 시간 대기 후 재시도.
* **504 Gateway Timeout**:
  + 서버가 다른 서버로부터 응답을 받지 못함.
  + 서버 연결이 불안정한 경우 발생.

**3. Scrapy에서 상태 코드 처리 방법**

**3.1 Scrapy 로그에서 상태 코드 확인**

Scrapy는 각 요청에 대한 HTTP 상태 코드를 출력합니다:

plaintext

코드 복사

[scrapy.core.engine] DEBUG: Crawled (200) <GET https://example.com>

[scrapy.downloadermiddlewares.retry] DEBUG: Retrying <GET https://example.com> (failed 429 Too Many Requests)

[scrapy.core.engine] INFO: Crawled (403) <GET https://example.com/forbidden>

**3.2 상태 코드 처리**

* **200 OK**: 데이터 처리.
* **429 Too Many Requests**: 속도 제한:
  + DOWNLOAD\_DELAY 증가.
  + CONCURRENT\_REQUESTS 감소.
  + AUTOTHROTTLE\_ENABLED 활성화.
* **403 Forbidden**: User-Agent 변경 또는 프록시 사용.
* **503 Service Unavailable**: 일정 시간 대기 후 요청 재시도.

**4. 요약**

| **상태 코드** | **의미** | **조치** |
| --- | --- | --- |
| **200** | 성공 (데이터 반환) | 정상적으로 데이터를 처리. |
| **429** | 요청이 너무 많아 차단됨 | 요청 속도 줄이기(DOWNLOAD\_DELAY), 동시 요청 수 줄이기(CONCURRENT\_REQUESTS). |
| **403** | 접근 거부 (Forbidden) | User-Agent 변경, IP 변경(프록시 사용). |
| **503** | 서버 과부하 또는 유지보수 | 일정 시간 대기 후 재시도(RETRY\_TIMES 설정). |
| **404** | 리소스를 찾을 수 없음 | URL 또는 요청 파라미터 확인. |

**5. 크롤링 안정성을 위한 팁**

* 요청 간 대기 시간을 충분히 설정하여 **429 상태**를 방지.
* 네이버 블로그의 **robots.txt**를 준수하여 **403 상태**를 피함.
* Scrapy의 **AutoThrottle** 기능을 사용해 서버 상태에 따라 요청 속도를 자동으로 조절.

scrapy crawl naver\_blog\_spider \

-o naver\_blog\_2024\_11\_27.json \

-o naver\_blog\_2024\_11\_27.csv \

--logfile naver\_blog\_2024\_11\_27.log \

--loglevel INFO \

-s JOBDIR=crawls/naver\_blog

-s JOBDIR=crawls/naver\_blog는 Scrapy에서 \*\*작업 디렉토리(Job Directory)\*\*를 지정하는 설정입니다.

이 설정은 Scrapy가 크롤링 작업 상태를 저장하고 관리할 수 있도록 해줍니다.

**작동 원리**

1. **JOBDIR 설정**
   * Scrapy는 크롤링 작업을 진행하면서 **진행 상태를 지정된 디렉토리에 저장**합니다.
   * 설정된 경로(crawls/naver\_blog)에 **중간 상태 파일**들이 생성됩니다.
   * 작업 중단 후 재개할 때 이 파일을 사용하여 **크롤링을 처음부터 다시 시작하지 않고** 중단된 지점부터 계속 진행합니다.
2. **기본 동작**
   * Scrapy는 크롤링 작업 중 URL 요청 상태, 이미 처리된 페이지, 대기 중인 페이지 등을 기록합니다.
   * 예를 들어, Scrapy가 100개의 페이지 중 50개를 크롤링한 후 작업이 중단되면, JOBDIR이 설정된 경우 다시 시작할 때 **나머지 50개만 크롤링**합니다.

**주요 이점**

1. **작업 중단 후 재개**
   * 네트워크 문제, 시스템 재부팅, 의도적인 작업 중단 등의 상황에서 중단된 작업을 이어서 실행할 수 있습니다.
2. **효율적인 크롤링**
   * 이미 크롤링된 URL을 다시 요청하지 않으므로 시간과 리소스를 절약할 수 있습니다.
3. **대규모 크롤링 관리**
   * 매우 많은 페이지를 크롤링할 때 작업을 여러 번에 나누어 수행할 수 있습니다.

**파일 저장 위치 및 내용**

JOBDIR=crawls/naver\_blog로 설정하면 Scrapy는 다음과 같은 내용을 crawls/naver\_blog 디렉토리에 저장합니다:

* **requests.queue**: 대기 중인 요청 목록.
* **requests.seen**: 이미 처리된 요청 목록.
* **spider.state**: Spider의 현재 상태.

**실행 예시**

1. **첫 번째 실행**:

scrapy crawl naver\_blog\_spider -s JOBDIR=crawls/naver\_blog

* + 크롤링 작업을 시작하고 진행 상태가 crawls/naver\_blog 디렉토리에 저장됩니다.

1. **작업 중단 후 재개**:
   * 작업을 중단한 뒤 동일한 명령어로 다시 실행:

scrapy crawl naver\_blog\_spider -s JOBDIR=crawls/naver\_blog

* + 이전에 완료된 작업을 건너뛰고, 중단된 지점부터 다시 시작합니다.

**주의 사항**

1. **JOBDIR 디렉토리를 삭제하면 재개 불가**:
   * 작업 상태 파일이 손실되면 크롤링 작업을 처음부터 다시 시작해야 합니다.
   * 따라서 중단된 작업을 이어서 실행하려면 해당 디렉토리를 삭제하지 않아야 합니다.
2. **경로 지정**:
   * crawls/naver\_blog는 상대 경로입니다. 절대 경로나 다른 디렉토리도 지정할 수 있습니다:

-s JOBDIR=/absolute/path/to/jobdir

1. **동시에 여러 JOBDIR 사용 금지**:
   * 동일한 Spider에 대해 다른 JOBDIR 설정을 사용하는 경우, 작업 상태가 섞여 예상치 못한 결과를 초래할 수 있습니다.

**요약**

-s JOBDIR=crawls/naver\_blog는 작업 진행 상태를 저장하여 **크롤링 작업을 중단 후 재개할 수 있도록 하는 설정**입니다. 크롤링이 중단될 가능성이 있는 대규모 작업에서 특히 유용합니다.

Scrapy의 --loglevel 옵션은 **크롤링 중 생성되는 로그 메시지의 상세 수준**을 설정하는 데 사용됩니다. 이 설정을 통해 원하는 중요도 이상의 로그만 출력하도록 조정할 수 있습니다.

**1. Scrapy의 로그 레벨**

Scrapy는 Python의 표준 **logging** 라이브러리를 기반으로 하며, 다음과 같은 로그 레벨을 지원합니다:

| **로그 레벨** | **설명** | **권장 상황** |
| --- | --- | --- |
| **CRITICAL** | 치명적인 오류만 출력. Scrapy 실행이 중단될 정도의 심각한 문제만 표시. | 시스템 안정성을 확인하거나 크롤러 실패 감지 시. |
| **ERROR** | 크롤링 실패나 주요 문제가 발생했을 때만 출력. | 오류 메시지만 필요한 경우. |
| **WARNING** | 경고 메시지와 그 이상의 중요도를 가진 메시지 출력. | 설정 오류나 성능 문제를 감지하고자 할 때. |
| **INFO** | 크롤러의 일반 실행 상태를 표시. Spider가 실행되거나 요청이 성공한 경우 등. | 기본 설정. 일반적인 실행 상태를 확인하고 싶을 때. |
| **DEBUG** | 상세한 디버깅 메시지. 요청/응답 세부 정보, 설정 값, 중간 데이터 등 모두 출력. | 크롤러를 개발 중이거나 문제를 깊이 조사할 때. |
| **NOTSET** | 로그 레벨 제한 없이 모든 메시지 출력. | 사용 권장하지 않음. |

**2. 권장 로그 레벨**

**2.1 초보자 또는 일반 사용**

* **추천 로그 레벨**: **INFO**
  + 실행 상태를 간결하게 확인할 수 있습니다.
  + 일반적인 크롤링 결과를 모니터링하기에 적합합니다.
* **예제**:

scrapy crawl naver\_blog\_spider --loglevel INFO

**2.2 크롤링 문제 조사 (디버깅)**

* **추천 로그 레벨**: **DEBUG**
  + 요청/응답과 관련된 모든 세부 정보를 표시.
  + 크롤링 도중 발생한 문제를 자세히 분석하고 해결할 때 적합.
* **예제**:

scrapy crawl naver\_blog\_spider --loglevel DEBUG

**2.3 실무 운영 또는 자동화**

* **추천 로그 레벨**: **WARNING**
  + 성공적인 요청 로그는 제외하고, 경고 및 오류 메시지만 출력.
  + 로그 파일 크기를 줄이고, 중요한 문제만 모니터링할 때 유용.
* **예제**:

scrapy crawl naver\_blog\_spider --loglevel WARNING

**2.4 치명적인 문제만 필요**

* **추천 로그 레벨**: **ERROR** 또는 **CRITICAL**
  + 크롤러의 중단 원인을 파악하는 데 집중.
  + 운영 중 로그를 최소화하고 주요 문제만 분석할 때 사용.
* **예제**:

scrapy crawl naver\_blog\_spider --loglevel ERROR

**3. 추천 로그 레벨 요약**

| **목적** | **권장 로그 레벨** |
| --- | --- |
| 기본적인 실행 상태 확인 | INFO |
| 크롤링 문제를 조사하거나 디버깅할 때 | DEBUG |
| 주요 문제만 확인하고자 할 때 | WARNING |
| 운영 환경에서 로그를 최소화할 때 | ERROR 또는 CRITICAL |

**4. 추가 팁**

* **로그 파일 저장**: 로그를 저장하면서 로그 레벨을 설정하려면 다음과 같이 사용:

scrapy crawl naver\_blog\_spider --logfile crawl.log --loglevel INFO

* **실행 중 변경**: 크롤링 중 로그 수준을 동적으로 변경할 수는 없습니다. 따라서 크롤링 시작 전에 적절한 로그 레벨을 설정해야 합니다.

**5. 결론**

* 개발 및 디버깅 단계에서는 **DEBUG**.
* 일반 크롤링과 실행 확인에는 **INFO**.
* 운영 환경에서는 **WARNING** 또는 **ERROR**를 권장합니다.

docker-compose up -d  
  -d 옵션은 Docker Compose에서 "detach mode"를 의미  
  docker-compose up 명령 실행 후 터미널을 차단하지 않고 백그라운드에서 컨테이너 실행

[동작 차이]  
1. Without -d  
  컨테이너 로그가 현재 터미널에 출력됩니다.  
  터미널이 해당 프로세스를 차단합니다.  
 예:  
  docker-compose up  
 출력:  
  Creating network "example\_default" with the default driver  
  Creating example\_service ... done  
  Attaching to example\_service  
  example\_service | Starting service...  
 사용 사례:  
  디버깅이나 로그를 실시간으로 확인하고 싶을 때.  
2. With -d  
  컨테이너가 백그라운드에서 실행됩니다.  
  터미널이 즉시 반환되어 다른 작업을 계속할 수 있습니다.  
  로그를 보려면 추가 명령이 필요합니다  
  (예: docker logs 또는 docker-compose logs).  
 예:  
  docker-compose up -d  
 출력:  
  Creating network "example\_default" with the default driver  
  Creating example\_service ... done  
 추가 작업: 로그 확인[사용 시기]  
-d 사용:  
  컨테이너가 정상적으로 실행 중인지 확인만 하고, 이후 작업을 계속 진행하고 싶을 때.  
  로그가 필요하지 않거나 나중에 별도로 확인할 때.  
-d 미사용:  
  컨테이너 실행 중 발생하는 문제를 실시간으로 확인하고 싶을 때.  
  로그를 바로 보고 디버깅할 때.

docker-compose는 여러 컨테이너로 구성된 애플리케이션을 정의하고 관리하기 위한 도구입니다.

docker-compose 명령은 주로 **docker-compose.yml** 파일을 기반으로 작업을 수행하며, 프로젝트 단위로 컨테이너와 네트워크, 볼륨을 관리할 수 있습니다.

**1. 주요 docker-compose 명령**

**1.1 docker-compose up**

* **용도**:
  + docker-compose.yml 파일에 정의된 모든 서비스(컨테이너)를 시작합니다.
  + 필요한 경우 이미지를 빌드하고 네트워크와 볼륨도 생성합니다.
* **옵션**:
  + -d: 백그라운드에서 컨테이너 실행.
  + --build: 항상 새로 이미지를 빌드하고 실행.
  + --force-recreate: 기존 컨테이너를 재사용하지 않고 항상 새로 생성.

**1.2 docker-compose down**

* **용도**:
  + 현재 프로젝트의 모든 컨테이너를 중단하고 삭제합니다.
  + 연결된 네트워크와 익명 볼륨도 삭제합니다.
* **옵션**:
  + -v: 컨테이너가 사용한 익명 볼륨도 함께 삭제.

**1.3 docker-compose ps**

* **용도**:
  + 현재 실행 중인 컨테이너의 상태를 확인합니다.
* **-a 옵션 지원하지 않음.**
  + 현재 docker-compose에는 **ps 명령**이 있지만, docker-compose ps 명령은 **docker ps의 일부 옵션(예: -a)을 지원하지 않습니다**.
  + 따라서 docker-compose ps로는 모든 컨테이너(종료된 것 포함)를 확인할 수 없으며, 실행 중인 컨테이너만 표시됩니다.
* **예제 출력**:

Name Command State Ports

---------------------------------------------------------------------

example\_service\_1 python app.py Up 0.0.0.0:5000->5000/tcp

**1.4 docker-compose exec**

* **용도**:
  + 실행 중인 컨테이너 내부에서 명령어를 실행합니다.
  + 특정 서비스의 컨테이너를 대상으로 합니다.
* **예제**:

docker-compose exec scrapy bash

위 명령은 scrapy 서비스의 컨테이너 내부에서 Bash 셸을 실행합니다.

**1.5 docker-compose run**

* **용도**:
  + 새로운 컨테이너를 생성하고 명령을 실행합니다.
  + 기존 컨테이너와는 별도로 동작하며, 임시 컨테이너를 실행할 때 유용합니다.
* **예제**:

docker-compose run scrapy bash

위 명령은 scrapy 서비스 정의를 기반으로 새로운 컨테이너를 생성하고 Bash 셸을 실행합니다.

**1.6 docker-compose logs**

* **용도**:
  + 모든 서비스의 로그를 출력합니다.
* **옵션**:
  + -f: 로그를 스트리밍하여 실시간으로 출력.
  + 서비스 이름: 특정 서비스의 로그만 출력.
* **예제**:

docker-compose logs -f scrapy

**1.7 docker-compose build**

* **용도**:
  + docker-compose.yml의 정의를 기반으로 이미지를 빌드합니다.
* **옵션**:
  + --no-cache: 캐시를 사용하지 않고 새로 이미지를 빌드.
  + 서비스 이름: 특정 서비스만 빌드.

**1.8 기타 명령**

* **docker-compose restart**: 정의된 컨테이너를 재시작.
* **docker-compose stop**: 컨테이너를 중단.
* **docker-compose start**: 중단된 컨테이너를 다시 시작.
* **docker-compose rm**: 정의된 컨테이너를 삭제.

**2. docker 명령이 필요한 경우**

**2.1 개별 컨테이너 작업**

docker-compose는 프로젝트 단위로 작업하기 때문에 개별 컨테이너나 이미지에 직접 작업할 때는 docker 명령을 사용해야 합니다.

* **이미지 삭제**:

docker rmi <이미지 ID 또는 이름>

* **컨테이너 삭제**:

docker rm <컨테이너 ID 또는 이름>

**2.2 모든 컨테이너나 이미지 관리**

docker-compose는 프로젝트에 속하지 않는 컨테이너나 이미지를 관리하지 않습니다. 예를 들어:

* 실행 중인 모든 컨테이너를 확인:

ocker ps -a

* 모든 이미지를 확인:

docker images

* 모든 네트워크 확인:

docker network ls

**2.3 프로젝트 외부 작업**

* 개별 컨테이너의 상세 정보 확인:

docker inspect <컨테이너 ID 또는 이름>

* 로그 확인:

docker logs <컨테이너 ID 또는 이름>

**3. 요약**

**docker-compose로 할 수 있는 작업:**

* 프로젝트 단위 컨테이너 실행, 중지, 로그 확인, 내부 접속 등.
* 동일한 docker-compose.yml 설정 기반으로 컨테이너와 네트워크를 관리.

**docker 명령이 필요한 경우:**

* 프로젝트 외부의 컨테이너나 이미지를 관리할 때.
* 모든 컨테이너/이미지/네트워크의 전역 관리.
* docker-compose로 제공되지 않는 고급 작업(예: 볼륨 마운트 변경, 디스크 사용량 확인 등).

**명령어 분석: docker-compose exec scrapy scrapy crawl naver\_blog\_spider**

이 명령어는 Docker Compose 환경에서 Scrapy 크롤러를 실행하기 위해 사용됩니다.

각 구성 요소의 의미를 자세히 설명하겠습니다.

**1. docker-compose**

* Docker Compose는 여러 Docker 컨테이너를 정의하고 실행하기 위한 도구입니다.
* 프로젝트의 docker-compose.yml 파일에 정의된 서비스들을 관리합니다.
* 이 명령어를 통해 특정 컨테이너 내부에서 명령을 실행할 수 있습니다.

**2. exec**

* Docker Compose 명령어 중 하나로, 실행 중인 컨테이너 내부에서 명령을 실행합니다.
* 여기서 exec는 **이미 실행 중인 컨테이너**에 접근하여 추가로 명령을 수행하는 역할을 합니다.
  + 예: 컨테이너 내부에서 특정 파일을 실행하거나, 디버깅 목적으로 쉘에 접속할 때 사용.

**명령 형식:**

docker-compose exec <SERVICE\_NAME> <COMMAND>

* **SERVICE\_NAME**: docker-compose.yml에서 정의된 서비스 이름. 여기서는 scrapy가 서비스 이름입니다.
* **COMMAND**: 컨테이너 내부에서 실행할 명령. 여기서는 scrapy crawl naver\_blog\_spider가 실행됩니다.

**3. scrapy (서비스 이름)**

* docker-compose.yml 파일에서 정의된 서비스 이름입니다.
* 예를 들어, 다음과 같은 docker-compose.yml 파일이 있다면:

services:

scrapy:

build: .

container\_name: naver-blog-crawler

volumes:

- .:/app

command: scrapy crawl naver\_blog\_spider

* + scrapy는 컨테이너 이름이 아니라, **서비스 이름**으로 정의된 것입니다.
  + 따라서 docker-compose exec scrapy는 이 서비스를 기반으로 생성된 컨테이너에서 명령을 실행합니다.

**4. scrapy crawl naver\_blog\_spider**

* Scrapy 명령어로, Scrapy 프로젝트 내부에서 크롤러를 실행하는 데 사용됩니다.

**세부 구성:**

1. **scrapy**:
   * Scrapy 실행 명령어입니다. Scrapy는 크롤링 프레임워크로, HTTP 요청, 데이터 추출, 저장 등을 처리합니다.
2. **crawl**:
   * Scrapy 명령어 중 하나로, 특정 크롤러를 실행합니다.
   * 프로젝트의 spiders 디렉토리 내에 정의된 크롤러 중 하나를 호출합니다.
3. **naver\_blog\_spider**:
   * Scrapy 프로젝트 내에서 정의된 크롤러 이름입니다.
   * 일반적으로 spiders 디렉토리의 Python 클래스에서 정의된 name 속성과 일치해야 합니다.

class NaverBlogSpider(scrapy.Spider):

name = "naver\_blog\_spider"

**전체 명령어의 의미**

docker-compose exec scrapy scrapy crawl naver\_blog\_spider

이 명령어는 다음을 수행합니다:

1. **docker-compose exec**:
   * 실행 중인 Docker Compose 서비스(컨테이너)에 접근하여 명령을 실행.
2. **scrapy (서비스 이름)**:
   * docker-compose.yml에서 정의된 scrapy 서비스를 기반으로 실행 중인 컨테이너에서 명령을 수행.
3. **scrapy crawl naver\_blog\_spider**:
   * 컨테이너 내부에서 Scrapy를 실행하여 naver\_blog\_spider라는 이름의 크롤러를 실행.

**결론**

* **docker-compose**: Docker Compose를 통해 컨테이너를 관리.
* **exec**: 실행 중인 컨테이너 내부에서 명령을 실행.
* **scrapy (서비스 이름)**: docker-compose.yml에 정의된 서비스 이름.
* **scrapy crawl naver\_blog\_spider**: Scrapy 명령어로 크롤러 실행.