airflow 의 kubernetesPodOperator 에 대해서 아주 상세한 설명과 사용법, 예시를 부탁해.

좋습니다. Kubernetes 환경에서 Airflow로 "컨테이너 하나를 잠깐 띄워서 어떤 일을 시키고 끝내기"에 가장 적합한 연산자가 바로 **`KubernetesPodOperator`**(이하 KPO)입니다. 아래에 **개념 → 연결/권한 → 핵심 파라미터 → 자주 쓰는 패턴/예제 → 운영 팁/트러블슈팅** 순서로 정리했습니다.
(사용 버전은 Airflow 2.6+ 기준으로 설명하며, 2.8/2.9의 변화도 각주처럼 함께 짚습니다.)

---

1) 무엇을 해결하나요?

- **컨테이너 단위의 작업 실행:** 특정 이미지로 파드를 만들고, 명령을 실행하고, 로그/결과(XCom)를 회수한 뒤 파드를 정리합니다.
- **언어/런타임 제약이 없음:** Python 환경이 아닌 R, Java, Shell, Spark-submit 등 *이미지에만 들어 있으면* 무엇이든 실행 가능합니다.
- **격리와 확장성:** 각 태스크가 **독립 파드**로 실행되어, 리소스/의존성 충돌을 최소화합니다.

---

2) 연결(Conn) & 권한(RBAC)

2.1 연결 방식 3가지

1. **in-cluster**: Airflow가 K8s 안에 있을 때, 파드 내의 ServiceAccount로 접근 (**`in_cluster=True`**, 기본값이 자동 감지).
2. **kubeconfig 파일**: 외부에서 접속할 때 **`config_file="~/.kube/config"`**, **`cluster_context="minikube"`** 등.
3. **Airflow Connection (kubernetes)**: **`kubernetes_conn_id="k8s_default"`**를 추천. kubeconfig/토큰/인증서를 이 Conn에 넣어 재사용합니다.

   운영팁: 팀 표준으로 **Connection 하나**를 정해두면 DAG 코드가 깔끔해집니다.

2.2 권한

- 실행 주체는 Airflow의 Webserver/Worker(또는 KubernetesExecutor/Local/Celery Worker)가 **타깃 네임스페이스에 파드를 생성/조회/삭제**할 권한이 있어야 합니다.
- 최소 **Role + RoleBinding**(네임스페이스 한정) 또는 **ClusterRole + ClusterRoleBinding**(여러 NS 필요 시)을 부여하세요.

---

3) 핵심 파라미터 빠르게 보기 (KPO)

자주 쓰는 것만 실무 감각으로 요약합니다.

- **`task_id`**: 태스크 이름

- **`namespace`**: 파드를 만들 네임스페이스
- **`image`**: 컨테이너 이미지 (예: **`alpine:3.20`**, **`python:3.11-slim`**)
- **`cmds`**, **`arguments`**: 컨테이너 엔트리포인트/인자 (리스트)
- **`env_vars`**, **`env_from`**: 환경변수 지정/ConfigMap/Secret 가져오기
- **`volumes`**, **`volume_mounts`**: PVC/NFS/ConfigMap/Secret 등을 마운트
- **`resources`**: **`k8s.V1ResourceRequirements`**(requests/limits)
- **`affinity`**, **`tolerations`**, **`node_selector`**: 스케줄링 제어
- **`image_pull_secrets`**, **`image_pull_policy`**
- **`labels`**, **`annotations`**: 운영/감사 목적 라벨링
- **`get_logs=True`**: 파드 stdout/stderr를 Airflow 로그에 스트리밍
- **`do_xcom_push=True`**: 컨테이너가 **`/airflow/xcom/return.json`**에 쓴 값을 XCom으로 수집
- **`on_finish_action`**: **2.8+ 권장** 파드 정리 정책
  - **`"delete_pod"`**: 항상 삭제(가장 흔함)
  - **`"keep_pod"`**: 남겨둠(디버깅 시)
  - **`"delete_succeeded_pods"`**: 성공 시만 삭제

  구버전의 **`is_delete_operator_pod=True`**는 **더 이상 권장되지 않습니다**.
- **`config_file`**, **`in_cluster`**, **`cluster_context`**, **`kubernetes_conn_id`**: 접속 방식
- **`pod_template_file`**: 미리 정해둔 **Pod 템플릿 YAML**과 병합
- **`full_pod_spec`**/**`pod_override`**: Python 객체/부분 덮어쓰기

---

4) 최소 예제 3종

4.1 가장 미니멀 (BusyBox 한 줄)

```
python


from datetime import datetime
from airflow import DAG
from airflow.providers.cncf.kubernetes.operators.kubernetes_pod import KubernetesPodOperator

with DAG(
    dag_id="kpo_minimal_busybox",
    start_date=datetime(2024, 1, 1),
    schedule=None,
    catchup=False,
) as dag:
    hello = KubernetesPodOperator(
        task_id="echo_hello",
        namespace="default",
        image="busybox:1.36",
        cmds=["/bin/sh", "-c"],
        arguments=["echo 'hello from kpo' && sleep 3"],
        get_logs=True,
        on_finish_action="delete_pod",  # Airflow 2.8+ 권장
    )
```

4.2 XCom으로 결과 받기 (JSON 파일에 쓰기)

```
python


import json
from datetime import datetime
from airflow import DAG
from airflow.providers.cncf.kubernetes.operators.kubernetes_pod import KubernetesPodOperator

# 컨테이너 안에서 /airflow/xcom/return.json 에 {"result":"..."} 형태로 쓰면 XCom으로 들어옵니다.
XCOM_PATH = "/airflow/xcom/return.json"

with DAG(
    "kpo_xcom_example",
    start_date=datetime(2024, 1, 1),
    schedule=None,
    catchup=False,
) as dag:
    xcom_task = KubernetesPodOperator(
        task_id="produce_xcom",
        namespace="default",
        image="python:3.11-slim",
        cmds=["python", "-c"],
        arguments=[f"""
import json, os
payload = {{"result":"ok", "value": 42}}
os.makedirs(os.path.dirname("{XCOM_PATH}"), exist_ok=True)
with open("{XCOM_PATH}", "w") as f: json.dump(payload, f)
print("wrote:", payload)
"""],
        do_xcom_push=True,
        get_logs=True,
        on_finish_action="delete_pod",
    )
```

Airflow에서 **`xcom_task.output`** 또는 **`ti.xcom_pull(task_ids="produce_xcom")`**로 dict를 바로 가져올 수 있습니다.

4.3 리소스/볼륨/시크릿/스케줄링까지 실제형

```
python


from datetime import datetime
from airflow import DAG
from kubernetes.client import models as k8s
from airflow.providers.cncf.kubernetes.operators.kubernetes_pod import KubernetesPodOperator

with DAG(
    "kpo_full_pattern",
    start_date=datetime(2024, 1, 1),
    schedule=None,
    catchup=False,
) as dag:

    # PVC 마운트
    volume = k8s.V1Volume(
        name="data-pvc",
        persistent_volume_claim=k8s.V1PersistentVolumeClaimVolumeSource(claim_name="data-pvc"),
    )
    volume_mount = k8s.V1VolumeMount(
        name="data-pvc",
        mount_path="/data",
        read_only=False,
    )

    # 리소스
```

```
    resources = k8s.V1ResourceRequirements(
        requests={"cpu": "250m", "memory": "256Mi"},
        limits={"cpu": "1", "memory": "1Gi"},
    )

    # Toleration/NodeSelector/Affinity
    tolerations = [k8s.V1Toleration(key="workload", operator="Equal", value="batch",
effect="NoSchedule")]
    node_selector = {"kubernetes.io/os": "linux"}
    affinity = k8s.V1Affinity(
        node_affinity=k8s.V1NodeAffinity(
            required_during_scheduling_ignored_during_execution=k8s.V1NodeSelector(
                node_selector_terms=[k8s.V1NodeSelectorTerm(
                    match_expressions=[k8s.V1NodeSelectorRequirement(key="nodepool",
operator="In", values=["general"])]
                )]
            )
        )
    )

    # Secret을 env로 주입 (K8s Secret: mysecret, key: ACCESS_KEY)
    env_vars = {"S3_ENDPOINT": "http://minio:9000"}
    env_from = [k8s.V1EnvFromSource(secret_ref=k8s.V1SecretEnvSource(name="mysecret"))]

    task = KubernetesPodOperator(
        task_id="batch_etl",
        namespace="data",
        image="python:3.11-slim",
        cmds=["bash", "-lc"],
        arguments=[
            "python - <<'PY'\n"
            "import os, time\n"
            "print('endpoint:', os.getenv('S3_ENDPOINT'))\n"
            "print('access:', os.getenv('ACCESS_KEY'))\n"
            "print('listing /data ...', os.listdir('/data'))\n"
            "time.sleep(2)\n"
            "PY"
        ],
        env_vars=env_vars,
        env_from=env_from,
        volumes=[volume],
        volume_mounts=[volume_mount],
        resources=resources,
        tolerations=tolerations,
        node_selector=node_selector,
        affinity=affinity,
        labels={"team": "infra", "job": "etl"},
        get_logs=True,
        image_pull_secrets=["regcred"],  # 프라이빗 레지스트리
        kubernetes_conn_id="k8s_default",  # 조직 표준 연결 권장
        on_finish_action="delete_pod",
    )
```

5) Pod 템플릿/오버레이로 깔끔하게

5.1 Pod 템플릿 파일과 병합

YAML(예: **`pod_template.yaml`**)에 공통 보안/사이드카/환경을 정의해 두고:

```
python

task = KubernetesPodOperator(
    task_id="use_template",
    image="alpine:3.20",
    cmds=["sh", "-c"], arguments=["echo hi"],
    pod_template_file="/opt/airflow/pod_template.yaml",
```

```
    on_finish_action="delete_pod",
)
```

5.2 pod_override로 부분만 덮어쓰기

```
python

from kubernetes.client import models as k8s

override = k8s.V1Pod(
    metadata=k8s.V1ObjectMeta(labels={"trace-id": "1234-xyz"}),
    spec=k8s.V1PodSpec(
        dns_policy="ClusterFirst",
        host_aliases=[k8s.V1HostAlias(ip="10.0.0.10", hostnames=["internal.example.local"])],
    ),
)

task = KubernetesPodOperator(
    task_id="override_bits",
    image="alpine",
    cmds=["sh","-c"], arguments=["nslookup internal.example.local || true"],
    pod_override=override,
    on_finish_action="delete_pod",
)
```

---

6) 패턴별 스니펫

6.1 Git 리포지토리에서 스크립트 가져와 실행

(컨테이너 시작 시 **`git clone`** → 스크립트 실행. 혹은 이미지에 **`git`** 내장 필수)

```
python

KubernetesPodOperator(
    task_id="run_repo_script",
    image="alpine/git",
    cmds=["sh","-lc"],
    arguments=["git clone https://github.com/... repo && cd repo && ./run.sh"],
    get_logs=True, on_finish_action="delete_pod",
)
```

6.2 S3/MinIO와 작업 (환경변수/Secret)

```
python

KubernetesPodOperator(
    task_id="s3_job",
    image="amazon/aws-cli:2.15.25",
    cmds=["bash","-lc"],
    arguments=[
        "aws s3 ls s3://my-bucket --endpoint-url $S3_ENDPOINT"
    ],
    env_vars={"AWS_ACCESS_KEY_ID":"{{ var.value.AWS_KEY }}",
              "AWS_SECRET_ACCESS_KEY":"{{ var.value.AWS_SECRET }}",
              "S3_ENDPOINT":"http://minio:9000"},
    get_logs=True, on_finish_action="delete_pod",
)
```

6.3 Spark-submit 같은 외부 런처

이미지에 **`spark-submit`**가 들어있으면 그대로 호출:

```
python

KubernetesPodOperator(
    task_id="spark_batch",
    image="my-spark-image:latest",
    cmds=["bash","-lc"],
    arguments=["spark-submit --master k8s://... --class Main ..."],
    resources={"request_memory":"2Gi"},  # k8s.V1ResourceRequirements 권장
    get_logs=True, on_finish_action="delete_pod",
)
```

6.4 실패 시 파드 남겨 디버깅

```
python

KubernetesPodOperator(
    task_id="debug_on_fail",
    image="alpine",
    cmds=["sh","-c"], arguments=["exit 1"],
    get_logs=True,
    on_finish_action="keep_pod",  # 파드 남김
)
```

---

7) 운영 팁 (실무에서 자주 겪는 포인트)

1. **파드가 Pending**
   - ImagePull 오류: **`image_pull_secrets`**, 레지스트리 접근권 확인
   - 리소스 부족: **`requests/limits`** 낮추거나 노드 증설
   - PVC 바인딩 실패: PVC/PV 상태 확인
   - Taint/Toleration 불일치: 노드 taint와 toleration 매칭
2. **CrashLoopBackOff**
   - 엔트리포인트/인자 오타, 의존패키지 미설치, 네트워크 DNS 문제 점검
   - **`kubectl logs`**와 **`kubectl describe pod`**에서 이벤트/exit code 확인
3. **로그가 Airflow에 안 찍힘**
   - **`get_logs=True`**인지 확인
   - 컨테이너가 너무 빨리 종료되면 로그 스트림이 짧게 끊길 수 있음 → **`sleep`**으로 유예 또는 작업 자체 로그를 파일로 남기고 마지막에 cat
4. **XCom이 비어 있음**
   - 반드시 **`/airflow/xcom/return.json`**에 **유효한 JSON**을 써야 합니다.
   - 파일 경로 커스터마이즈 시 **`xcom_push_kpo_executor_output`** 옵션(버전에 따라 이름 다름) 또는 문서 확인. 보통 기본 경로 유지가 안전합니다.
5. **파드 정리 정책**

- 개발/디버깅: **`keep_pod`**
- 운영: **`delete_pod`** 또는 **`delete_succeeded_pods`**
- 구 옵션 **`is_delete_operator_pod`**는 2.8+에선 지양

6. **보안/비밀관리**
   - 민감정보는 Airflow Variables보다 **K8s Secret**이 선호됩니다. **`env_from`**으로 깔끔하게 로딩
   - 이미지에 자격증명 bake-in 금지
7. **성능/비용 감각**
   - 짧은 잡이면 이미지 **슬림/로컬 레지스트리** 사용, **`requests`**를 낮춰 스케줄링 지연을 줄이세요.
   - 공통 베이스 이미지를 표준화하면 **캐시 히트율**이 올라 속도 안정화에 도움 됩니다.
8. **템플릿화**
   - 팀 공통 **`pod_template_file`** + 태스크별 **`pod_override`** 조합이 가장 깔끔합니다.
   - 레이블/어노테이션 정책(예: **`team`**, **`owner`**, **`cost-center`**, **`trace-id`**)을 정해두면 추적이 편합니다.

---

8) 체크리스트 (배포 전)

- ☐ **`kubernetes_conn_id`** 또는 in-cluster 접근이 정상 동작한다.
- ☐ 네임스페이스/SA/RBAC 권한이 파드 생성/조회/삭제 가능하다.
- ☐ 이미지 풀 권한(**`image_pull_secrets`**) 및 태그 고정(불변태그 권장).
- ☐ 리소스 **`requests/limits`**가 클러스터 정책과 맞는다.
- ☐ **`get_logs`**, **`on_finish_action`** 원하는 정책으로 설정.
- ☐ (사용 시) PVC/볼륨 마운트가 정상.
- ☐ (사용 시) XCom JSON 경로/형식이 정확.
- ☐ 라벨/어노테이션으로 오너십/추적이 가능.

---

9) 자주 묻는 질문

Q. Celery/LocalExecutor에서도 KPO 쓸 수 있나요?
A. 가능합니다. 작업 파드는 *클러스터 어디서든* 떠요. 다만 Airflow 워커가 K8s에 접속할 **자격**이 있어야 합니다.

Q. 멱등성은 어떻게 담보하나요?
A. 컨테이너 내부 로직에서 입력 키/타임스탬프 기반으로 **재실행 안전**(예: 기존 산출물 덮어쓰기 방지, 트랜잭션/Upsert) 처리를 하세요.

Q. 한 파드에 여러 컨테이너(사이드카)를 쓰고 싶어요.
A. **`pod_template_file`**로 사이드카(예: git-sync, log-shipper)를 넣고, KPO의 메인 컨테이너만 명령을 바꾸면 됩니다.

필요하시면 **Minikube(WSL2) 기준 예제**나 **사내 레지스트리/네임스페이스 정책**에 맞춘 템플릿도 바로 맞춰 드리겠습니다. 지금 환경(네임스페이스 이름, 레지스트리, 접속 방식)만 알려주시면, 상용 배포용 베스트 프랙티스 템플릿으로 깔끔하게 정리해 드릴게요.