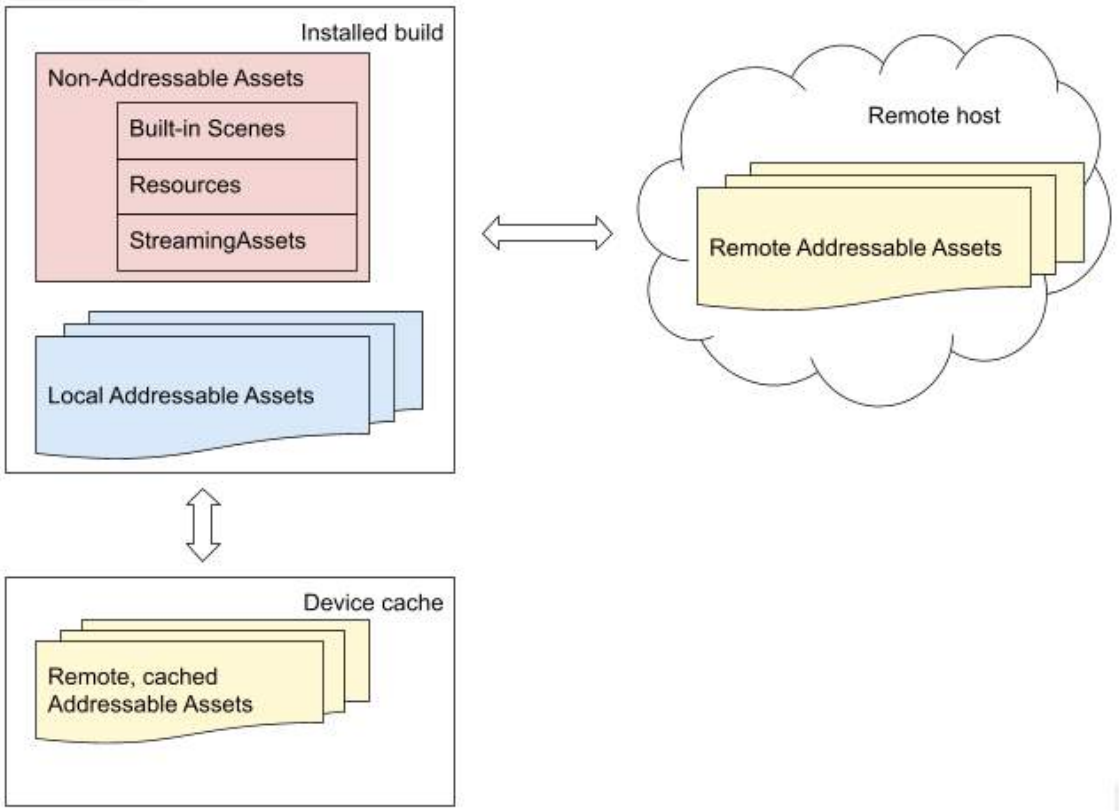
〈Unity Addressables〉

1. 한 줄 요약 / 핵심 역할

Addressables는 애셋을 “주소 기반으로 비동기 로드 / 해제 / 그룹화 / 원격 배포” 가능하게 해 주는 Unity의 리소스 관리 시스템입니다.

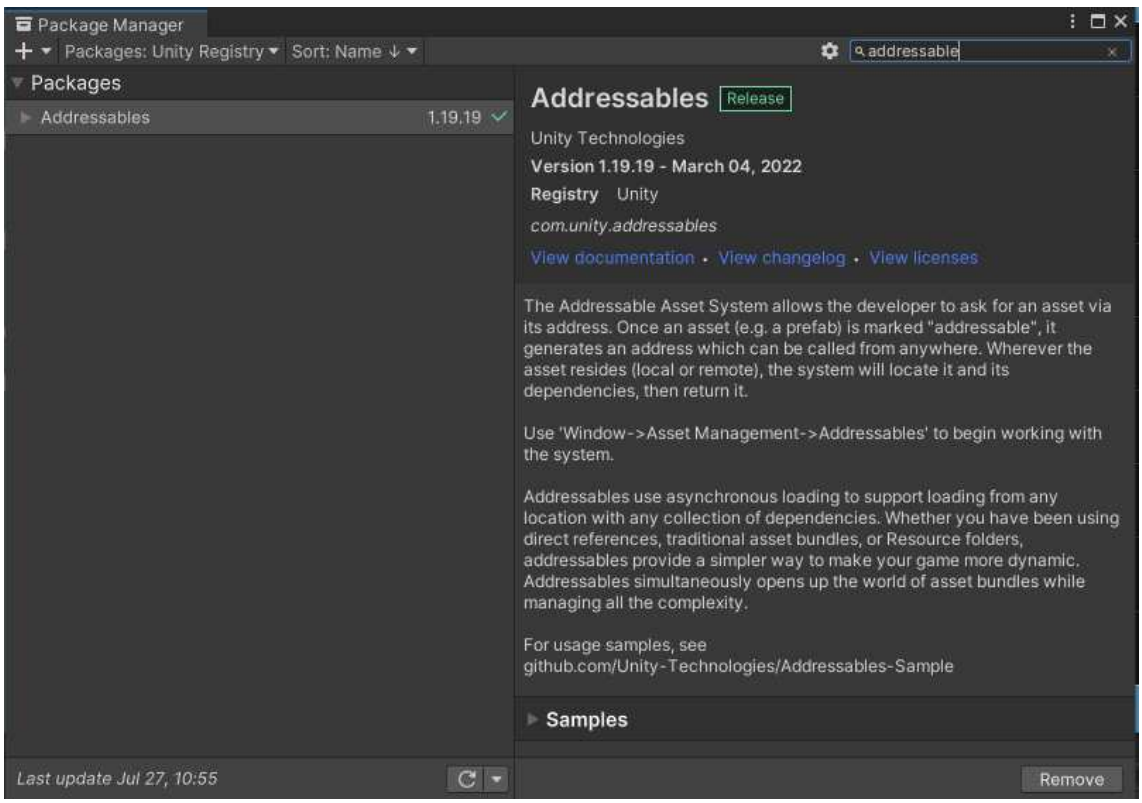
리소스 로딩 / 메모리 관리 / 번들 관리 복잡도를 줄이면서도 대규모 게임의 자산 관리를 유연하게 해 줘요.



2. Addressables가 왜 필요한가?

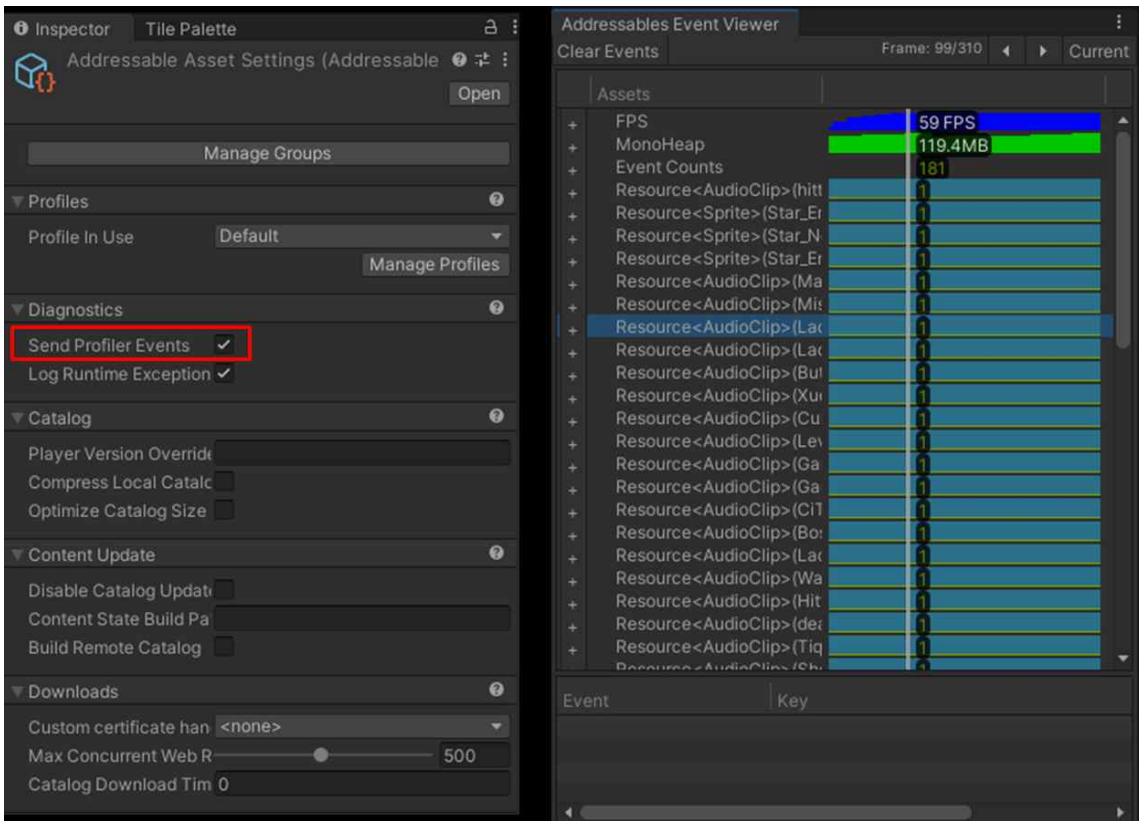
- 기본 방식 (직접 참조나 Resources 폴더) 은 빌드 시 모든 참조된 리소스를 한꺼번에 포함 – 메모리 과부하 / 빌드 크기 증가
- AssetBundle 을 직접 관리하는 방식은 복잡한 의존성 관리 + 스크립트 코드가 많아짐
- Addressables는 AssetBundle 위에 추상화를 제공하면서,
  - 의존성 자동 처리,
  - 비동기 로드,
  - 리소스 해제 관리,
  - 로컬/원격 분리 가능성 등을 제공함

  ([Unity Docs][1])



3. 기본 개념 & 구조

개념	설명
Addressable Asset	Addressables 시스템에 등록된 애셋 (프리팹, 텍스처, 사운드 등)
Address	애셋을 식별하기 위한 문자열 키 (기본은 경로, 커스텀 변경 가능)
Asset Reference	스크립트 또는 인스펙터에서 주소 기반 참조를 나타내는 클래스 타입 (예: AssetReference)
Addressable 그룹(Group)	여러 Addressable 애셋을 묶는 단위 (로컬/원격 설정, 번들 구성 단위)
빌드/번들 컨텐츠(Build Player Content)	Addressables 데이터를 번들로 변환하는 과정 (번들 + 메타 데이터 생성)
비동기 로드/Instantiate	Addressables.LoadAssetAsync<T>(), Addressables.InstantiateAsync() 등을 통해 실행 중에 애셋 가져오기
해제/릴리즈 관리	로드 후 Release() 호출 또는 참조 카운트 기반 관리 필수



4. 설치 및 기본 설정 절차

4.1 Addressables 패키지 설치

Unity〉 Window〉 Package Manager〉 'Addressables' 검색〉 설치

(Unity 2018.3 이상부터 지원) ([Unity Docs][2])

4.2 Addressables 설정 생성

Window〉 Asset Management〉 Addressables〉 Groups 메뉴 열기〉 기본 설정 생성

4.3 Asset을 Addressable로 표시하기

- 프로젝트 창에서 애셋 선택〉 Inspector〉 "Addressable" 체크
- 또는 Addressables Groups 창에서 드래그해서 그룹에 넣기
- 주소(Address) 이름을 명시하거나 기본 경로 사용 가능 ([Unity Docs][2])

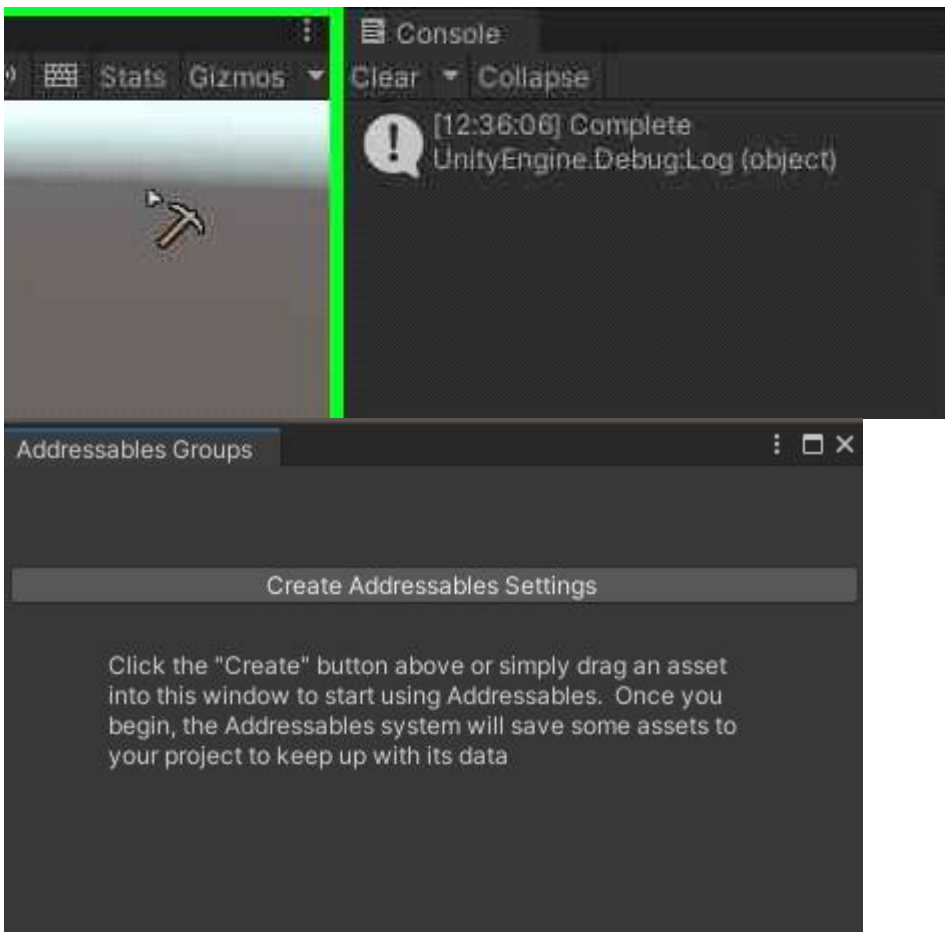
4.4 그룹 설정 조정

- 그룹별 Build / Load 경로 설정
- 로컬 또는 원격(예: CDN) 경로 지정 가능
- "Include in Build"옵션으로 번들 포함 여부 제어

4.5 Build Player Content 수행

- Groups 창〉 Build〉 New Build / Default Build Script

- 이 과정을 통해 Addressables 번들 + 관련 메타데이터가 생성됨 ([Unity Docs][3])



5. 애셋 로드 & 사용 예제

```
using UnityEngine;
using UnityEngine.AddressableAssets;
using UnityEngine.ResourceManagement.AsyncOperations;

public class AddressablesExample : MonoBehaviour
{
    public string addressKey = "MyPrefab";

    void Start()
    {
        // 애셋 로드 (비동기)
        Addressables.LoadAssetAsync<GameObject>(addressKey).Completed        +=
handle =>
        {
            if (handle.Status == AsyncOperationStatus.Succeeded)
            {
                GameObject prefab = handle.Result;
                // 인스턴스화 (씬에 배치)
                Instantiate(prefab);
            }
            else
            {
                Debug.LogError("Addressables load failed: " + addressKey);
            }
        };
    }

    void OnDestroy()
    {
        // 만약 Release 필요하다면
        Addressables.ReleaseInstance(gameObject);
    }
}
```

〉 참고: Addressables는 로드된 애셋의 서브 애셋, 컴포넌트 등을 직접 로드할 수 없고, 전체 GameObject를 로드한 뒤 필요한 컴포넌트를 가져와야 함 ([Unity Docs][2])

6. 메모리 관리 & 해제 전략

- Addressables는 내부적으로 참조 카운트 시스템을 사용
- ‘LoadAssetAsync’ 또는 ‘InstantiateAsync’로 획득한 리소스는 반드시 Release / ReleaseInstance 호출
- 누적된 참조를 해제하지 않으면 메모리 누수 발생
- ‘DownloadDependenciesAsync’ 등을 이용해 미리 그룹 단위 의존성 다운로드 가능
- 프로파일러에서 메모리 사용량 / 누적 카운트 감시 필요 ([Unity Docs][2])

7. 원격 콘텐츠 + CDN / 클라우드 연계

- Addressables는 로컬 리소스 + 원격 리소스 혼합 구성이 가능
- 그룹 설정에서 로드 경로를 원격 URL(CDN)로 지정〉 타임에 다운로드 가능
-Unity의 Cloud Content Delivery (CCD)와 통합해 Addressables 콘텐츠를 클라우드에 배포하는 워크플로우 제공됨 ([docs.unity.com][4])
- 빌드된 Addressables 번들을 CCD에 업로드〉 런타임에서 다운로드 & 캐시 방식
- 서버 쪽 버전 관리(번들 버전, 캐싱 전략)와 함께 설계 필요 ([docs.unity.com][4])

8. 계획 및 베스트 프랙티스 (대규모 프로젝트 관점)

Unity 공식 블로그에서도 Addressables 설계 팁이 정리되어 있는데, 이를 바탕으로 대규모 프로젝트에서 고려해야 할 사항들을 정리했어요: ([Unity][5])

항목	고려 사항/팁
그룹 구조 설계	레벨별, 기능별, 공통 리소스별 그룹 나누기. 너무 넓은 그룹은 변경 시 번들 변화 폭 커짐
의존성 최소화	그룹 간 불필요한 교차 참조 줄이기
자주 쓰는 리소스 vs 드물게 쓰는 리소스	자주 쓰는 리소스는 로컬 포함, 드문 리소스는 원격 그룹으로 분리
빌드 관리	“Incremental Build” 활용, 변경된 번들만 새로 빌드하기
번들 버전 관리/캐시 전략	클라이언트가 이미 다운로드한 번들을 다시 받지 않도록 버전/해시 관리 중요
프리로드/사전 다운로드	레벨 진입 전 주요 의존성을 미리 DownloadDependenciesAsync로 가져오기
로딩 UX 처리	로딩 바, 대기 화면, 프리로딩 타이밍 조절
디버깅 & 모니터링	빌드 번들 구조, 참조 카운트, 메모리 프로파일링 지속적으로 확인

9. 주의할 점 / 한계 & 트랩

- 그룹 변경 시 번들 해시 변경 〉 다른 그룹 번들도 다시 다운로드될 가능 〉 무심코 그룹을 변경하면 사용자에게 재다운로드 유발
- 스크립트 또는 리소스 참조가 비Addressable인 상태면 빌드 시 누락되거나 잘못 복사될 수 있음
- Play Mode 상태 (Editor) 에서 Addressables 를 어떻게 동작할 것인지 설정 (예: 에디터 모드에서 즉시 로드 / 번들 모드)
- 원격 다운로드 실패 시 예외 처리 필요
- 큰 번들을 초기 로드 시 끊김 / 지연 문제 → 번들 크기 분할 고려
- 여러 플랫폼 빌드 시 각 플랫폼별 Addressables 설정 확인