

# 자동화와 함께하는 오픈소스


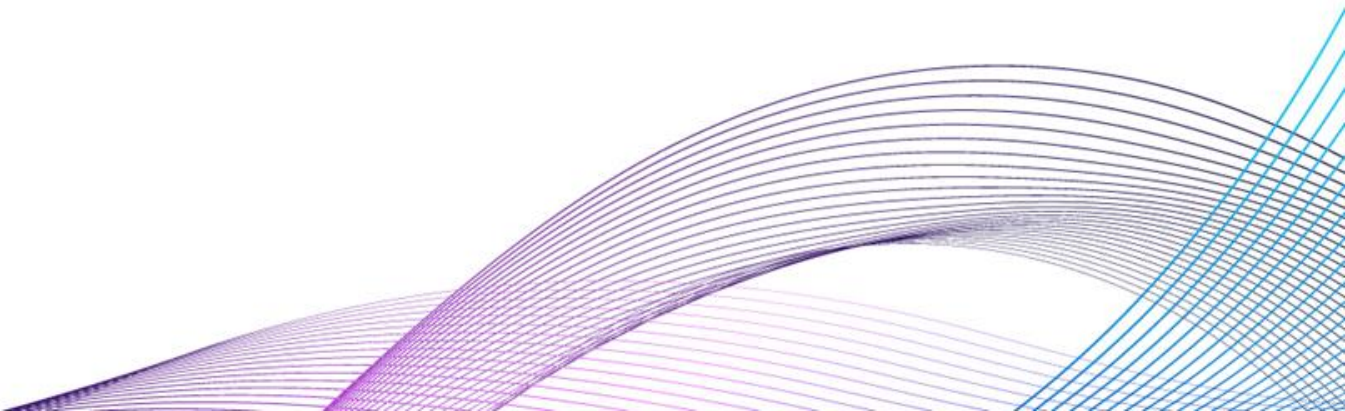
Open-source with automation

신정규  
래블업 주식회사



# CONTENTS



- 
- 01** 오픈소스가 자라는 과정
  - 02** 오픈소스와 자동화의 필요성
  - 03** 실제 자동화 적용하기
  - 04** 마치며
- 



# 01

## 오픈소스가 자라는 과정

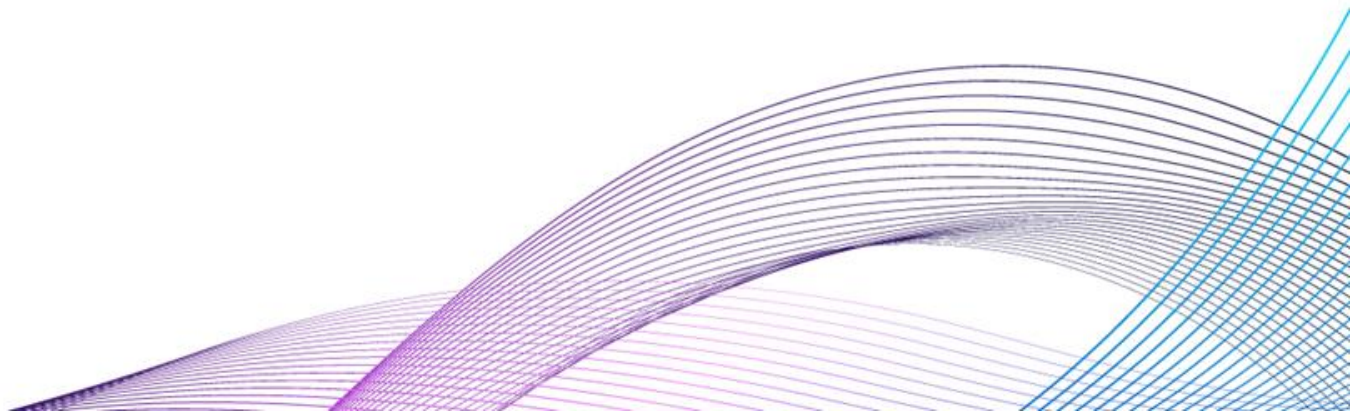




# 01 오픈소스 시작하기

여러 이유로 시작하게 됩니다.

- 어쩌다 보니...
- 학교 과제라...
- 위에서 시켜서...
- 시험 기간이라...
- 나도 명성을...



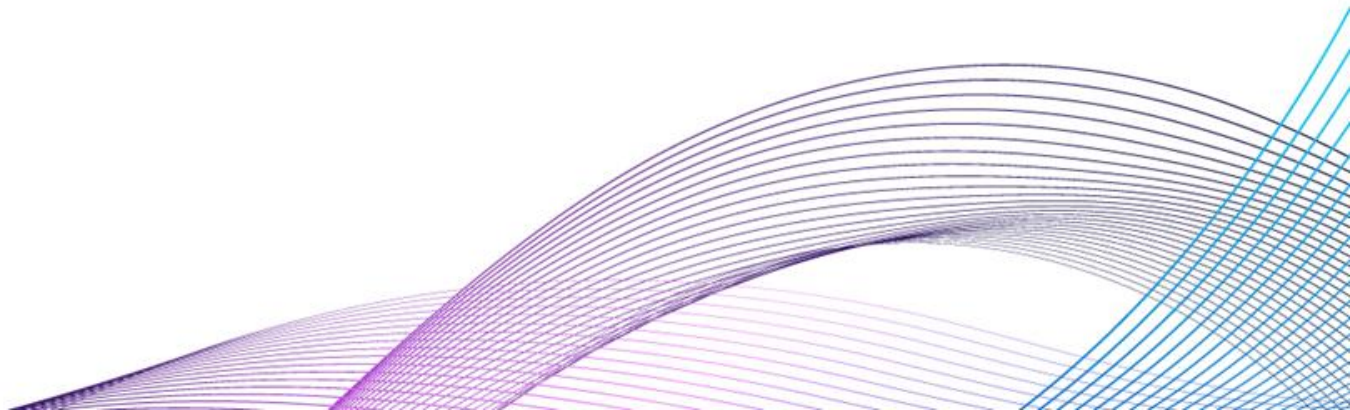




# 01 잘되면

## 점점 커지죠

- 어쩌다 보니... 엇 사용자가 많아졌네
- 학교 과제라... 엥 그 기능이 필요했나?
- 위에서 시켜서... 우와 이걸 사람들이 진짜 쓰는구나
- 시험 기간이라... 시험 끝나서 놀아야 하는데 왜 자꾸 요청이..?
- 나도 명성을... 유튜브 찍을까?

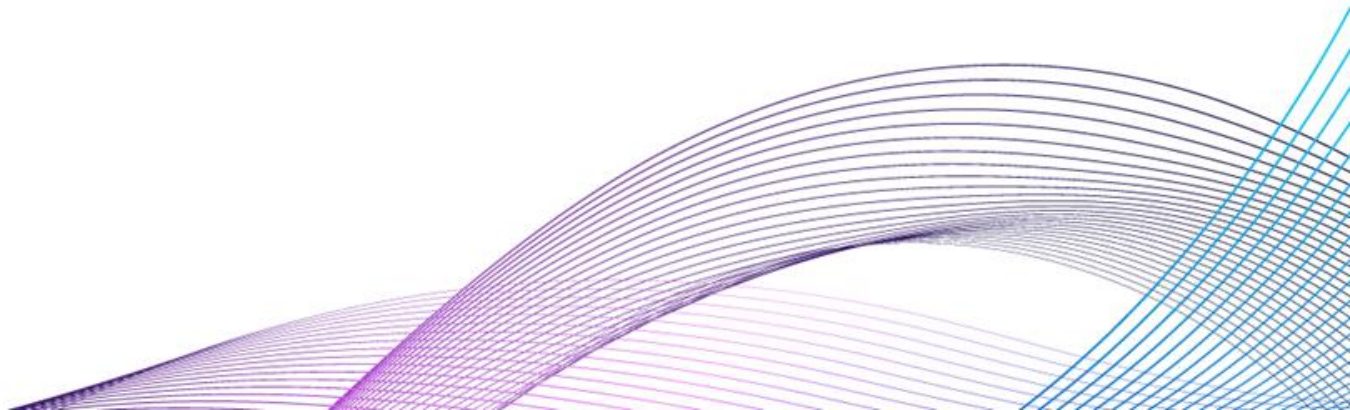




## 지치는 시점

사람이 할 일이 너무 많아집니다

- 늘어나는 코드 작성 요청
- 즐거웠던 PR이 코드 리뷰 일거리로 변신
- 참 쉽게 싸우는 사람들
  - (코드 줄바꿈 문자를 뭘 쓰는지 결정하는 게 뭐가 그렇게 중요해?)
- 버그, 기능, 패키징, 릴리즈...

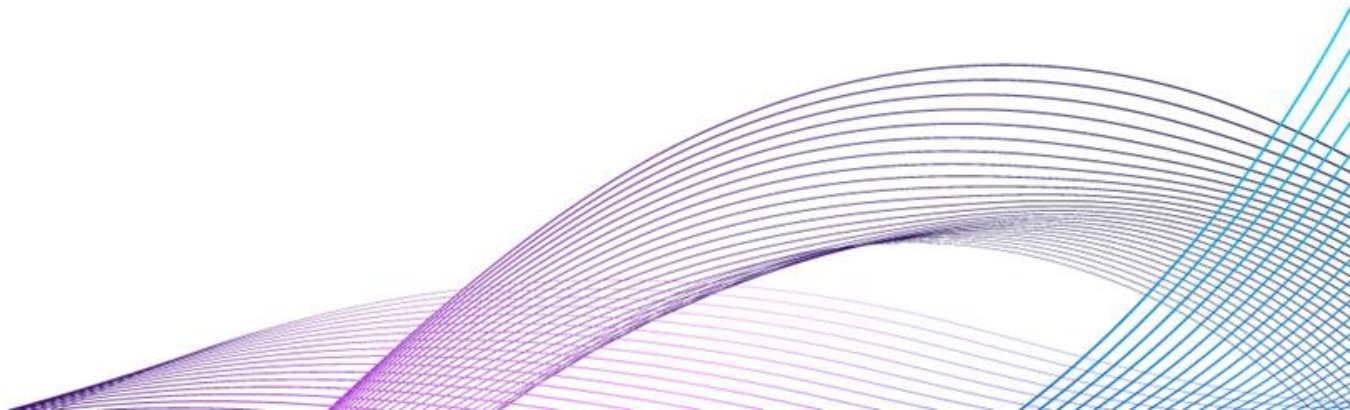




## 01 고맙적 도구들

오픈소스가 인터넷 위에 얹혀진 이후 모두가 겪던 문제라 솔루션도 많죠

- SCM의 보급과 이슈 트래커
- Trac / Redmine의 시대
- 커뮤니케이션 도구들
- 지속적 통합 도구





# 새로운 이슈들

- Git / GitHub everywhere: 여기에도 저기에도...
- Linux 커널 라이선스 변경 실패의 교훈
  - “내 코드 넣지 마!”
  - 모든 사람들에게 동의를 구하지 못하면 라이선스 변경도 어렵다!
- MIT 전성시대
  - 서서히 움직이는 GNU의 시대
  - 쓰든지 말든지... 단 책임은 안 진다!
- Easy come, easy go
  - LLVM부터 Web Framework까지, 무지막지하게 빠른 테크 스택의 변화
  - 마이크로서비스 아키텍처: 조각조각난 소프트웨어
  - IT 기반의 세상: 보안 문제의 대두





# 02

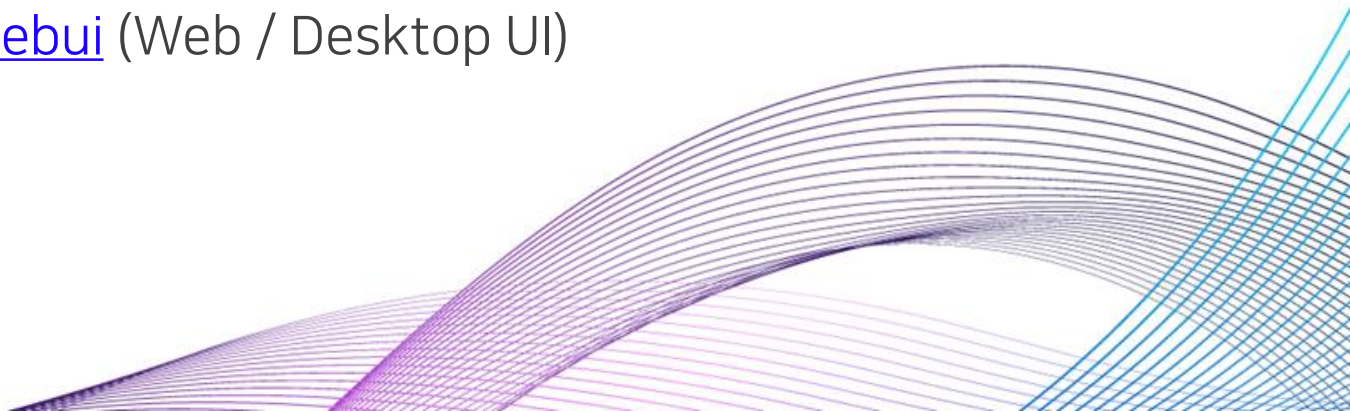
## 오픈소스와 자동화의 예





## 02 사람이 하기 싫으면

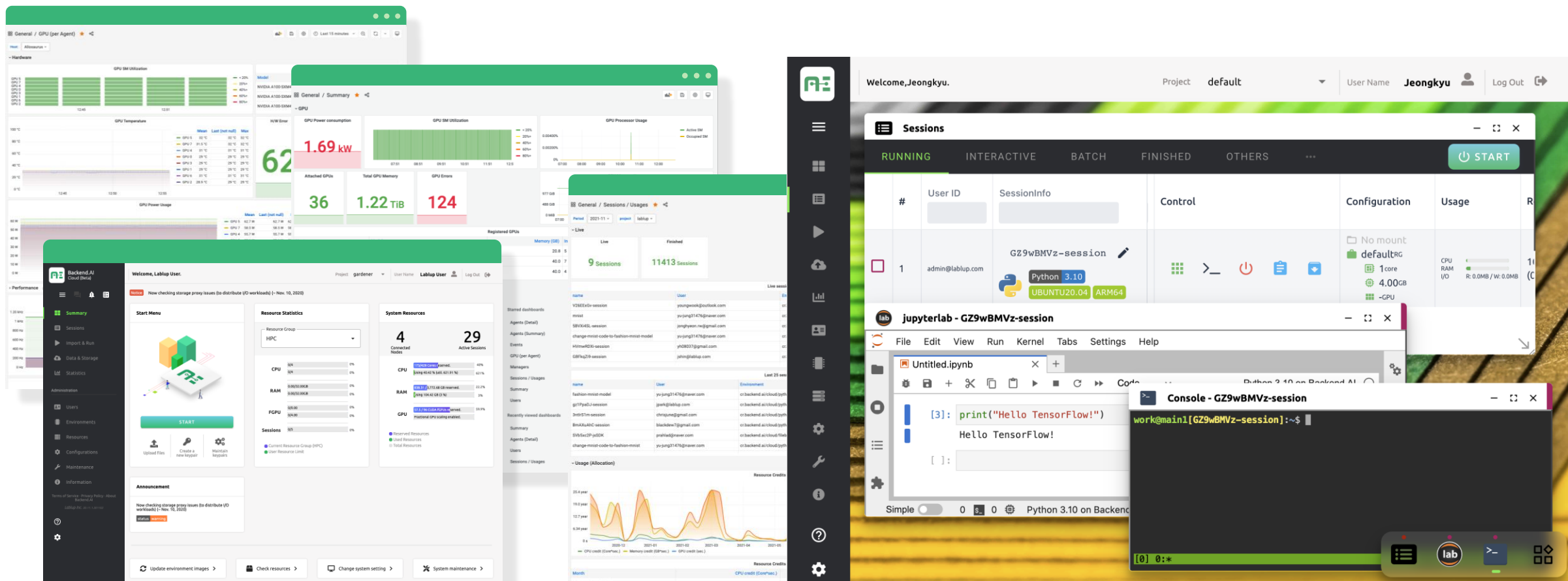
- 컴퓨터를 시킵시다
- CI/CD 솔루션들
  - 코드 리뷰나 배포 과정을 자동화
  - 다양한 방법들
- 오늘 소개할 케이스
  - 다양한 과정을 거쳤던 래블업의 케이스를 소개합니다
  - 모든 적용예는 공개 저장소에서 바로 확인할 수 있습니다.
    - <https://github.com/lablup/backend.ai> (mono-repo)
    - <https://github.com/lablup/backend.ai-webui> (Web / Desktop UI)
    - 및 그 외!





# Backend.AI (<https://www.backend.ai>)

- AI 개발 및 서비스를 위한 올인원 엔터프라이즈 운영 플랫폼
- (여러분들이 흔히 접하는) 온갖 다양한 곳에서 사용되고 있음!





## Backend.AI: 역사

매우 깁니다.

- 2015년 프로토타입 및 클라우드 서비스
- 2016년 베타 테스트 / 오픈소스화
- 2017년 정식 버전 공개 / API 클라우드 서비스 시작
- 2018년 **Fractional GPU 세계 최초 개발** / 분산훈련 기능
- 2019년 엔터프라이즈 버전 발표 / MLOps 파이프라인 배포
- 2020년 로컬 패키지 저장소 / 초고속 스토리지 가속 기능
- 2021년 AI 가속기 추상화 / ARM 아키텍처 지원
- 2022년 커스텀 컨테이너 이미지 빌더 / AI/MLOps 독립 기능





## 왜 오픈소스를?

- 주도 오픈소스 프로젝트
  - 태터툴즈·텍스트큐브 프로젝트 (2006~)
    - 블로그 서비스 소프트웨어
    - 다양한 상용화 서비스 및 플랫폼의 기반 (티스토리 / blogger.com)
    - 글로벌 오픈프론티어 (1기 / 3기)
  - NBA (Network Balancing Act)
    - 패킷라우팅 소프트웨어 (2015~)
    - 공개SW개발자대회 금상 수상 (2015년)
  - Backend.AI
    - 머신러닝 및 코드 분산처리 프레임워크
    - 글로벌 오픈프론티어 (4기 / 5기)

= 익숙하게 하던 거라서...

(팀 자체가 오픈소스 하다 만나 구성된 팀임)

### Python 및 과학연산 라이브러리 시스템 소프트웨어

- |  |   |
|--|---|
| <ul style="list-style-type: none"><li>- Python</li><li>- Numpy</li><li>- TensorFlow</li><li>- asyncio</li><li>- asyncio-redis</li><li>- pyzmq</li><li>- django-money</li><li>- Callosum 🏆</li><li>- aiodocker 🏆</li><li>- aiotoools 🏆</li><li>- aiomonitor-ng 🏆</li><li>- LAPACK</li><li>- Etcetera</li><li>- AirSim</li></ul> | <ul style="list-style-type: none"><li>- Intel DPDK</li><li>- FreeBSD</li><li>- OpenStack</li><li>- OPNFV</li><li>- Libreoffice</li><li>- zeromq</li><li>- zeromq.node</li><li>- NBA 🏆</li><li>- Google Test</li></ul> |
|--|---|

#### contextlib

Add a `contextlib.aclosing()` context manager to safely close async generators and objects representing asynchronously released resources. (Contributed by Joongi Kim and John Belmonte in [bpo-41229](#).)

- [bpo-45416](#): Fix use of `asyncio.Condition` with explicit `asyncio.Lock` objects, which was a regression due to removal of explicit loop arguments. Patch by Joongi Kim.

🔗 Python 3.10 changelog 발췌 (2021년 10월 릴리즈)





## 오픈소스 구조: 개발 방법

- **나선형 개발 모델**

- 요구사항 분석 - 설계 - 구현 - 테스트 반복
- 상대적으로 작고 필수적인 기능부터 시작하여 세부적이고 부차적인 기능 순서로 진행
- 사이클 주기: 4주 / 12주
- 릴리즈 주기: 24주

- **테스트**

- 단위 테스트는 개발과 동시에 작성
- Docker, Codecov 및 GitHub Action을 활용한 독립적이고 자동화된 테스트 수행

- **협업 및 버전관리**

- github.com

- **문서화**

- ReST 마크업 기반 Sphinx 문서화 도구 사용
- 코드와 직접 연결된 상세 문서 작성



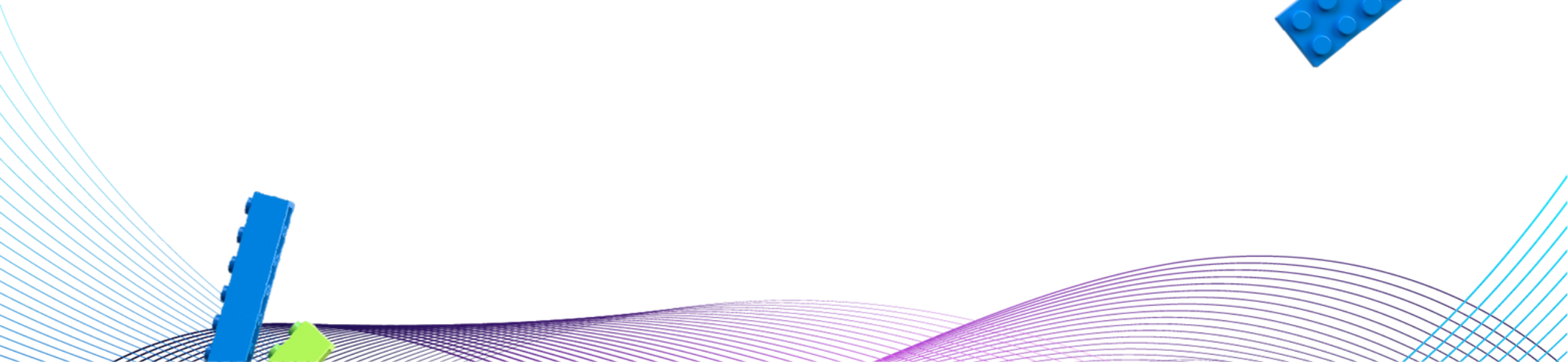
## 오픈소스 구조: 개발 요소

- 폴리글랏 구성
- 백엔드
  - Docker / native Linux
  - Python 3.10 with aggressive type annotations for new codes
  - asyncio + aiohttp, aioredis-py, sqlalchemy + alembic + asyncpg, pyzmq, aiocoder, callosum
  - PostgreSQL 14, Redis 6, etcd v3
- 프론트엔드
  - Javascript ESMModule / React 18 / TypeScript
  - Lit v2 + Material Web Components (MWC) + weightless.dev
  - Electron 14/20 (desktop app)
- **Microservices / Backoffice**
  - Python Django / FastAPI / Flask



# 03

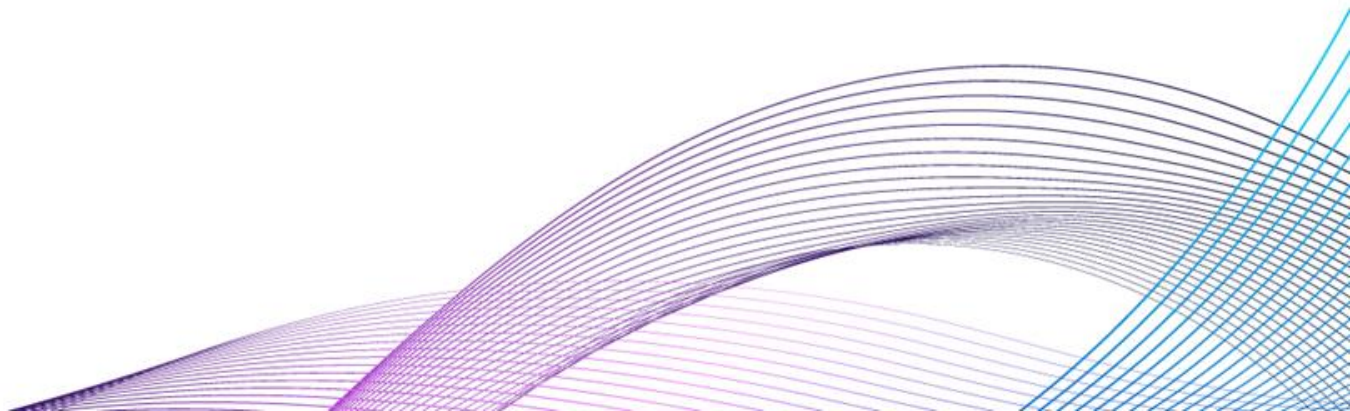
## 실제 자동화 적용하기





## 03 오픈소스 자동화: 왜?

- 자동화
  - “사람에게 맡기면 안되는” 요소들 다수 존재:
  - 엄청나게 많이 해야 하는데 지치는 일들
  - 빼먹기 좋은 일들
  - 그냥... **일이 엄청 많음**





## 03 문제 요약

- **폴리글랏**: 말은 멋지지만 **테크스택이 엄청나게 복잡하다**는 이야기
- **MSA**: 서비스를 구성하는 **패키지 수가 상당하다**는 의미
- **6개월 단위의 릴리즈 일정**: 마이너 릴리즈는 거의 **주당 1회** 가까이 함
- **오픈소스**: 이 모든 과정에 **수많은 사람들**이 참여하는 중

=> 이 **복잡도**를 어떻게 관리할 수 있을까?

Pants기반 모노레포  
런타임 환경 자동화

GitHub CodeSpace  
개발환경 자동생성

GitHub Action  
기여절차/ 테스트 자동화

GitHub Action  
릴리즈 / 패키징 자동화





## CI/CD 의 도전과제

- 자체 개발 CI/CD 운영 (on AWS EC2)
  - 돈을 아끼려고... *사람이 어느 순간 더 비싸짐*
- Travis CI
  - De Facto Standard처럼 쓰이던 CI/CD 도구
  - 필요한 기능이 충분하지 않게 됨
- 필요한 것
  - 다양한 배포 타겟에 따른 빌드 및 테스트
  - 테스트를 위해 전체 클러스터 구성이 필요
    - 그런데 클러스터를 구성하려면 최소 7개의 저장소를 클론해야 하고
    - 변경사항이 그 저장소들의 별도 브랜치에 걸쳐 있음 (MSA / Polyglot!)



## 03 오픈소스 자동화: mono-repo 기반 이전

- Pantsbuild 기반의 모노레포 기반 개발 (2022.7~)
  - 핵심 컴포넌트들의 개발 주기 동일 유지, 강의존성이 존재하는 코드들의 리팩토링을 도움
  - 개발 주기가 크게 차이 나며 버전 호환성 유지가 중요한 일부 컴포넌트는 외부 유지 (WebUI 등)
- 장점
  - 빠른 릴리즈 과정 및 간단한 테스트 파이프라인 구축
  - Python 환경 관리의 통합
  - 커밋 전 전처리 테스트 과정 강제 및 자동화로 최소한의 테스트를 통과한 코드 기여
- 단점
  - 높은 학습 곡선
  - 플러그인 등 Pants 모노레포 밖의 소스코드를 결합하여 테스트 하는 경우 기존보다 추가 고려가 소요됨
  - Pants 내의 별도의 Python 환경이 운영됨



## 03 오픈소스 거버넌스: GitHub 기반 운영

- 거버넌스 구조

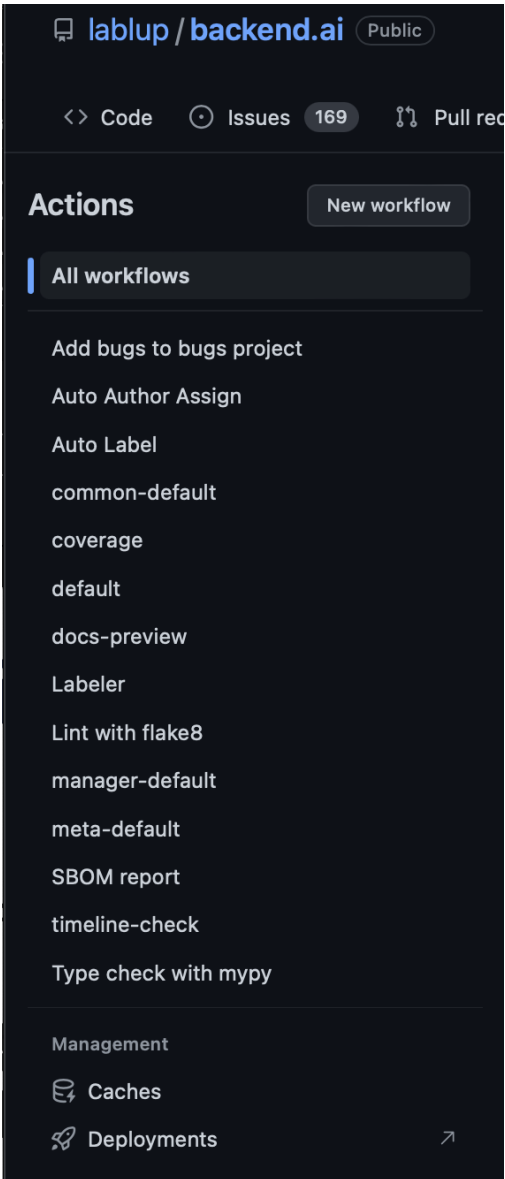
- Backend.AI 프로젝트: 20개 이상의 패키지 및 모듈로 구성
- 프로젝트 별 메인테이너, 개발자, 커미터
  - 메인 프로젝트들: 래블업에서 직접 관리
  - 응용, 예제 및 서드파티 구현체: 각 개발자가 관리
- 로드맵
  - 향후 1년간의 안정 버전 대상 로드맵 공개
  - 제안들은 백로그 및 이슈 형태로 기록 / 이후 약 6개월당 1회의 회의를 통해 로드맵 추가 및 수정

- 프로젝트 구조

- 공개 저장소 및 엔터프라이즈용 비공개 저장소로 구성
- 환경 구성시의 옵션에 따라 설치할 수 있는 범위가 변함



# 오픈소스 자동화: GitHub Action 사용



## • GitHub Action?

- 다양한 동작을 가상환경 상에서 수행하는 워크플로우 도구
- 커밋, PR<sup>[1]</sup>등의 다양한 트리거로 실행
- YAML 템플릿 기반의 유연한 동작 지정
- Marketplace를 통해 여러 사용자들이 만든 다양한 Action 사용 가능

## • 왼쪽의 예 (<https://github.com/lablup/backend.ai/tree/main/.github/workflows>)

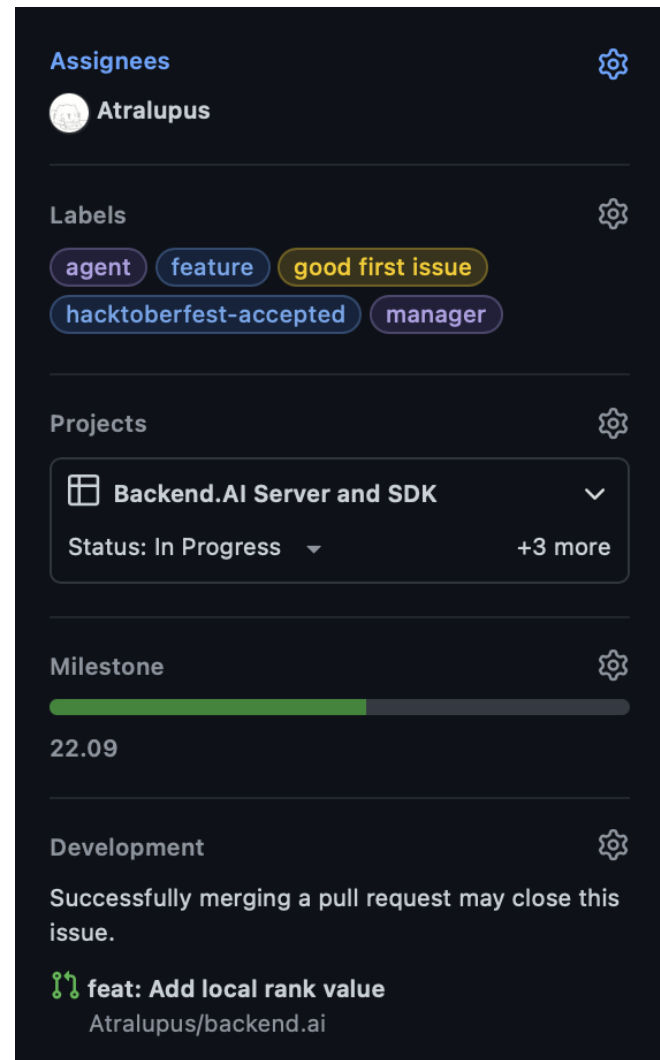
- 이슈가 발행되면 자동으로 개발자 지정
- 이슈가 올라오면 자동으로 적절한 레이블을 붙임
- PR이 올라오면 자동으로 연동된 이슈의 레이블과 같은 레이블을 붙임
- 코드 변경이 발생하면 커버리지 계산을 자동으로 수행
- 코드 변경이 발생하면 구문 규칙을 적용하고, 잘못된 경우 머지를 막음
- 코드 변경이 발생하는 경우 변경내역 조각<sup>[2]</sup>이 존재하는지 확인하고, 없을 경우 머지를 막음
- 포함된 라이브러리가 변경된 경우 자동으로 SBOM<sup>[3]</sup> 생성

[1] Pull Request [2] News Fragment [3] SBOM: Software Bill of Material

# 03

## 오픈소스 자동화: GitHub 기반 기여 절차 간소화

- 기여자 동의 미리 받기
  - 자동 CLA<sup>[1]</sup> (기여자 동의 사항) 사인
  - 이후 발생할 수 있는 다양한 일들에 대해 미리 동의를 받아 두는 절차
- 자동 이슈 할당 및 레이블링
  - 담당자 자동 지정
  - 이슈의 분야에 따른 레이블 자동 지정
  - 만약 연관된 수정 및 개선사항 PR을 받는 경우, 해당 PR에도 동일한 레이블 부여



[1] Contributor License Agreement



# 03

## 오픈소스 자동화: PantsBuild 기반 테스트 자동화

- 로컬 테스트
  - 코딩 컨벤션 확인 (Flake8 기반의 린팅)
  - 문법상 오류 확인 (mypy 기반 타입체킹)
  - BUILD 등의 설정 오류 확인
- 통과를 못하면~ (로컬에서도) 커밋이 안돼요~

```
inureyes@Cube ~/Development/backend.ai/branches/main/bai-main
Performing lint for changed files ...
21:37:16.69 [INFO] Initializing scheduler...
21:37:16.81 [INFO] Scheduler initialized.
Performing lint for changed files ...
On branch docs/installation-guide-updates
nothing to commit, working tree clean
inureyes@Cube ~/Development/backend.ai/branches/main/bai-main
```



# 오픈소스 자동화: GitHub 기반 테스트 자동화

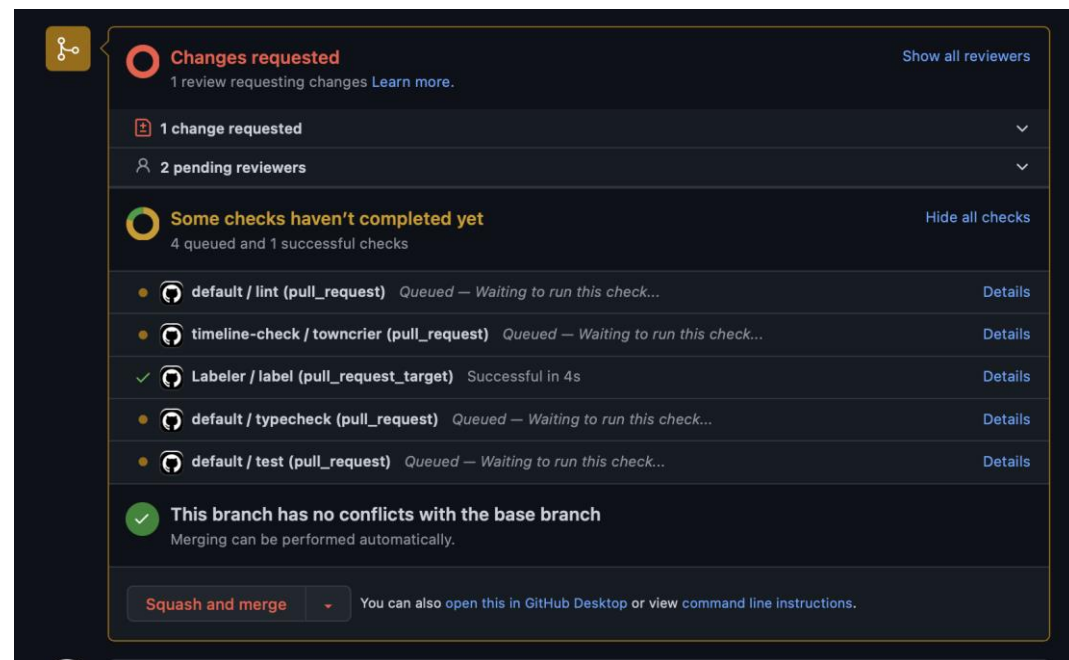
## 코드 검수절차

- 드라이 테스트

- 코딩 컨벤션 확인 (Flake8 기반의 린팅)
- 문법상 오류 확인 (mypy 기반 타입체킹)
- 정적 코드 분석 도구 실행 (CodeQL)

- 동작 테스트

- 유닛 테스트 / 기능 테스트 수행
- 코드 커버리지 자동 계산
- 새로운 기능: 테스트 슈이트를 함께 제출
- 버그 수정: 기존 테스트 슈이트 실행



<https://github.com/lablup/backend.ai/pull/826>



# 오픈소스 자동화: GitHub 기반 배포

<https://github.com/lablup/backend.ai/tree/main/changes>

## 릴리즈 과정 자동화

- **릴리즈 로그 적기:** 가장 힘들(그리고 귀찮음)
- News 작성 규칙 / 체인지로그 작성 규칙
- 모든 PR에 News Fragment 요구
- News Fragment
  - 코드의 변경 사항에 대한 간단한 요약
- 릴리즈 로그 / 체인지로그 생성
  - 릴리즈 시점에 News Fragment를 이용한 체인지로그 자동 생성
- SBOM 생성
  - 릴리즈에 포함되는 다른 패키지의 리스트!
  - 보안 역추적을 위해 필수적으로 요구되고 있음

## Writing changelogs

### Most microscopic: Commit messages

When writing commit messages, we use [the conventional commit style](#) and describe "what I did". Note that we not only use the 'meaning of action' prefixes such as `fix:`, `feat:` but also the 'area of change' prefixes such as `ci:`, `repo:`, `setup:`, etc. Optionally you may include a parenthesized mention on the primarily changed component like `fix(install-dev):`, `refactor(client.cli):`, etc. When mentioning component names, use the Python module name or the file name without its extension.

A commit message may contain multiple lines of detailed description on individual code changes and any related background contexts in either list items or just a bunch of plain text paragraphs. When squash-merging a PR, maintainers should revise the commit message auto-generated as a mere concatenation of all commit messages in the way that it highlights significant changes and technical details without noise.

The target audience is the code reviewers and future developers who dig in the history and background of code lines.

Please refer the commit histories of several PRs as standard examples:

- [lablup/backend.ai#501](#)
- [lablup/backend.ai#480](#)

Please refer the commit message of squash-merged PRs as standard examples:

- [76f933ac2a3a \(#605\)](#)

### In the middle: News fragments

The changelog (aka news fragment) is different. It is automatically included in the release notes via [towncrier](#) and include a collective summary of an entire pull request (i.e., "sum" and "summary" of the commit messages). It should focus on "what the problem this PR resolves" and "how it affects the target audience".

The target audience is the users and/or fellow developers.

A news fragment should be a single-line single English sentence, either as an imperative (aka "commanding") form with no subjective (e.g., "Fix XXX bug", "Support YYY feature") or a complete sentence with a full subjective, verb, and objectives (e.g., "Now it does ZZZ"). There are a few exceptions historically, but it is always recommended to keep multi-line details in the commit message of the squash-merge commit and/or in the pull request body.

Please refer the existing release notes as standard examples.

Note: maintainers may modify the news fragment without notice before merge.

### In the middle: Pull request titles

The title of a pull request is a *conventional commit stylization of the news fragment*, usually as a shorter and more brief version, because

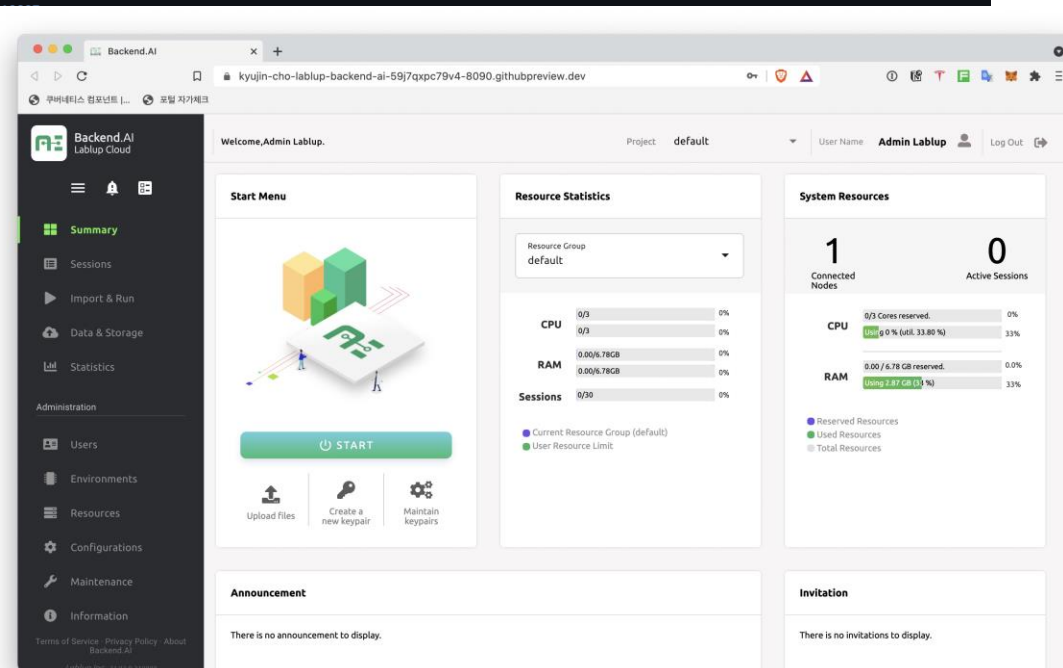
# 03

## 오픈소스 자동화: GitHub Codespace

### • GitHub CodeSpace

- 재현 가능한 / 바로 준비되는 환경
- MS Azure 상의 VM 기반
- Devcontainer 스펙 및 포스트 훅을 이용한 자동 환경 생성
- 개발본 전체 설치 다운로드 및 설정 자동화 만들기
  - .devcontainer/devcontainer.json 형식으로 추가
  - 기존 설정은 길었으나, 7월부터 Mono-repo로 통합한 이후 엄청나게 간결해짐
- 하나씩 전부 설치할 필요 없이 한 번에 개발 환경 셋업 + VSCode 연결

```
1 // For format details, see https://aka.ms/devcontainer.json. For config options, see the README at:
2 // https://github.com/microsoft/vscode-dev-containers/tree/v0.187.0/containers/ubuntu
3 {
4   "name": "Ubuntu",
5   "onCreateCommand": "cd /workspaces/backend.ai && bash /workspaces/backend.ai/scripts/install-dev.sh --codespaces-on-create",
6   "postCreateCommand": "cd /workspaces/backend.ai && bash /workspaces/backend.ai/scripts/install-dev.sh --codespaces-post-create",
7   "forwardPorts": [
8     6022, // storage-proxy
9     8090, // webserver
10    8091, // manager API
11    // wsproxy ports
12    10200,
13    10201,
14    10202,
15    10203,
16    10204,
17    10205,
18    10206,
```

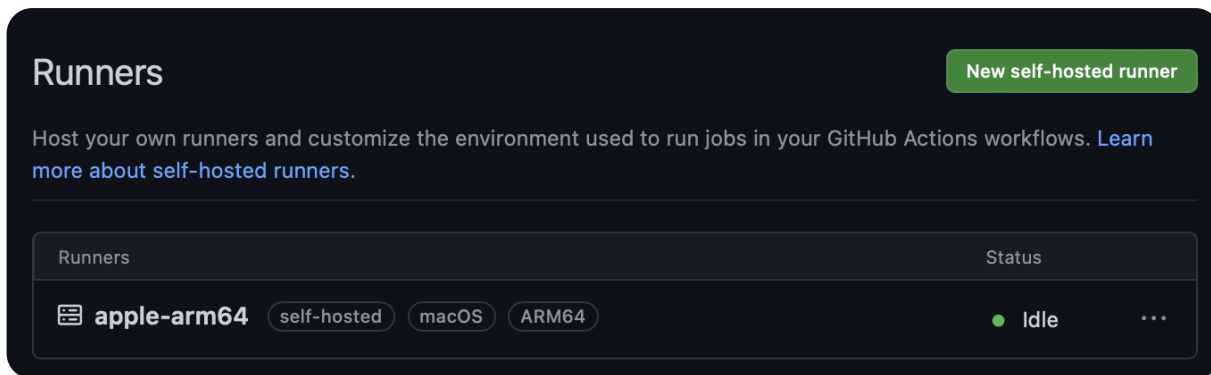


<https://github.com/lablup/backend.ai/blob/main/.devcontainer/devcontainer.json>

# 03

## 오픈소스 거버넌스: GitHub 기반 패키징

- 패키지 테스트, 호환성 테스트 및 빌드 자동화
  - 일반적인 CI/CD 도구들로는 통합 테스트가 불가능한 경우들이 존재
- Local runner
  - 내 PC를 GitHub에 붙여 사용: GitHub에서 제공하지 않는 테스트 및 빌드 환경 구성
- 패키징 자동화
  - 패키지 사이닝 및 PyPi 업로드
    - 하나의 Mono-repo에서 6개의 패키지를 생성함
    - 각 패키지마다 Python 버전별, 운영체제별 버전을 만들고 등록
  - 데스크탑 앱 빌드, 사이닝 및 업로드
    - macOS 패키지는 macOS 상에서 사이닝을 필수적으로 해야 함







# 04

## 마치며





## 04 오늘 우리는

- 삽질을 줄여보려는 장대한 삽질을 보았습니다.

Pants기반 모노레포  
런타임 환경 자동화

GitHub CodeSpace  
개발환경 자동생성

GitHub Action  
기여절차/ 테스트 자동화

GitHub Action  
릴리즈 / 패키징 자동화

- 엄청난 투자와 테스트가 들지만 길게 보면 남는 일에 해당됩니다.

### • 삽질러기여자

- Joongi Kim (Mono-repo / Action) / Sion Kang (Action)
- Kyujin Cho (CodeSpace / WebUI Packaging) / Jeongkyu Shin (WebUI CodeQL/ SBOM)



## 오픈소스의 접근성과 자동화 이야기

- 오픈소스를 시작하기는 쉽습니다
  - 오픈소스를 유지하기는 어렵습니다
  - 오픈소스를 함께 하기는 더 어렵습니다
- 
- 자동화의 의미
    - 계속 관심사가 바뀌는 테크 세상에서,
    - 사람이 할 일만 사람에게 남겨서
    - 조금이라도 더 힘을 모아보는 방법

# 감사합니다

자동화와 함께 하는 오픈소스

