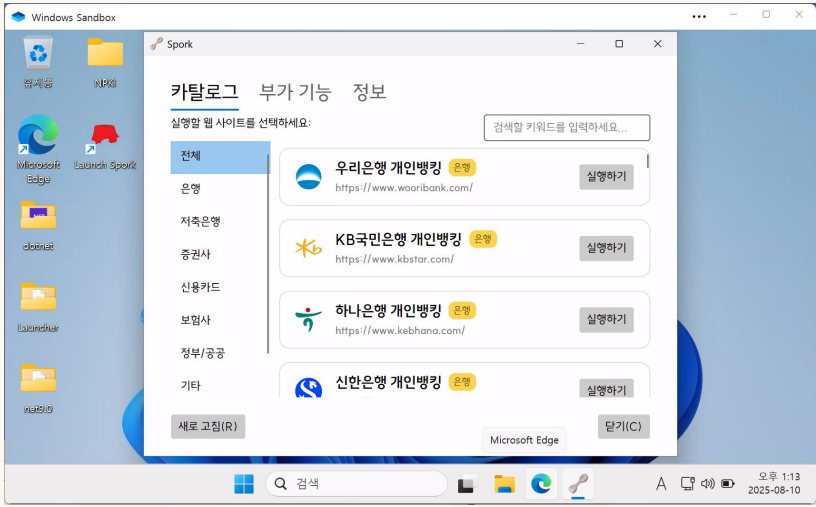
식탁보 프로젝트의 세계관 확장 – LXD 기반 샌드박스 구현

닷넷데브 / Microsoft MVP (since 2009)
남정현

Contents



식탁보 프로젝트 소개 및 근황

식탁보 프로젝트란?

Keeping Your Computer Sound and Safe!

추가 Windows OS 라이선스 구매 없이

Windows Sandbox로 1회용 가상 PC를 생성

인터넷 뱅킹, 공공기관 접속 시 필요한 보안 플러그인, 공동인증서 복사를 자동화

컴퓨터에 설치되는 보안 프로그램으로 인한 기능 충돌과 불편을 최소화

https://yourtablecloth.app/

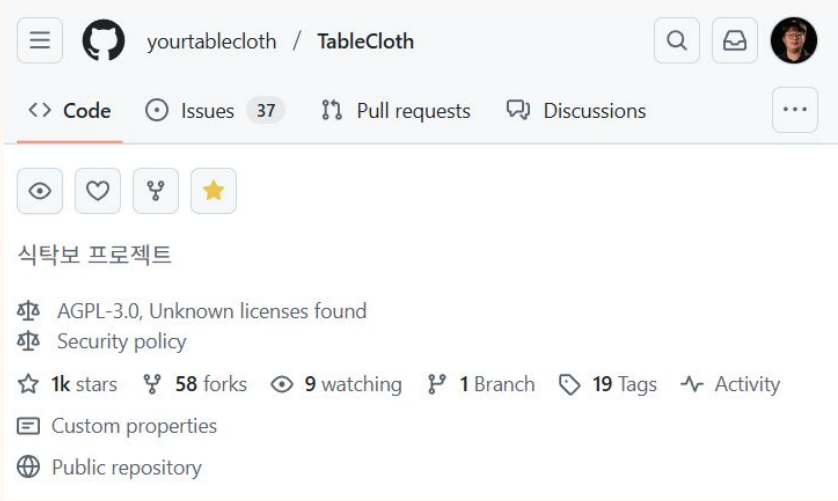
식탁보 프로젝트 근황

2025년 7월 25일 현재 Star 1k 달성 🤦

Avalonia + Native AOT 기반으로 식탁보 3.0 개발 중

Blazor WASM과 OpenRouter를 활용한 식탁보 라이트 개발 중

LXD 기반의 리눅스용 식탁보 개발 중



상용화 단계의 현실적 고민과 대안 찾기

식탁보에 대한 아쉬움

아쉽게도 식탁보는 현재 다음의 한계가 있습니다.

- Windows 11 Pro, Education, Enterprise 이상의 SKU가 필수
- Qualcomm, Apple Silicon 환경에서 제한 또는 지원되지 않음
- Linux, macOS, 태블릿 PC 환경에서 지원되지 않거나 사용이 불편함

그래서 Windows Sandbox를 클라우드로 만든다면 어떨까 하는 고민을 했습니다.

식탁보 클라우드의 기술적 검토

Windows Sandbox 클라우드 전환

- 식탁보 클라우드 상용화의 핵심 기술
- 기술 개발은 완료된 상태입니다.

그러나 사업화 장벽이 문제입니다.

- Windows 원격 세션 송출 == RDS 라이선스 과금 대상
- 고객 당 사용료 급증으로 경제성 악화
- 이로 인해 너무 비싼 상용 서비스가 되는 문제가 있음

현재 상황

- 기술적 성공 vs 사업적 한계
- 라이선스 비용이 상용화 걸림돌

대안: 리눅스 기반의 VDI와 보안 플러그인

오픈뱅킹과 리눅스 보안 플러그인

- Non-ActiveX 정부 정책 준수의 연장선
- 리눅스(amd64) 플러그인 제공 확대

개발자 vs 프로덕트 오너의 딜레마 – 양가감정

- 생태계의 관점에서는 갈라파고스화 진화 우려
- 프로덕트 오너로서는 비즈니스 기회 포착의 기쁨

현재의 선택

- 복잡한 감정 속에서도 명확한 입장
- 식탁보 프로덕트 오너 역할에 집중

보안 플러그인의 문제점

보안 플러그인은 Windows든 Linux든 큰 차이는 없습니다.

- Windows 버전은 많이 사용되고 있지만. 이미 많은 문제와 불편 야기
- Linux 버전은 Windows에 비해 늦은 지원이 늦은 탓에 더 위험하고 불편한 설치 과정

해결 방안?

- 컨테이너 기술 활용 필요성이 그래서 대두됩니다.

Docker vs. LXD 컨테이너 **vs.** 가상 머신

세 가지 기술의 미묘하지만 분명한 차이점

- **Docker (containerd)**
  애플리케이션 레벨 격리, 호스트 커널 공유로 가볍고 빠른 실행

- **LXD**
  시스템 레벨 컨테이너, VM과 유사한 완전한 리눅스 환경을 컨테이너로 제공.
  커널 기능 일부 지원.

- **Virtual Machine (VirtualBox, VMware, ...)**
  하드웨어 레벨 격리, 독립적인 OS 실행으로 완전한 격리와 높은 보안성

ContainerD 기반의 환경이 참 매력적이긴 하지만...

ContainerD 환경의 한계

- Docker 기반 = 업계 사실상 표준
- 커널 모듈 적재 불가능
- 보안 플러그인의 시스템 컨테이너 수준 기능 필요

기존 대안의 문제

- VM 배포 = 경량성, 재구성/재현성 목적에 부적합
- 오버헤드 과다

해결책: LXD 선택

- 시스템 컨테이너 수준 기능 지원
- 경량 환경 + 재구성/재현 가능성 확보

예: ASTx - LXD가 반드시 필요한 환경

ASTx 구동 요구사항

- `nf_conntrack`, `nf_conntrack_netlink` 모듈 필수
- `nf_conntrack_ipv4` 모듈 탐지
- `nf_conntrack` 으로 통합되어 지원이 중단되었지만 억지로 맞춰야 함

문제점

- 이런 패치를 일상 시스템에 직접 적용하기에는 부적절
- 하지만 인터넷뱅킹, 공공서비스 접속 시 반드시 적용해야만 함

해결책: LXD 활용

- 격리된 컨테이너 환경에서 안전하게 구동
- 호스트 시스템 영향 최소화

Windows에서 cloud-init Inner Loop 만들기

WSL 2와 LXD 사이의 공통점 – cloud-init

저는...

- 23년차 Windows 기술 스택 기반 개발자입니다.
- Windows 환경에서 가능한 많은 일을 해결하고 싶습니다.

그렇지만...

- LXD 개발을 위해 리눅스를 사용해야 합니다.
- 익숙한 개발 환경과 도구를 최대한 활용하고 싶습니다.

해결 방안

- 다행히 WSL 2와 LXD 모두 cloud-init 지원
- 그래서 공통 수단을 통한 환경 통합이 가능합니다.

LXD 자동화를 위한 cloud-init

cloud-init의 역할

- 퍼블릭 클라우드(Azure, AWS, GCP)의 VM 초기 설정 자동화
- 마스터 이미지 기반 통일된 설정 수단

LXD 환경에서의 활용

- LXD의 cloud-init 지원으로 쉬운 활용 가능
- 개발 환경 자유로운 선택/설정 가능

식탁보 리눅스 버전에서의 **cloud-init** 채택 과정

Root FS로 리눅스 배포판을 만드는 시점에서 `cloud-init` 파일을 활용

Ubuntu Linux VM을 만든 후, LXD를 snap에서 설치/초기화하여 컨테이너를 만들 때 `cloud-init` 파일 지정

VM에 LXD를 설정한 후 필요한 만큼, 그리고 수용 가능한 만큼 LXC 컨테이너를 만들어 서비스합니다.

cloud-init 구조 살펴보기

cloud-init 자체는 YAML 형식의 선언형 스크립트입니다.

스크립트 시작은 #cloud-config 문구를 넣어 시작합니다. (이 부분만 YAML 스펙에서 갈라지는 예외입니다.)

이후에는 일반적인 YAML 파일과 동일한 문법을 사용하여 스키마에 맞추어 작성합니다.

기본 사용자 이름, 소속 될 그룹, sudo 권한, 기본 셸 등을 users: 섹션에 대입합니다.

```
#cloud-config
users:
- name: tablecloth
  gecos: TableCloth
  groups:
  - adm
  - sudo
  - audio
  - video
  - netdev
  sudo: ALL=(ALL) NOPASSWD:ALL
  shell: /bin/bash
…
```

cloud-init 구조 살펴보기

cloud-init 자체는 YAML 형식의 선언형 스크립트입니다.

write_files: 섹션에는 커널 모듈, 서비스 등 필요한 Descriptor 파일이나 설정 파일을 넣을 수 있습니다.

packages: 섹션에는 설치할 패키지들의 목록을 기재합니다.

locale: 섹션에는 시스템이 사용할 기본 언어, 지역, 인코딩 설정을 지정합니다.

runcmd: 섹션에는 root 계정으로 시스템 초기화 즉시 실행할 명령어들을 지정합니다.

```
write_files:
- path:
/etc/modprobe.d/nf_conntrack_ipv4.conf
  append: false
  content: |
    alias nf_conntrack_ipv4 nf_conntrack
packages:
- locales
…
locale: ko_KR.UTF-8
runcmd:
- modprobe nf_conntrack
- modprobe nf_conntrack_netlink
- depmod -a
- modprobe nf_conntrack_ipv4
```

WSL 2에서 `cloud-init` 사용하기

앞서 이야기한대로 저는 주로 Windows에서 개발합니다.

물론 Hyper-V를 통해 Ubuntu Server를 설치해도 되지만, 노트북 배터리 타임을 생각하면 좀 더 경량화된 환경을 선호합니다.

WSL의 경우 다음의 조건을 충족하면 사용 가능합니다.

- `systemd` 와 `cloud-init` 지원을 포함한 배포판 필요 (Ubuntu 추천)
- `systemd` 초기화를 지원하는 WSL 2 필요
- Windows와 WSL 간 상호 운용 기능 활성화 필요
- WSL 2로 인하여 중첩 가상화가 반드시 필요

WSL 2에서 `cloud-init` 사용하기 (계속)

사용하는 방법

- `%USERPROFILE%\.cloud-init\<InstanceName>.user-data` 파일 경로에 `cloud-init` 선언 파일을 넣은 후
- `<InstanceName>` 에 해당하는 WSL 배포판을 설치 - 또는 - 다른 RootFS에서 가져오기
- `cloud-init status --wait` 명령으로 실행 완료될 때까지 대기
- 실행 결과를 확인하고, `wsl --unregister` 명령으로 배포판을 지우면 쉽게 초기화할 수 있습니다.

WSL 2에서 cloud-init 사용하기 (계속)

```
> curl.exe -L
https://cloud-images.ubuntu.com/wsl/releases/2
4.04/current/ubuntu-noble-wsl-arm64-wsl.rootfs
.tar.gz -o
ubuntu-nobel-wsl-arm64-rootfs.tar.gz
> mkdir %USERPROFILE%\.cloud-init
> notepad
%USERPROFILE%\.cloud-init\MyDistro.user-data
> mkdir C:\Distro
> wsl --import MyDistro C:\Distro\MyDistro
.\ubuntu-nobel-wsl-arm64-rootfs.tar.gz
--version 2
> wsl -d cloud-init status --wait
> wsl -d cat /var/tmp/hello-world.txt
> wsl --unregister MyDistro
```

```
#cloud-config
write_files:
- content: |
    Hello from cloud-init
  path: /var/tmp/hello-world.txt
  permissions: '0777'
```

WSL 2에서 확인이 끝났다면?

실제 리눅스 배포판에 LXC를 바로 설치해서 확인해 볼 수도 있겠지만, 저는 Hyper-V에서 한 번 더 확인해보는 것을 선호합니다.

Intel/AMD CPU 사용자는 Hyper-V를 포함하여 여러 가상화 소프트웨어를 쓸 수 있으니 선호하시는 것을 택하시면 됩니다.

저는 Qualcomm CPU 사용자여서 사실상 Hyper-V로 선택지가 제한됩니다.

만약 Hyper-V를 사용하기 원하신다면 알아두시면 좋은 팁 몇 가지를 언급하려 합니다.

Hyper-V 환경에 Ubuntu Server 구축하기

- **Subiquity – GitHub에서 SSH Public Key 불러오기**
  지금 내 노트북에 있는 SSH Public Key를 직접 붙여넣어도 되지만, GitHub에 등록했던 적이 있는 Public Key를 아이디만 넣으면 바로 불러와 `~/.ssh/authorized_keys` 에 자동으로 추가해줍니다.

- **자동으로 부여되는 DNS 이름 활용하기**
  기본적으로 Hyper-V VM을 만들면 DHCP로 주소 할당이 이루어지므로 호스트 컴퓨터에서 접근이 쉽지 않습니다. IP 주소 대신, `computername.mshome.net` 주소를 사용하면 쉽게 접근 가능합니다.
  예: `ssh userid@hostname.mshome.net`

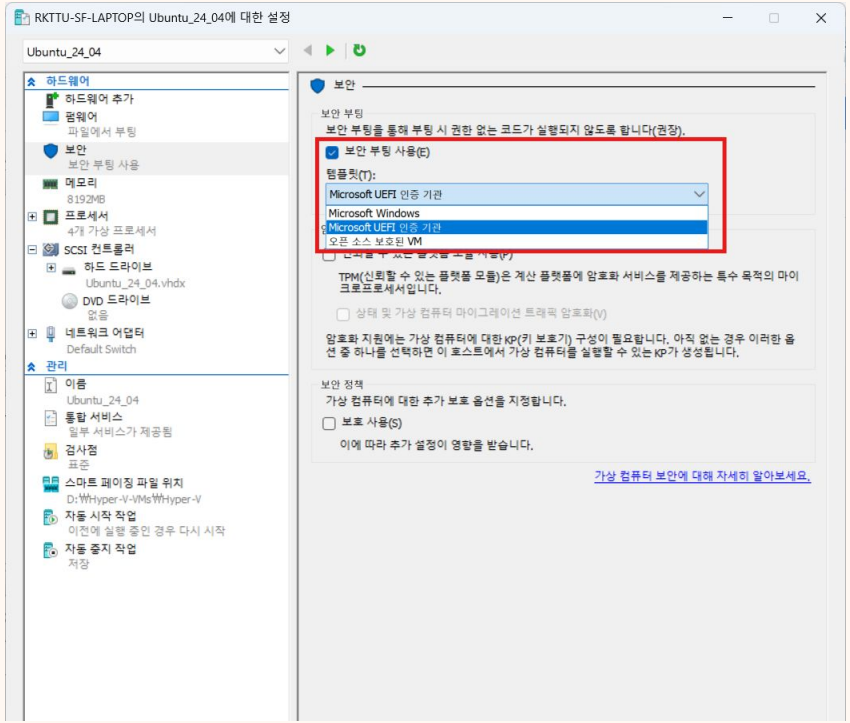
Hyper-V 환경에 Ubuntu Server 구축하기 (계속)

보안부팅 관련 설정

Hyper-V 기본 설정만으로는 리눅스 VM ISO 부팅이 안되는 점이 있어 꼭 체크가 필요한 부분입니다.

Microsoft UEFI 인증 기관을 선택해야 Ubuntu에서 UEFI 부팅이 가능합니다.

그리고 다른 템플릿을 선택했다가 TPM을 켜면, 템플릿 변경이 안되며 이 때는 VM을 새로 만들어야 하므로 TPM은 필요하지 않다면 켜지 않는 것을 추천합니다.



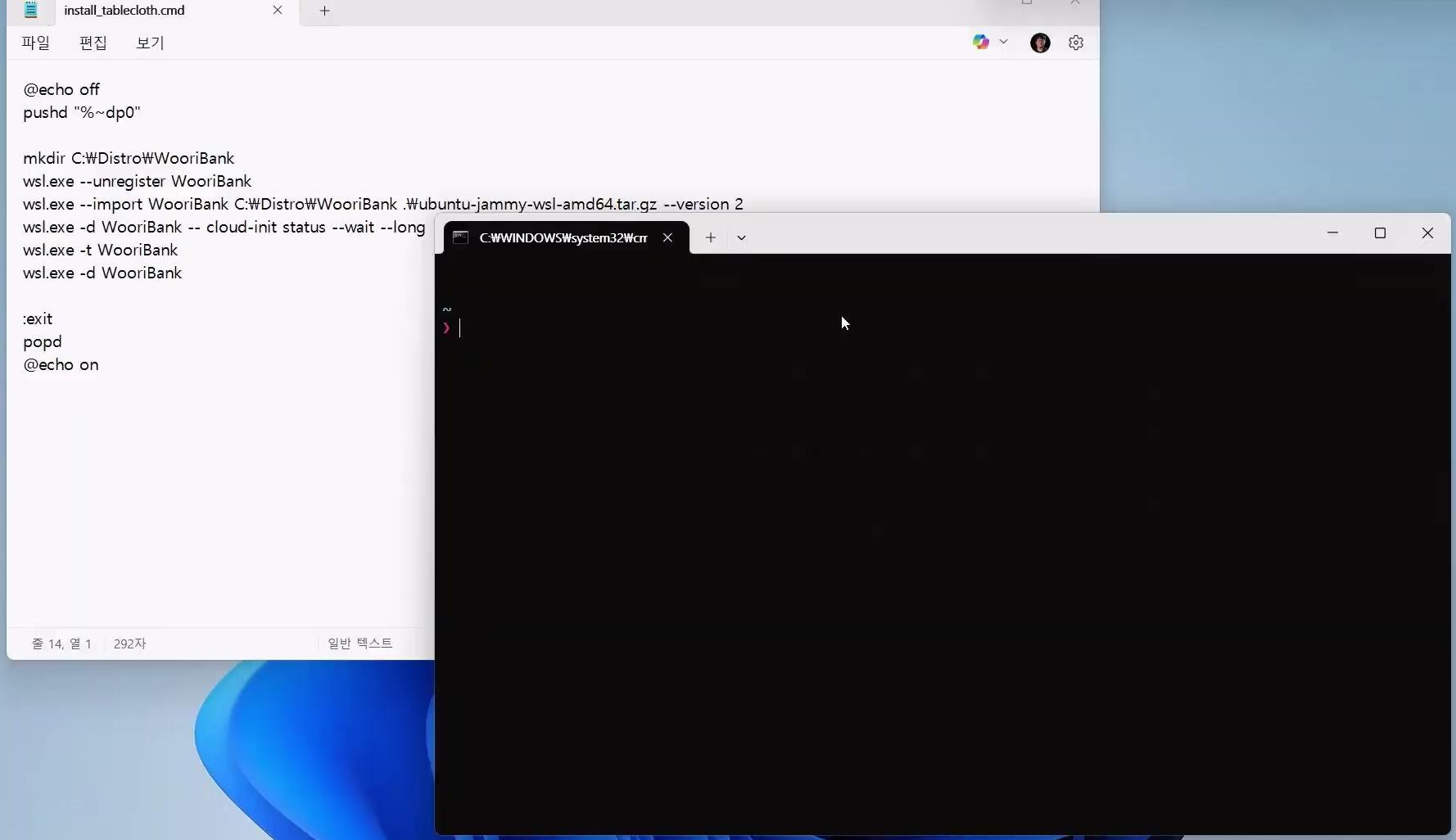
LXC에서 cloud-init 파일 사용하기

보통 cloud-init 파일은 커맨드라인에서 cat 명령으로 읽어서 처리할 수 있을만큼 길지 않고 단순한 편이지만, 제가 만들 파일은 내용이 긴 편입니다.

그래서 아래의 절차대로 LXC 컨테이너를 "만들기"만 하고, user-data를 따로 설정한 다음 "시작"하는 절차로 3단계로 나누어 처리합니다.

```
lxc init ubuntu:24.04 mycontainer
cat my-data | lxc config set mycontainer user.user-data -
lxc start mycontainer
```

Demonstration


install_tablecloth.cmd
파일 편집 보기
@echo off
pushd "%~dp0"
mkdir C:₩Distro₩WooriBank
wsl.exe --unregister WooriBank
wsl.exe --import WooriBank C:₩Distro₩WooriBank .₩ubuntu-jammy-wsl-amd64.tar.gz --version 2
wsl.exe -d WooriBank -- cloud-init status --wait --long
wsl.exe -t WooriBank
wsl.exe -d WooriBank
:exit
popd
@echo on
줄 14, 열 1
292자
일반 텍스트
C:₩WINDOWS₩system32₩cmd

향후 로드맵

몇 가지 진행 중인 프로젝트

TableCloth 3.0

- Avalonia와 Native AOT를 통하여 .NET 런타임, JIT 과정 없이 빠르게 실행되는 네이티브 GUI 프로그램 개발 (리눅스, 맥 지원 고려)
- 사용자 경험 변경: 샌드박스 실행 후 세션 내에서 플러그인 설치 후 접속하는 방식으로 변경 (세션 재 사용성 증대)
- 직접 뱅킹용 VM을 설치해서 사용하는 시나리오 지원 (Spork)
- OpenRouter 기반 AI 컨시어지, MCP 서버 추가 예정

TableCloth Lite

- 웹 브라우저와 Windows Sandbox만 설치되어있으면 바로 쓸 수 있음
- OpenRouter 기반 AI 컨시어지 기능 강화 예정
- 추후 클라우드 버전 식탁보의 프론트엔드 스택으로 쓰일 것을 고려하여 설계

식탁보 리눅스

지금까지의 진행 상황

- Starlark 기반의 cloud-init 템플릿 생성 엔진으로 WSL 2에서 PoC를 진행했습니다.
  - Bazel은 Google이 만든 고성능 빌드 시스템이며, Starlark는 그 빌드를 선언적으로 정의하기 위한 구성 언어입니다.
- 개발 스택을 .NET 10 File-based App으로 전환 중입니다.
  - Native AOT를 사용하여 역시 Standalone BInary로 만들 예정입니다.
  - Windows, Linux 기반 Desktop/Server 애플리케이션을 모두 C#으로 개발 스택을 통일하려 합니다.
- 연말에 Private Preview를 출시하는 것을 목표로 진행 중입니다.

Thank You!

yourtablecloth.app

발표 자료 다시 보기