아이디어 기획안

Arfni

> “기존 인프라의 개념을 완전히 뒤집다”

1. 프로젝트 배경

프로젝트 소개

기존의 어렵고 복잡한 인프라 구축을 GUI 기반으로 쉽고 간편하게 만들어주는 배포 자동화 서비스

프로젝트 주제 선정 이유 및 배경

- 인프라 초기 구축은 도커, 데이터베이스, 메시지 브로커, 로드밸런서 등 구성 요소마다 설정 방식이 달라 복잡하고 재현성이 낮아 많은 시간이 소요된다.
- 환경 파일과 네트워크 설정을 수동으로 맞추다 보니 사람마다 결과가 달라 품질 편차가 발생하고, 로컬·서버·클라우드 전환 시 반복 설정으로 개발 생산성이 떨어진다.
- 이를 해결하기 위해 빠르고 신뢰 가능한 인프라 자동 구성 방식에 대한 요구가 증가하고 있다.

2. 프로젝트 주요 기능

블록처럼 끌어다 놓고 폼만 채우면, 선언 파일을 자동 생성해 **설계→생성→연결→실행→모니터링**까지 끝내주는 오픈소스 인프라 빌더.

1. **드래그&드롭으로 웹·DB·캐시 등 서비스 구성**
    - **설명:**

      React Flow 기반 GUI에서 웹, 백엔드, 프록시, 캐시, 메시지 브로커 등의 서비스를 블록 형태로 끌어다 놓으며 직관적으로 인프라를 설계할 수 있다.
    - **상세 동작:**
        - 노드를 추가/연결(엣지)하여 서비스 간 종속성을 시각적으로 표현

    - 우측 속성 패널에서 포트, 볼륨, 환경변수, 시크릿 등의 세부 설정을 폼으로 입력
    - 입력 즉시 `stack.yaml` 선언 파일로 동기화되어 CLI에서 바로 실행 가능

2. **stack.yaml 선언 파일 생성**
    - **설명:**

      캔버스에서 구성한 결과를 기반으로 표준화된 선언형 파일(`stack.yaml`)을 자동 생성
    - **상세 동작:**
        - 각 노드의 설정(런타임, 포트, 볼륨, 환경변수 등)을 YAML 구조로 변환
        - 서비스 간 참조(`ref`), 링크(`links`), 헬스 체크(`health`) 규칙 포함

3. **Go 기반 CLI 엔진으로 Docker/EC2 환경 자동 배포**
    - **설명:**

      Arfni의 핵심인 Go CLI 엔진이 `stack.yaml`을 해석해 실제 배포를 수행
    - **상세 동작:**
        - 로컬 환경은 **Docker Compose**, 원격 환경은 **SSH(EC2)** 기반으로 동작
        - 각 대상(driver)에 맞게 Compose 파일을 생성 후 `build`, `up` 명령을 실행
        - EC2의 경우 `scp`로 산출물 업로드 후 원격에서 동일 명령 실행

4. **Generate → Build → Up → Health 파이프라인 자동 실행**
    - **설명:**

      한 번의 명령으로 인프라 전체의 생성부터 헬스체크까지 일괄 수행
    - **상세 동작:**
        1. **Generate** — `stack.yaml` 기반으로 Compose/.env/Dockerfile 자동 생성
        2. **Build** — 이미지 빌드 및 캐시 최적화
        3. **Up** — 컨테이너 실행 및 네트워크 연결
        4. **Health** — HTTP/TCP 헬스체크 및 자동 재시도

5. **프리플라이트 체크로 도커 설치·포트 충돌 사전 감지**
    - **설명:**

      배포 실행 전, 환경 문제를 자동 점검해 실패를 예방
    - **상세 동작:**
        - Docker 설치 여부, 포트 충돌, SSH 연결, 권한(0600) 등 사전 검증
        - 필요한 경우 Docker 설치 스크립트 자동 실행(EC2)

6. **시크릿·크리덴셜 OS 키체인에 안전 저장**
    - **설명:**

      비밀번호, PEM 키 등 민감 정보를 안전하게 관리
    - **상세 동작:**
        - OS별 Keychain(`Windows DPAPI`, `macOS Keychain`, `libsecret`) 사용
        - CLI 명령(`arfni secrets set KEY`)으로 암호 저장
        - 실행 시 `.env`에만 주입되어 로그나 이미지에 평문 노출 방지

3. 프로젝트 기술 스택

- FE: React + TypeScript + Vite + Tailwind(+ React Flow), Tauri
- BE/엔진: Go, Rust
- 어댑터: **Compose(MVP)** → Helm/K8s, Terraform(클라우드) 단계 확장
- 저장: SQLite(상태/이력) → 필요 시 Postgres
- 보안: 시크릿 암호화(age/OS 키체인), OIDC/IAM Role 우선, 감사 로그
- 관측: 로그 스트림, 헬스체크, (선택) Prometeus/Grafana/ Node Exporter

4. 프로젝트 기대 효과

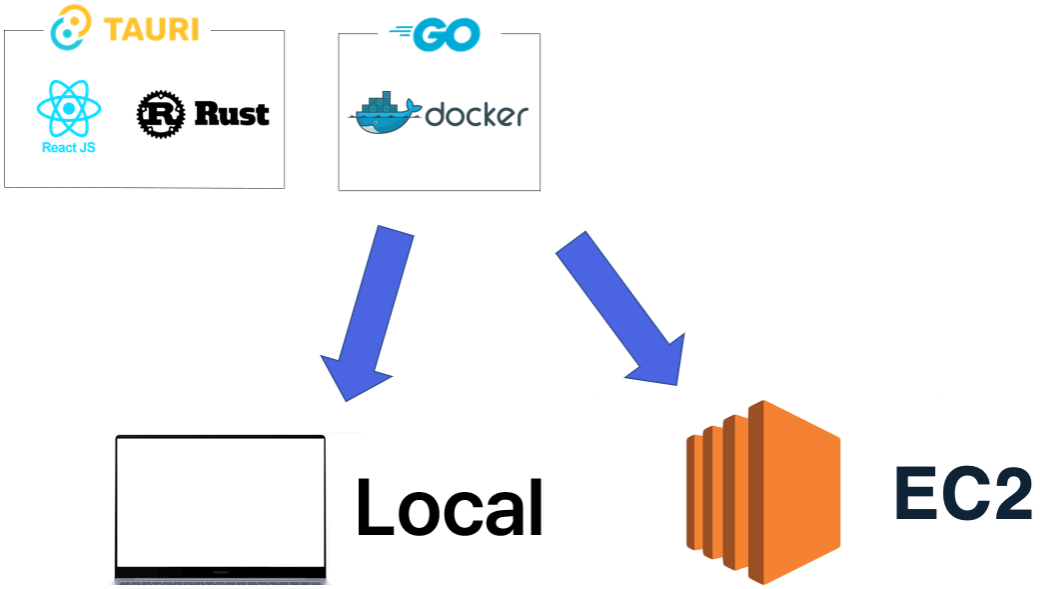
- **생산성 향상**
    - 수동 설정 없이 시각적으로 설계하므로 **인프라 초기 구축 시간 단축**

    - 클릭 몇 번으로 **PoC·데모 환경 즉시 배포 가능**
- **표준화 및 재현성 확보**
    - 모든 구성은 선언형(`stack.yaml`)으로 관리되어 **팀 간 환경 불일치 해소**
    - 버전관리(Git)와 결합 시 **감사·리뷰·복구 용이**
- **운영 안정성 강화**
    - 단계별 Health Check 및 Rollback으로 복구 시간 감소
    - 시크릿 자동 암호화로 보안 사고 위험 최소화
- **시장 및 생태계 확장성**
    - 오픈소스 기반으로 DB/클라우드/미들웨어 어댑터를 커뮤니티가 확장 가능
    - 벤더 종속 최소화, 누구나 커스터마이즈 가능한 **오픈 생태계**

5. 프로젝트 계획 및 일정

버전	목표	기능
v0.1	서버 배포 및 모니터링 기능 구현	1차 MVP로 GUI를 구현하여 원격 서버에 배포 및 모니터링 기능 추가
v0.2	Plan/Diff/Drift 지원	변경 전후 비교, 상태 추적
v0.3	추가 설명 및 서버 비용 계산	LLM을 사용하여 각 배포에 관해 추가 설명 제공 및 예상 인원과 서버를 바탕으로 예쌍 비용 계싼
v0.4	모바일 버전 확장 or 플러그인 확장	로그인 기능을 추가하여 모바일에서도 사용 가능하도록 버전 개발 or 배포 환경을 위한 플러그인 확장

6. 아키텍처


TAURI
React JS
Rust
GO
docker
Local
EC2