Mini Replit Clone — Docker + Nix + FastAPI (Python Version)

🎯 Goal

Build a Replit-style system using:

- FastAPI (control API)
- Docker (isolation)
- Nix (reproducible Python environment)
- Dynamic dependency installation via JSON request
- Each project runs inside its own container

🏗️ High-Level Architecture

User → FastAPI Control Server → Workspace Folder → Docker Container → Nix → pip insta
main.py

📁 Folder Structure

mini-replit-python/ │ ├── api/ │ └── main.py │ ├── docker/ │ ├── Dockerfile │
flake.nix │ └── workspaces/

1️⃣ Nix Environment (Python Runtime)

File: docker/flake.nix

```
{
  description = "Python Dev Environment";

  inputs = {
    nixpkgs.url = "github:NixOS/nixpkgs/nixos-23.11";
  };

  outputs = { self, nixpkgs }:
    let
      system = "x86_64-linux";
      pkgs = import nixpkgs { inherit system; };
    in
    {
      devShells.${system}.default = pkgs.mkShell {
        buildInputs = [
          pkgs.python311
          pkgs.python311Packages.pip
        ];
      };
    };
}
```

This guarantees:

- Python 3.11
- pip
- Reproducible runtime

2️⃣ Dockerfile

File: docker/Dockerfile

```
FROM nixos/nix

WORKDIR /workspace

RUN mkdir -p /etc/nix &&
    echo "experimental-features = nix-command flakes" >> /etc/nix/nix.conf

COPY flake.nix /workspace/

RUN nix develop --command true

EXPOSE 8000

CMD ["nix", "develop"]
```

Build image:

```
docker build -t mini-replit-python ./docker
```

3️⃣ FastAPI Control Server

File: api/main.py

```python
import os
import uuid
import json
import subprocess
from pathlib import Path
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel

app = FastAPI()

WORKSPACE_DIR = Path(__file__).resolve().parent.parent / "workspaces"
WORKSPACE_DIR.mkdir(exist_ok=True)

# Request model
class ProjectCreateRequest(BaseModel):
    dependencies: dict = {}

# CREATE PROJECT
@app.post("/create")
def create_project(request: ProjectCreateRequest):
    project_id = str(uuid.uuid4())
    project_path = WORKSPACE_DIR / project_id
    project_path.mkdir(parents=True, exist_ok=True)

    # Create requirements.txt
    requirements = "\n".join([
        f"{pkg}=={version.strip('^')}" if isinstance(version, str) else pkg
        for pkg, version in request.dependencies.items()
    ])

    (project_path / "requirements.txt").write_text(requirements)

    # Create starter FastAPI app
    main_file = f"""
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def root():
    return {{"message": "Project {project_id} running"}}

"""

    (project_path / "main.py").write_text(main_file)

    return {"project_id": project_id}


# RUN PROJECT
```

```
@app.post("/run/{project_id}")
def run_project(project_id: str):
    project_path = WORKSPACE_DIR / project_id

    if not project_path.exists():
        raise HTTPException(status_code=404, detail="Project not found")

    container_name = f"repl-{project_id}"

    command = f"""
    docker run -d
      --name {container_name}
      -v {project_path}:/workspace
      -p 0:8000
      --memory=256m
      --cpus=0.5
      --pids-limit=100
      mini-replit-python
      nix develop --command sh -c "if [ ! -d venv ]; then python -m venv venv
&& . venv/bin/activate && pip install -r requirements.txt; fi && . venv/bin/
activate && uvicorn main:app --host 0.0.0.0 --port 8000"
    """

    result = subprocess.run(command, shell=True, capture_output=True,
text=True)

    if result.returncode != 0:
        raise HTTPException(status_code=500, detail=result.stderr)

    return {"message": "Project running", "container": container_name}
```

📦 Example Request

POST /create

```
{
  "dependencies": {
    "fastapi": "0.110.0",
    "uvicorn": "0.27.0"
  }
}
```

🔥 What Happens Internally

1. User sends dependencies
2. System generates requirements.txt
3. User clicks /run
4. Docker container starts
5. Workspace mounted
6. Nix loads Python 3.11
7. If venv missing → create virtualenv
8. pip install dependencies
9. uvicorn runs FastAPI app

🛡️ Security Controls

Docker limits:

- 256 MB RAM
- 0.5 CPU
- 100 process limit

Prevents abuse and fork bombs

🚀 Production Improvements

- Replace subprocess with docker SDK for Python
- Add container stop endpoint
- Add log streaming
- Add port detection
- Add authentication
- Use Docker named volumes instead of host paths
- Use Kubernetes for scaling

✅ Final Result

You now have a fully dynamic mini Replit system in Python:

- User sends dependencies
- requirements.txt generated
- Docker + Nix provides runtime
- Virtualenv created inside container
- pip installs packages
- FastAPI app runs in isolated sandbox

This mirrors real cloud IDE architecture in a simplified educational form.