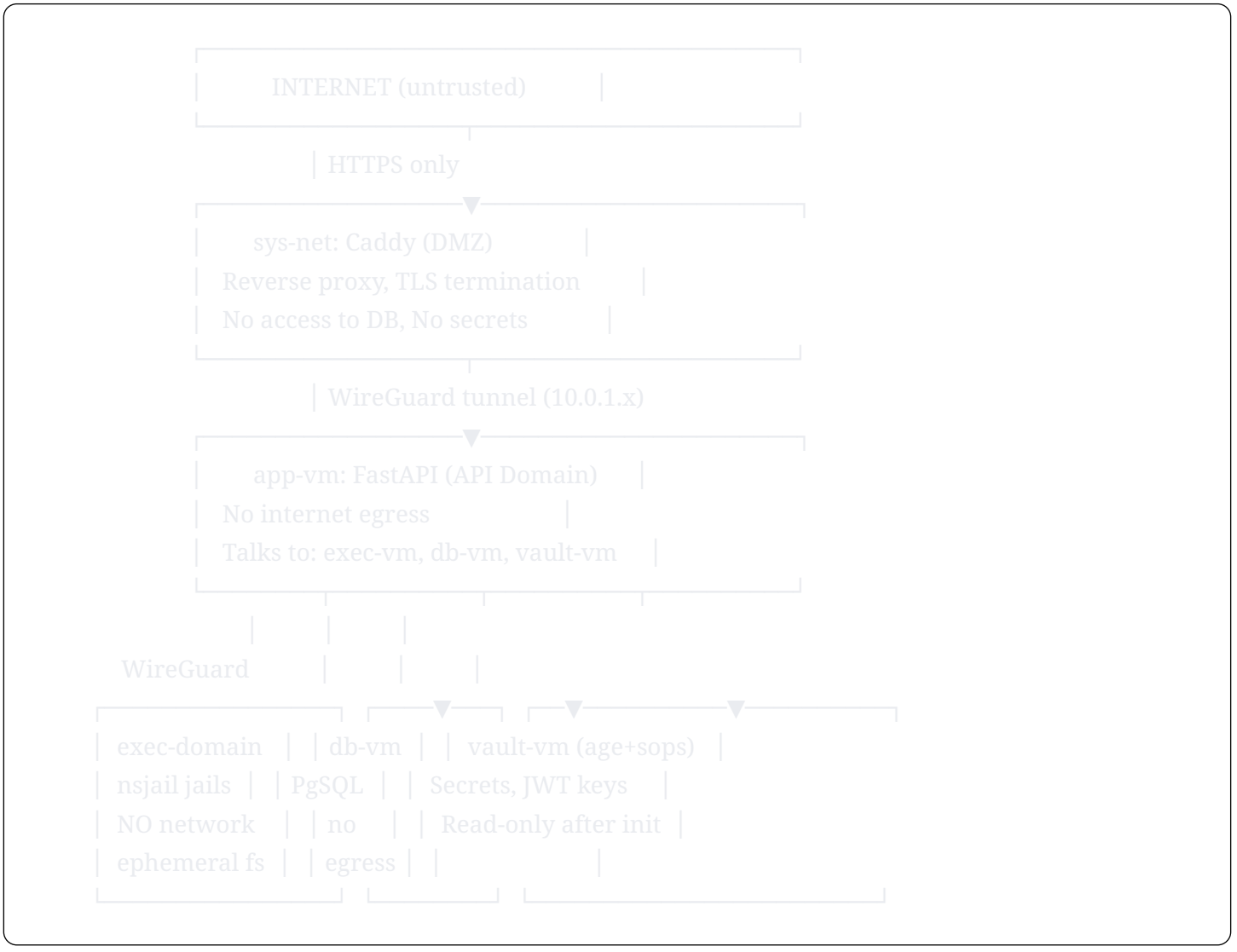
Sovereign Deployment Guide

Qubes-Inspired Compartmentalized Architecture for rust-learning-ground

> **Philosophy:** Every component lives in its own trust domain. Nothing is trusted by default. Compromise of one domain does not cascade. Code executions are disposable — born and destroyed in milliseconds, leaving no trace. No root daemons. No shared secrets. No single point of failure.

Architecture Overview: The Domain Model

Inspired by Qubes OS's compartmentalization model, the entire system is split into **isolated trust domains** that communicate only through narrow, well-defined channels.



Trust levels (Qubes-style):

Domain	Trust Level	Internet	Notes
sys-net (Caddy)	Untrusted	In only	Compromised = attacker sees only proxied traffic
app-vm (API)	Semi-trusted	None	Can't reach internet even if exploited
exec-domain	Disposable	None	Destroyed after every execution
db-vm	Trusted	None	Only accepts connections from app-vm subnet
vault-vm	High trust	None	Secrets loaded once at boot, never written to disk

---

Part 1: Infrastructure — Rootless, Immutable, Declarative

1.1 Base OS Choice: NixOS (recommended) or Debian Hardened

Why NixOS for sovereignty:

- Entire system declared in one file — reproducible bit-for-bit
- Atomic upgrades with rollback
- No package manager pollution (no `apt install random-thing` drift)
- Every service runs in isolation by default

`configuration.nix` — The entire server declared:

nix

```nix
{ config, pkgs, ... }:
{
  # ── Immutable base ──────────────────────────────────────────
  system.stateVersion = "24.05";
  boot.loader.grub.enable = true;

  # ── No password auth, no root login ─────────────────────────
  services.openssh = {
    enable = true;
    settings = {
      PasswordAuthentication = false;
      PermitRootLogin = "no";
      X11Forwarding = false;
    };
  };

  # ── Firewall: deny everything, whitelist explicitly ─────────
  networking.firewall = {
    enable = true;
    allowedTCPPorts = [ 80 443 ];       # Only Caddy faces internet
    allowedUDPPorts = [ 51820 ];        # WireGuard
  };

  # ── WireGuard inter-service mesh ────────────────────────────
  networking.wg-quick.interfaces.wg0 = {
    address = [ "10.0.1.1/24" ];
    privateKeyFile = "/run/secrets/wg_private_key";
    peers = [
      { publicKey = "APP_VM_PUBKEY"; allowedIPs = [ "10.0.1.2/32" ]; }
      { publicKey = "DB_VM_PUBKEY";  allowedIPs = [ "10.0.1.3/32" ]; }
    ];
  };

  # ── nsjail for code execution sandbox ───────────────────────
  environment.systemPackages = with pkgs; [
    nsjail
    rustup
    podman
    caddy
    age
    sops
  ];

  # ── Podman: rootless containers, no daemon ──────────────────
  virtualisation.podman = {
    enable = true;
    dockerCompat = true;
```

```
  defaultNetwork.settings.dns_enabled = true;
 };

 # ── AppArmor mandatory access control ─────────────────────
 security.apparmor.enable = true;

 # ── Automatic security updates (unattended) ───────────────
 system.autoUpgrade = {
  enable = true;
  allowReboot = false;
  dates = "04:00";
 };
}
```

1.2 Secret Management: `age` + `sops` (Zero plaintext at rest)

No `.env` files. No secrets in git. Ever.

bash

```
# Install tools
nix-env -iA nixpkgs.age nixpkgs.sops

# Generate a master age keypair (store private key OFFLINE or in password manager)
age-keygen -o master.key
# Public key: age1ql3z7hjy54pw3hyww5ayyfg7zqgvc7w3j2elw8zmrj2kg5sfn9aqmcac8p

# Create .sops.yaml at repo root
cat > .sops.yaml << 'EOF'
creation_rules:
 - path_regex: secrets/.*\.yaml$
   age: age1ql3z7hjy54pw3hyww5ayyfg7zqgvc7w3j2elw8zmrj2kg5sfn9aqmcac8p
EOF

# Encrypt your secrets
cat > secrets/prod.yaml << 'EOF'
jwt_secret: "your-256-bit-secret-here"
database_url: "postgresql://user:pass@10.0.1.3:5432/rustlearner"
wg_private_key: "..."
EOF

sops --encrypt --in-place secrets/prod.yaml
# secrets/prod.yaml is now safe to commit to git

# At deploy time, decrypt into memory only (never written to disk)
sops --decrypt secrets/prod.yaml | install -m 600 /dev/stdin /run/secrets/app_secrets
```

Part 2: The Execution Sandbox — Disposable VM Equivalent

This is the most critical part. Every piece of user-submitted Rust code runs in an **nsjail** environment that is the philosophical equivalent of a Qubes **Disposable VM**: ephemeral, isolated, network-free, destroyed after use.

2.1 Why nsjail over Docker/gVisor/Firecracker

Option	Startup	Isolation	Complexity	Used by
Docker	~500ms	Namespace	Medium	Everyone
gVisor	~200ms	Syscall intercept	High	Google
nsjail	**~5ms**	**Full namespace+seccomp**	**Low**	**Google CTF**
Firecracker	~125ms	Full microVM	High	AWS Lambda
Kata Containers	~1s	Full VM	Very High	OpenStack

nsjail wins for this use case: near-zero overhead, battle-tested by Google's security team for running untrusted code in CTF competitions, and fully open source.

2.2 nsjail Config: `nsjail_rust.cfg`

protobuf

```
# /etc/nsjail/rust_exec.cfg
# Complete isolation profile for Rust code execution

name: "rust_executor"

# ── Process limits ──────────────────────────────────────────
rlimit_as_type: INF          # Address space (Rust needs this for compilation)
rlimit_cpu_type: SOFT        # CPU time
rlimit_fsize_mb: 64          # Max output file size: 64MB
rlimit_nofile: 32            # Max open files
rlimit_nproc: 8              # Max child processes (for Rustc threads)
rlimit_stack_mb: 8

# ── No network whatsoever ──────────────────────────────────
disable_clone_newnet: false    # Create new network namespace
iface_lo: false              # No loopback either

# ── New user namespace (run as nobody) ─────────────────────
clone_newuser: true
uidmap {
  inside_id: 65534    # nobody
  outside_id: 65534
  count: 1
}
gidmap {
  inside_id: 65534
  outside_id: 65534
  count: 1
}

# ── Filesystem: minimal read-only mounts ───────────────────
mount_proc: false
mount {
  src: "/usr"
  dst: "/usr"
  is_bind: true
  rw: false
}
mount {
  src: "/lib"
  dst: "/lib"
  is_bind: true
  rw: false
}
mount {
  src: "/lib64"
  dst: "/lib64"
```

```
  is_bind: true
  rw: false
  mandatory: false
}
# Rust toolchain (read-only)
mount {
  src: "/root/.rustup/toolchains/stable-x86_64-unknown-linux-gnu"
  dst: "/rust"
  is_bind: true
  rw: false
}
# Ephemeral tmpfs for compilation — wiped on exit
mount {
  dst: "/tmp"
  fstype: "tmpfs"
  options: "size=128m,noexec,nosuid"
  rw: true
}
mount {
  dst: "/home/user"
  fstype: "tmpfs"
  options: "size=32m"
  rw: true
}

# ── Seccomp syscall whitelist (deny everything not listed) ──────
seccomp_string: "
  POLICY rust_policy {
    ALLOW { read, write, open, openat, close, fstat, lstat, stat,
            mmap, mprotect, munmap, brk, pread64, pwrite64,
            access, execve, exit, exit_group, wait4, clone,
            fork, vfork, getpid, getppid, getuid, geteuid,
            getgid, getegid, arch_prctl, set_tid_address,
            set_robust_list, futex, sched_yield, nanosleep,
            getcwd, chdir, mkdir, unlink, rename, readlink,
            ioctl, fcntl, dup, dup2, pipe, pipe2,
            poll, select, epoll_create, epoll_ctl, epoll_wait,
            socket, bind, listen, accept, connect, sendto, recvfrom,
            getsockname, getpeername, setsockopt, getsockopt,
            sigaltstack, rt_sigaction, rt_sigprocmask,
            rt_sigreturn, kill, tgkill, prctl, uname,
            lseek, writev, readv, prlimit64
    }
    # Explicitly deny dangerous syscalls
    DENY { ptrace, process_vm_readv, process_vm_writev,
           kexec_load, perf_event_open, bpf, userfaultfd }
  }
  USE rust_policy DEFAULT KILL
"
```

2.3 New `code_runner.py` — Disposable Execution Engine

```python
"""
code_runner.py — Qubes-inspired disposable execution
Each run = ephemeral jail, born and destroyed in milliseconds.
No shared state. No network. No persistence.
"""

import asyncio
import hashlib
import os
import shutil
import tempfile
import uuid
from dataclasses import dataclass
from pathlib import Path
from typing import Optional

# ── Constants ──────────────────────────────────────────────────────────────
NSJAIL_BIN      = "/usr/sbin/nsjail"
NSJAIL_CFG      = "/etc/nsjail/rust_exec.cfg"
RUSTC_PATH      = "/rust/bin/rustc"      # read-only mount inside jail
CARGO_PATH      = "/rust/bin/cargo"
COMPILE_TIMEOUT = 12   # seconds
RUN_TIMEOUT     = 5    # seconds
MAX_CODE_BYTES  = 65_536         # 64KB source limit
MAX_OUTPUT_BYTES = 65_536        # 64KB output limit
MAX_CONCURRENT  = 8              # semaphore: max parallel executions
SANDBOX_BASE    = Path("/var/sandboxes")  # tmpfs-backed, see systemd unit below


@dataclass
class ExecutionResult:
    success: bool
    stdout: str
    stderr: str
    compile_time_ms: int
    run_time_ms: int
    exit_code: int
    sandbox_id: str


class DisposableExecutor:
    """
    Each instance represents one ephemeral execution domain.
    Inspired by Qubes DisposableVMs — created on demand, destroyed on exit.
    """
    _semaphore = asyncio.Semaphore(MAX_CONCURRENT)

    def __init__(self):
```

```
def __init__(self):
    self.sandbox_id = str(uuid.uuid4())[:8]
    # Each sandbox gets its own tmpfs directory, never reused
    self.sandbox_dir = SANDBOX_BASE / self.sandbox_id

async def execute(self, code: str, exercise_id: str) -> ExecutionResult:
    """Run untrusted code in a disposable sandbox. Returns result."""
    if len(code.encode()) > MAX_CODE_BYTES:
        return ExecutionResult(
            success=False, stdout="", stderr="Code exceeds 64KB limit.",
            compile_time_ms=0, run_time_ms=0, exit_code=1,
            sandbox_id=self.sandbox_id
        )

    async with self._semaphore:
        try:
            return await self._run_sandboxed(code, exercise_id)
        finally:
            self._cleanup()

async def _run_sandboxed(self, code: str, exercise_id: str) -> ExecutionResult:
    # ── Create ephemeral workspace ──────────────────────────────
    self.sandbox_dir.mkdir(parents=True, exist_ok=True)
    src_path = self.sandbox_dir / "main.rs"
    bin_path = self.sandbox_dir / "main"
    src_path.write_text(code)

    # ── Phase 1: Compile inside nsjail ──────────────────────────
    compile_start = asyncio.get_event_loop().time()
    compile_result = await self._nsjail_run(
        cmd=[RUSTC_PATH, "/home/user/main.rs", "-o", "/home/user/main",
             "--edition", "2021", "-C", "opt-level=0"],
        bind_src=str(src_path),
        bind_dst="/home/user/main.rs",
        output_bind=str(bin_path),
        timeout=COMPILE_TIMEOUT,
    )
    compile_ms = int((asyncio.get_event_loop().time() - compile_start) * 1000)

    if compile_result["exit_code"] != 0:
        return ExecutionResult(
            success=False,
            stdout="",
            stderr=compile_result["stderr"][:MAX_OUTPUT_BYTES],
            compile_time_ms=compile_ms,
            run_time_ms=0,
            exit_code=compile_result["exit_code"],
            sandbox_id=self.sandbox_id,
        )
```

```
    # ── Phase 2: Execute compiled binary inside fresh nsjail ──────────
    run_start = asyncio.get_event_loop().time()
    run_result = await self._nsjail_run(
        cmd=["/home/user/main"],
        bind_src=str(bin_path),
        bind_dst="/home/user/main",
        timeout=RUN_TIMEOUT,
    )
    run_ms = int((asyncio.get_event_loop().time() - run_start) * 1000)

    return ExecutionResult(
        success=run_result["exit_code"] == 0,
        stdout=run_result["stdout"][:MAX_OUTPUT_BYTES],
        stderr=run_result["stderr"][:MAX_OUTPUT_BYTES],
        compile_time_ms=compile_ms,
        run_time_ms=run_ms,
        exit_code=run_result["exit_code"],
        sandbox_id=self.sandbox_id,
    )

async def _nsjail_run(
    self,
    cmd: list[str],
    bind_src: str,
    bind_dst: str,
    timeout: int,
    output_bind: Optional[str] = None,
) -> dict:
    """Invoke nsjail with config file + dynamic bind mounts."""
    nsjail_cmd = [
        NSJAIL_BIN,
        "--config", NSJAIL_CFG,
        "--log_fd", "3",          # nsjail logs to fd 3, not stderr
        "--bindmount_ro", f"{bind_src}:{bind_dst}",
    ]

    if output_bind:
        # Create empty output file so nsjail can bind it rw
        Path(output_bind).touch()
        nsjail_cmd += ["--bindmount", f"{output_bind}:/home/user/main"]

    nsjail_cmd += ["--", *cmd]

    try:
        proc = await asyncio.wait_for(
            asyncio.create_subprocess_exec(
                *nsjail_cmd,
                stdout=asyncio.subprocess.PIPE,
                stderr=asyncio.subprocess.PIPE,
```

```
                pass_fds=(3,),  # nsjail log fd
            ),
            timeout=timeout + 2,
        )
        stdout, stderr = await asyncio.wait_for(
            proc.communicate(), timeout=timeout
        )
        return {
            "stdout": stdout.decode(errors="replace"),
            "stderr": stderr.decode(errors="replace"),
            "exit_code": proc.returncode or 0,
        }
    except asyncio.TimeoutError:
        try:
            proc.kill()
        except Exception:
            pass
        return {
            "stdout": "",
            "stderr": f"Execution timed out after {timeout}s",
            "exit_code": 124,
        }

def _cleanup(self):
    """Destroy the ephemeral sandbox. No trace left."""
    try:
        shutil.rmtree(self.sandbox_dir, ignore_errors=True)
    except Exception:
        pass


# ── Public API ──────────────────────────────────────────────────────
async def run_code(code: str, exercise_id: str = "") -> ExecutionResult:
    """
    Entry point. Creates a disposable executor, runs code, destroys everything.
    Caller never interacts with the sandbox directly.
    """
    executor = DisposableExecutor()
    return await executor.execute(code, exercise_id)
```

2.4 Systemd Hardening for the API Service

ini

```ini
# /etc/systemd/system/rust-learner.service
[Unit]
Description=Rust Learning Platform API
After=network.target postgresql.service
Requires=postgresql.service

[Service]
Type=exec
User=rustlearner
Group=rustlearner
WorkingDirectory=/opt/rust-learner/server
ExecStart=/opt/rust-learner/venv/bin/uvicorn app.main:app \
    --host 10.0.1.2 --port 8000 --workers 4

# ── Systemd sandboxing (free hardening layer) ──────────────────
NoNewPrivileges=yes
PrivateTmp=yes              # Isolated /tmp
PrivateDevices=yes          # No access to /dev
ProtectSystem=strict        # Read-only system dirs
ProtectHome=yes             # No access to /home
ProtectKernelTunables=yes
ProtectKernelModules=yes
ProtectControlGroups=yes
RestrictNamespaces=~user    # Can't create new user namespaces (nsjail handles this)
RestrictRealtime=yes
RestrictSUIDSGID=yes
LockPersonality=yes
SystemCallFilter=@system-service  # Whitelist systemd's "sane service" syscall set
SystemCallErrorNumber=EPERM

# ── Resource limits ────────────────────────────────────────────
LimitNOFILE=65536
LimitNPROC=512
MemoryMax=2G
CPUQuota=200%

# ── Environment from sops-decrypted secrets ────────────────────
EnvironmentFile=/run/secrets/app_secrets

[Install]
WantedBy=multi-user.target
```

Part 3: Rootless Podman Compose (No root daemon, ever)

Docker daemon runs as root. Podman does not. This matters.

yaml

```yaml
# compose.prod.yml — Podman-compatible

version: "3.9"

networks:
  internal:
    driver: bridge
    internal: true        # No internet access from any container
  dmz:
    driver: bridge

services:
  # ── sys-net: The only internet-facing component ──────────────────
  caddy:
    image: docker.io/caddy:2-alpine
    networks: [dmz, internal]
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./Caddyfile:/etc/caddy/Caddyfile:ro
      - caddy_data:/data
      - caddy_config:/config
    cap_drop: [ALL]
    cap_add: [NET_BIND_SERVICE]
    read_only: true
    tmpfs: [/tmp]
    security_opt: [no-new-privileges:true]

  # ── app-vm: API, no internet ─────────────────────────────────────
  api:
    build:
      context: ./server
      dockerfile: Dockerfile.hardened
    networks: [internal]    # NOT in dmz — can't reach internet
    depends_on: [postgres]
    environment:
      - JWT_SECRET_FILE=/run/secrets/jwt_secret
      - DATABASE_URL_FILE=/run/secrets/db_url
    secrets: [jwt_secret, db_url]
    volumes:
      - ./problems:/app/problems:ro
      - /var/sandboxes:/var/sandboxes   # tmpfs-backed in host
      - /usr/sbin/nsjail:/usr/sbin/nsjail:ro
      - /etc/nsjail:/etc/nsjail:ro
    cap_drop: [ALL]
    cap_add: [SYS_ADMIN]    # Required for nsjail namespace creation only
```

```yaml
    security_opt:
      - no-new-privileges:true
      - apparmor:rust-learner-api
    read_only: true
    tmpfs: [/tmp, /var/cache]

  # ── db-vm: PostgreSQL, isolated ──────────────────────────────
  postgres:
    image: docker.io/postgres:16-alpine
    networks: [internal]
    volumes:
      - pgdata:/var/lib/postgresql/data
    environment:
      - POSTGRES_DB=rustlearner
      - POSTGRES_USER_FILE=/run/secrets/db_user
      - POSTGRES_PASSWORD_FILE=/run/secrets/db_pass
    secrets: [db_user, db_pass]
    cap_drop: [ALL]
    cap_add: [SETUID, SETGID, DAC_OVERRIDE]
    security_opt: [no-new-privileges:true]
    read_only: true
    tmpfs: [/tmp, /run/postgresql]

volumes:
  caddy_data:
  caddy_config:
  pgdata:

secrets:
  jwt_secret:
    file: /run/secrets/jwt_secret    # Decrypted by sops at boot, tmpfs only
  db_url:
    file: /run/secrets/db_url
  db_user:
    file: /run/secrets/db_user
  db_pass:
    file: /run/secrets/db_pass
```

Part 4: Hardened Dockerfile

dockerfile

```
# Dockerfile.hardened — Multi-stage, minimal attack surface

# ── Stage 1: Build deps ──────────────────────────────────────────────
FROM python:3.12-slim AS builder
WORKDIR /build
COPY requirements.txt .
RUN pip install --no-cache-dir --prefix=/install -r requirements.txt

# ── Stage 2: Rust toolchain for sandboxed execution ──────────────────
FROM rust:1.75-slim AS rust-toolchain
# Pre-compile a dummy project to warm the cache
RUN cargo new --bin warmup && cd warmup && cargo build --release

# ── Stage 3: Final runtime — smallest possible image ─────────────────
FROM debian:bookworm-slim AS runtime

# Install only absolute minimums
RUN apt-get update && apt-get install -y --no-install-recommends \
    libssl3 ca-certificates \
    && rm -rf /var/lib/apt/lists/*

# Copy Python runtime from builder
COPY --from=builder /install /usr/local

# Copy Rust toolchain (read-only bind mount in production, copy for portability)
COPY --from=rust-toolchain /usr/local/cargo /opt/cargo
COPY --from=rust-toolchain /usr/local/rustup /opt/rustup
ENV CARGO_HOME=/opt/cargo RUSTUP_HOME=/opt/rustup PATH="/opt/cargo/bin:$PATH"

# Create non-root user
RUN groupadd -r rustlearner && useradd -r -g rustlearner -s /sbin/nologin rustlearner

WORKDIR /app
COPY --chown=rustlearner:rustlearner . .

# ── Remove everything that shouldn't be in a production image ────────
RUN find /app -name "*.py[co]" -delete \
    && find /app -name "__pycache__" -type d -exec rm -rf {} + \
    && find /app -name "test_*.py" -delete \
    && find /app -name "*.md" -delete

USER rustlearner

EXPOSE 8000
HEALTHCHECK --interval=30s --timeout=5s --start-period=10s \
    CMD python -c "import urllib.request; urllib.request.urlopen('http://localhost:8000/api/v4/health')"

CMD ["python", "-m", "uvicorn", "server:app", "--host", "0.0.0.0", "--port", "8000"]
```

```
CMD ["python", "-m", "uvicorn", "app.main:app",
    "--host", "0.0.0.0", "--port", "8000",
    "--workers", "4",
    "--no-access-log"]
```

Part 5: Caddyfile — TLS + Security Headers

caddyfile

```
# Caddyfile — Modern TLS, HSTS, hardened headers

{
    # Use Let's Encrypt (or ZeroSSL as alternative CA)
    email admin@yourdomain.com
    # Uncomment for Tor hidden service:
    # servers {
    #     listener_wrappers {
    #         http_redirect
    #         tls
    #     }
    # }
}

yourdomain.com {
    # ── Reverse proxy to API (internal network only) ──────────────
    reverse_proxy api:8000 {
        health_uri /api/v4/health
        health_interval 30s
        header_up X-Real-IP {remote_host}
        header_up X-Forwarded-Proto {scheme}
    }

    # ── Security headers ──────────────────────────────────────────
    header {
        Strict-Transport-Security "max-age=31536000; includeSubDomains; preload"
        X-Content-Type-Options "nosniff"
        X-Frame-Options "DENY"
        X-XSS-Protection "1; mode=block"
        Referrer-Policy "no-referrer"
        Permissions-Policy "geolocation=(), camera=(), microphone=()"
        Content-Security-Policy "default-src 'self'; script-src 'self' 'unsafe-inline'; style-src 'self' 'unsafe-inline'"
        # Remove server fingerprinting
        -Server
        -X-Powered-By
    }

    # ── Rate limiting (built into Caddy v2.7+) ────────────────────
    # Protects code execution endpoint
    @exec_endpoint path /api/v4/exercises/*/run /api/v4/exercises/*/submit
    rate_limit @exec_endpoint 10r/m

    # ── Logging: structured, no PII ───────────────────────────────
    log {
        output file /var/log/caddy/access.log {
            roll_size 100mb
            roll_keep 5
        }
```

```
        }
        format json
    }
}
```

Part 6: Self-Hosted CI/CD — Woodpecker + Gitea

Zero GitHub. Zero cloud. Your code, your pipeline.

yaml

```yaml
# .woodpecker.yml — CI pipeline

pipeline:
  # ── Security: check for leaked secrets ──────────────────
  secret-scan:
    image: docker.io/trufflesecurity/trufflehog:latest
    commands:
      - trufflehog filesystem . --fail

  # ── Dependency audit ────────────────────────────────────
  audit:
    image: python:3.12-slim
    commands:
      - pip install pip-audit
      - pip-audit -r server/requirements.txt

  # ── Tests ───────────────────────────────────────────────
  test:
    image: python:3.12-slim
    commands:
      - cd server
      - pip install -r requirements.txt
      - pytest tests/ -v --tb=short

  # ── Build image (only on main branch) ──────────────────
  build:
    image: docker.io/podman/stable
    when:
      branch: main
    commands:
      - podman build -f server/Dockerfile.hardened -t gitea.yourdomain.com/james/rust-learner:${CI_COMMI
      - podman push gitea.yourdomain.com/james/rust-learner:${CI_COMMIT_SHA}

  # ── Deploy ──────────────────────────────────────────────
  deploy:
    image: docker.io/alpine/ssh
    when:
      branch: main
    secrets: [deploy_ssh_key]
    commands:
      - echo "$DEPLOY_SSH_KEY" > /tmp/key && chmod 600 /tmp/key
      - ssh -i /tmp/key deploy@yourvps.com "cd /opt/rust-learner && git pull && podman-compose -f compos
```

Part 7: Full Deployment Playbook

Day 1: Provision

```bash
# 1. Spin up VPS (Hetzner CX22 recommended — EU datacenter, no US jurisdiction)
#    Use Hetzner Cloud CLI or their console
#    OS: Debian 12 Bookworm or NixOS 24.05

# 2. Initial hardening (run as root, then never again)
ssh root@YOUR_VPS_IP << 'EOF'
# Create deploy user
useradd -m -s /bin/bash deploy
mkdir -p /home/deploy/.ssh
cp ~/.ssh/authorized_keys /home/deploy/.ssh/
chown -R deploy:deploy /home/deploy/.ssh

# Disable root SSH
sed -i 's/PermitRootLogin yes/PermitRootLogin no/' /etc/ssh/sshd_config
sed -i 's/#PasswordAuthentication yes/PasswordAuthentication no/' /etc/ssh/sshd_config
systemctl restart sshd

# Install essentials
apt-get update && apt-get install -y \
  nsjail podman git age sops \
  fail2ban ufw unattended-upgrades

# Firewall
ufw default deny incoming
ufw default allow outgoing
ufw allow 22/tcp   # SSH
ufw allow 80/tcp   # HTTP (Caddy redirect)
ufw allow 443/tcp  # HTTPS
ufw --force enable

# Auto-updates
dpkg-reconfigure -plow unattended-upgrades
EOF
```

Day 2: Deploy

```bash
# 3. Switch to deploy user
ssh deploy@YOUR_VPS_IP

# 4. Clone your Gitea repo (self-hosted) or GitHub
git clone https://git.yourdomain.com/james/rust-learner.git /opt/rust-learner
cd /opt/rust-learner

# 5. Set up secrets
# (Do this on your LOCAL machine with the age private key)
sops --decrypt secrets/prod.yaml > /tmp/decrypted_secrets
# Transfer decrypted secrets to VPS via SSH (in-memory, not saved to disk)
cat /tmp/decrypted_secrets | ssh deploy@YOUR_VPS_IP \
  "sudo install -m 600 -o deploy /dev/stdin /run/secrets/app_secrets"
rm /tmp/decrypted_secrets

# 6. Prepare sandbox tmpfs (cleared on reboot automatically)
sudo mkdir -p /var/sandboxes
sudo mount -t tmpfs -o size=2G,noexec,nosuid tmpfs /var/sandboxes
# Add to /etc/fstab for persistence:
echo "tmpfs /var/sandboxes tmpfs size=2G,noexec,nosuid 0 0" | sudo tee -a /etc/fstab

# 7. Deploy
podman-compose -f compose.prod.yml up -d

# 8. Verify all containers are up
podman ps
podman logs rust-learner-api-1

# 9. Check the sandbox works
curl -H "Authorization: Bearer YOUR_JWT" \
    -H "Content-Type: application/json" \
    -d '{"code": "fn main() { println!(\"Hello, sovereign world!\"); }"}' \
    https://yourdomain.com/api/v4/exercises/1/run
```

Part 8: Monitoring — Open Source, Privacy-First

yaml

```yaml
# Observability stack — no Datadog, no New Relic, no telemetry phones home

  # Metrics: VictoriaMetrics (Prometheus-compatible, lighter)
  victoria-metrics:
    image: docker.io/victoriametrics/victoria-metrics:latest
    networks: [internal]
    volumes:
      - vm_data:/victoria-metrics-data
    command: ["-retentionPeriod=12"]   # 12 months retention
    cap_drop: [ALL]

  # Dashboards: Grafana OSS
  grafana:
    image: docker.io/grafana/grafana-oss:latest
    networks: [internal]
    environment:
      - GF_AUTH_ANONYMOUS_ENABLED=false
      - GF_SERVER_ROOT_URL=https://metrics.yourdomain.com
      - GF_SECURITY_ADMIN_PASSWORD_FILE=/run/secrets/grafana_pass
    volumes:
      - grafana_data:/var/lib/grafana

  # Log aggregation: Loki (no external SaaS)
  loki:
    image: docker.io/grafana/loki:latest
    networks: [internal]
    volumes:
      - loki_data:/loki
```

Threat Model: What This Architecture Defeats

Attack	Mitigation
Malicious Rust code reads /etc/passwd	nsjail: no /etc mount, seccomp blocks open() on host paths
Malicious code exfiltrates via network	nsjail: new network namespace, no interfaces
Malicious code forks bomb	rlimit_nproc=8, cgroup limits
Malicious code writes infinite output	MAX_OUTPUT_BYTES=64KB enforced
Code execution escapes container	Runs in nsjail inside Podman — two isolation layers
JWT secret extraction if API is compromised	Secrets in /run (tmpfs), never on disk, API has read-only filesystem
Database breach	DB is on internal network only, no internet, encrypted at rest
Supply chain attack on Docker Hub	All images pinned by SHA256 digest, built from source where possible
SSH brute force	fail2ban + key-only auth + non-standard port optional
DDoS on code runner	Rate limiting in Caddy + semaphore in code_runner.py

---

License: AGPL-3.0

Add this to your repo root as `LICENSE`:

```
AGPL-3.0 — Affero General Public License v3.0

This means: anyone who deploys this software as a network service
must release their modifications under the same license.
Corporate capture requires open-sourcing back.
Sovereignty propagates.
```

"Security is not a product, it is a process." — Bruce Schneier "Compartmentalization is the only honest security model." — Qubes OS team