STUNIR Deployment Guide

Complete guide for deploying STUNIR in production environments.

Table of Contents

Deployment Options

Method	Best For	Complexity
Docker	Most deployments	Low
Bare Metal	Maximum performance	Medium
Kubernetes	Large scale	High
AWS/GCP/Azure	Cloud native	Medium

Docker Deployment

Quick Start

```
# Build the image
docker build -t stunir:latest .

# Run container
docker run -d \
  --name stunir \
  -v /path/to/specs:/data/specs:ro \
  -v /path/to/output:/data/output \
  -e STUNIR_MAX_FILE_SIZE=104857600 \
  stunir:latest
```

Dockerfile

```
# Dockerfile
FROM python:3.11-slim as python-base

# Install system dependencies
RUN apt-get update && apt-get install -y \
    curl \
    build-essential \
    && rm -rf /var/lib/apt/lists/*

# Install Rust
RUN curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh -s -- -y
ENV PATH="/root/.cargo/bin:${PATH}"

# Set working directory
WORKDIR /app

# Copy requirements first for caching
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Build Rust tools
COPY tools/native/rust/stunir-native tools/native/rust/stunir-native
WORKDIR /app/tools/native/rust/stunir-native
RUN cargo build --release

# Copy rest of application
WORKDIR /app
COPY . .

# Add Rust binary to PATH
ENV PATH="/app/tools/native/rust/stunir-native/target/release:${PATH}"

# Create non-root user
RUN useradd -m -s /bin/bash stunir
USER stunir

# Health check
HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --retries=3 \
  CMD python -c "from tools.ir_emitter import emit_ir; print('OK')"

# Default command
ENTRYPOINT ["python", "-m", "tools.cli"]
CMD ["--help"]
```

Docker Compose

```yaml
# docker-compose.yml
version: '3.8'

services:
  stunir:
    build: .
    image: stunir:latest
    container_name: stunir
    restart: unless-stopped
    volumes:
      - ./specs:/data/specs:ro
      - ./output:/data/output
      - ./receipts:/data/receipts
    environment:
      - STUNIR_MAX_FILE_SIZE=104857600
      - STUNIR_VERBOSE=false
    deploy:
      resources:
        limits:
          cpus: '2'
          memory: 4G
        reservations:
          cpus: '0.5'
          memory: 512M
    logging:
      driver: json-file
      options:
        max-size: "10m"
        max-file: "3"

  # Optional: Web UI
  stunir-ui:
    image: nginx:alpine
    volumes:
      - ./docs:/usr/share/nginx/html:ro
    ports:
      - "8080:80"
    depends_on:
      - stunir
```

Multi-Stage Build (Production)

```
# Multi-stage build for smaller image
FROM rust:1.75-slim as rust-builder
WORKDIR /build
COPY tools/native/rust/stunir-native .
RUN cargo build --release

FROM python:3.11-slim as final
WORKDIR /app

# Copy only the built binary
COPY --from=rust-builder /build/target/release/stunir-native /usr/local/bin/

# Install Python dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application
COPY tools/ tools/
COPY manifests/ manifests/
COPY targets/ targets/
COPY scripts/ scripts/

# Security hardening
RUN useradd -m -s /bin/false stunir && \
    chown -R stunir:stunir /app
USER stunir

ENTRYPOINT ["python", "-m", "tools.cli"]
```

Bare Metal Deployment

System Requirements

Component	Minimum	Recommended
CPU	2 cores	4+ cores
RAM	2 GB	8+ GB
Disk	10 GB	50+ GB SSD
OS	Ubuntu 20.04+	Ubuntu 22.04 LTS

Installation Script

```
#!/bin/bash
# install_stunir.sh

set -e

# Check if running as root
if [[ $EUID -ne 0 ]]; then
   echo "This script must be run as root"
   exit 1
fi

# Variables
STUNIR_USER="stunir"
STUNIR_DIR="/opt/stunir"
PYTHON_VERSION="3.11"

echo "=== Installing STUNIR ==="

# Install system dependencies
apt-get update
apt-get install -y \
    python${PYTHON_VERSION} \
    python${PYTHON_VERSION}-venv \
    python3-pip \
    curl \
    build-essential \
    git

# Install Rust
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh -s -- -y
source "$HOME/.cargo/env"

# Create stunir user
useradd -m -s /bin/bash -d /home/${STUNIR_USER} ${STUNIR_USER} || true

# Create directories
mkdir -p ${STUNIR_DIR}/{bin,config,data,logs,receipts}
chown -R ${STUNIR_USER}:${STUNIR_USER} ${STUNIR_DIR}

# Clone repository
git clone https://github.com/org/stunir.git ${STUNIR_DIR}/src

# Setup Python environment
python${PYTHON_VERSION} -m venv ${STUNIR_DIR}/venv
${STUNIR_DIR}/venv/bin/pip install -r ${STUNIR_DIR}/src/requirements.txt

# Build Rust tools
cd ${STUNIR_DIR}/src/tools/native/rust/stunir-native
cargo build --release
cp target/release/stunir-native ${STUNIR_DIR}/bin/

# Create systemd service
cat > /etc/systemd/system/stunir.service << 'EOF'
[Unit]
Description=STUNIR Build Service
After=network.target

[Service]
Type=simple
User=stunir
Group=stunir
WorkingDirectory=/opt/stunir/src
```

```
Environment="PATH=/opt/stunir/bin:/opt/stunir/venv/bin:/usr/bin"
ExecStart=/opt/stunir/venv/bin/python -m tools.service
Restart=always
RestartSec=10

# Security settings
NoNewPrivileges=true
PrivateTmp=true
ProtectSystem=strict
ProtectHome=true
ReadWritePaths=/opt/stunir/data /opt/stunir/receipts /opt/stunir/logs

[Install]
WantedBy=multi-user.target
EOF

# Enable and start service
systemctl daemon-reload
systemctl enable stunir
systemctl start stunir

echo "=== STUNIR installed successfully ==="
echo "Check status: systemctl status stunir"
echo "View logs: journalctl -u stunir -f"
```

Configuration File

```
# /opt/stunir/config/stunir.conf

# General settings
STUNIR_ENV=production
STUNIR_VERBOSE=false
STUNIR_LOG_LEVEL=INFO

# Paths
STUNIR_DATA_DIR=/opt/stunir/data
STUNIR_RECEIPTS_DIR=/opt/stunir/receipts
STUNIR_LOG_DIR=/opt/stunir/logs

# Limits
STUNIR_MAX_FILE_SIZE=104857600    # 100MB
STUNIR_MAX_JSON_DEPTH=50
STUNIR_MAX_DIR_DEPTH=100

# Security
STUNIR_ALLOW_SYMLINKS=false
STUNIR_STRICT_MODE=true

# Performance
STUNIR_WORKER_COUNT=4
STUNIR_HASH_BUFFER_SIZE=8192
```

Cloud Deployment

AWS Deployment

Using AWS Lambda

```yaml
# serverless.yml
service: stunir

provider:
  name: aws
  runtime: python3.11
  region: us-east-1
  memorySize: 1024
  timeout: 300

  environment:
    STUNIR_MAX_FILE_SIZE: 52428800
    S3_BUCKET: ${self:custom.bucket}

custom:
  bucket: stunir-artifacts-${self:provider.stage}

functions:
  processSpec:
    handler: handler.process_spec
    events:
      - s3:
          bucket: ${self:custom.bucket}
          event: s3:ObjectCreated:*
          rules:
            - prefix: specs/
            - suffix: .json

  verifyBuild:
    handler: handler.verify_build
    events:
      - http:
          path: verify
          method: post

resources:
  Resources:
    ArtifactsBucket:
      Type: AWS::S3::Bucket
      Properties:
        BucketName: ${self:custom.bucket}
        VersioningConfiguration:
          Status: Enabled
```

AWS Lambda Handler

```python
# handler.py
import json
import boto3
import os
from tools.ir_emitter.emit_ir import spec_to_ir, canonical_json

s3 = boto3.client('s3')
BUCKET = os.environ['S3_BUCKET']

def process_spec(event, context):
    """Process spec file uploaded to S3."""

    # Get the uploaded file
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = event['Records'][0]['s3']['object']['key']

    # Download spec
    response = s3.get_object(Bucket=bucket, Key=key)
    spec = json.loads(response['Body'].read())

    # Convert to IR
    ir = spec_to_ir(spec)
    ir_json = canonical_json(ir)

    # Upload IR
    ir_key = key.replace('specs/', 'ir/').replace('.json', '_ir.json')
    s3.put_object(
        Bucket=bucket,
        Key=ir_key,
        Body=ir_json,
        ContentType='application/json'
    )

    return {
        'statusCode': 200,
        'body': json.dumps({
            'message': 'Spec processed',
            'ir_key': ir_key
        })
    }

def verify_build(event, context):
    """Verify a build via HTTP endpoint."""

    body = json.loads(event['body'])
    manifest_key = body['manifest_key']

    # Download and verify manifest
    response = s3.get_object(Bucket=BUCKET, Key=manifest_key)
    manifest = json.loads(response['Body'].read())

    # Verify each entry
    errors = []
    for entry in manifest.get('entries', []):
        # Verification logic...
        pass

    return {
        'statusCode': 200 if not errors else 400,
        'body': json.dumps({
            'valid': len(errors) == 0,
            'errors': errors
```

```
        })
    }
```

Google Cloud Deployment

```yaml
# cloudbuild.yaml
steps:
  # Build container
  - name: 'gcr.io/cloud-builders/docker'
    args: ['build', '-t', 'gcr.io/$PROJECT_ID/stunir:$COMMIT_SHA', '.']

  # Push to registry
  - name: 'gcr.io/cloud-builders/docker'
    args: ['push', 'gcr.io/$PROJECT_ID/stunir:$COMMIT_SHA']

  # Deploy to Cloud Run
  - name: 'gcr.io/google.com/cloudsdktool/cloud-sdk'
    entrypoint: gcloud
    args:
      - 'run'
      - 'deploy'
      - 'stunir'
      - '--image=gcr.io/$PROJECT_ID/stunir:$COMMIT_SHA'
      - '--region=us-central1'
      - '--platform=managed'
      - '--memory=2Gi'
      - '--cpu=2'
      - '--set-env-vars=STUNIR_MAX_FILE_SIZE=104857600'
```

Configuration

Environment Variables

Variable	Description	Default
`STUNIR_ENV`	Environment (dev/staging/prod)	`production`
`STUNIR_MAX_FILE_SIZE`	Max file size in bytes	`104857600` (100MB)
`STUNIR_MAX_JSON_DEPTH`	Max JSON nesting depth	`50`
`STUNIR_VERBOSE`	Enable verbose logging	`false`
`STUNIR_LOG_LEVEL`	Log level	`INFO`
`STUNIR_RECEIPTS_DIR`	Receipts directory	`receipts/`
`STUNIR_ALLOW_SYMLINKS`	Allow symlinks	`false`
`STUNIR_STRICT_MODE`	Enable strict verification	`true`

Configuration File Format

```
// stunir.config.json
{
  "version": "1.0",
  "environment": "production",

  "paths": {
    "data_dir": "/opt/stunir/data",
    "receipts_dir": "/opt/stunir/receipts",
    "logs_dir": "/opt/stunir/logs"
  },

  "limits": {
    "max_file_size": 104857600,
    "max_json_depth": 50,
    "max_dir_depth": 100,
    "max_workers": 4
  },

  "security": {
    "allow_symlinks": false,
    "strict_mode": true,
    "verify_on_build": true
  },

  "logging": {
    "level": "INFO",
    "format": "json",
    "output": "file",
    "rotation": {
      "max_size": "100MB",
      "max_files": 10
    }
  }
}
```

Security Considerations

Hardening Checklist

- [ ] Run as non-root user
- [ ] Enable strict mode
- [ ] Disable symlink following
- [ ] Set appropriate file size limits
- [ ] Use read-only mounts where possible
- [ ] Enable audit logging
- [ ] Keep dependencies updated
- [ ] Use HTTPS for all communications

File Permissions

```
# Recommended permissions
chmod 755 /opt/stunir
chmod 750 /opt/stunir/bin
chmod 640 /opt/stunir/config/*
chmod 700 /opt/stunir/data
chmod 700 /opt/stunir/receipts

chown -R stunir:stunir /opt/stunir
```

Network Security

```
# nginx.conf (if using reverse proxy)
server {
    listen 443 ssl http2;
    server_name stunir.example.com;

    ssl_certificate /etc/ssl/certs/stunir.crt;
    ssl_certificate_key /etc/ssl/private/stunir.key;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256;

    # Rate limiting
    limit_req_zone $binary_remote_addr zone=stunir:10m rate=10r/s;
    limit_req zone=stunir burst=20 nodelay;

    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

Performance Tuning

System Tuning

```
# /etc/sysctl.conf additions
# Increase file descriptors
fs.file-max = 65535

# Optimize network
net.core.somaxconn = 65535
net.ipv4.tcp_max_syn_backlog = 65535
```

Application Tuning

```
# Recommended settings for high throughput
STUNIR_CONFIG = {
    "limits": {
        "max_workers": os.cpu_count() or 4,
        "hash_buffer_size": 65536,  # 64KB for I/O
    },
    "performance": {
        "enable_caching": True,
        "cache_ttl": 3600,
        "parallel_hashing": True,
    }
}
```

Resource Allocation

Workload	CPU	RAM	Disk
Light (<100 files/min)	2	2GB	HDD
Medium (<1000 files/min)	4	4GB	SSD
Heavy (<10000 files/min)	8+	8GB+	NVMe

Monitoring and Logging

Structured Logging

```
# logging_config.py
import logging
import json

class JSONFormatter(logging.Formatter):
    def format(self, record):
        log_record = {
            "timestamp": self.formatTime(record),
            "level": record.levelname,
            "message": record.getMessage(),
            "module": record.module,
            "function": record.funcName,
        }
        if record.exc_info:
            log_record["exception"] = self.formatException(record.exc_info)
        return json.dumps(log_record)

# Configure logging
handler = logging.StreamHandler()
handler.setFormatter(JSONFormatter())
logging.getLogger("stunir").addHandler(handler)
```

Metrics

```
# metrics.py
import time
from prometheus_client import Counter, Histogram, Gauge

# Counters
files_processed = Counter('stunir_files_processed_total', 'Total files processed')
verification_success = Counter('stunir_verification_success_total', 'Successful verifications')
verification_failure = Counter('stunir_verification_failure_total', 'Failed verifications')

# Histograms
hash_duration = Histogram('stunir_hash_duration_seconds', 'Time to hash files')
build_duration = Histogram('stunir_build_duration_seconds', 'Total build time')

# Gauges
active_workers = Gauge('stunir_active_workers', 'Number of active workers')
```

Health Checks

```python
# health.py
from flask import Flask, jsonify
import subprocess

app = Flask(__name__)

@app.route('/health')
def health():
    checks = {
        'python_tools': check_python_tools(),
        'rust_tools': check_rust_tools(),
        'disk_space': check_disk_space(),
    }

    status = 'healthy' if all(checks.values()) else 'unhealthy'
    code = 200 if status == 'healthy' else 503

    return jsonify({'status': status, 'checks': checks}), code

def check_python_tools():
    try:
        from tools.ir_emitter import emit_ir
        return True
    except ImportError:
        return False

def check_rust_tools():
    try:
        result = subprocess.run(['stunir-native', '--version'],
                                capture_output=True, timeout=5)
        return result.returncode == 0
    except Exception:
        return False

def check_disk_space():
    import shutil
    total, used, free = shutil.disk_usage('/')
    return free > 1024 * 1024 * 1024  # At least 1GB free
```

Backup and Recovery

Backup Strategy

```
#!/bin/bash
# backup_stunir.sh

BACKUP_DIR="/backups/stunir/$(date +%Y%m%d)"
STUNIR_DIR="/opt/stunir"

mkdir -p "$BACKUP_DIR"

# Backup configuration
cp -r "$STUNIR_DIR/config" "$BACKUP_DIR/"

# Backup receipts
tar -czf "$BACKUP_DIR/receipts.tar.gz" -C "$STUNIR_DIR" receipts/

# Backup data (optional)
tar -czf "$BACKUP_DIR/data.tar.gz" -C "$STUNIR_DIR" data/

# Cleanup old backups (keep 30 days)
find /backups/stunir -type d -mtime +30 -exec rm -rf {} +

echo "Backup completed: $BACKUP_DIR"
```

Recovery Procedure

```
#!/bin/bash
# restore_stunir.sh

BACKUP_DIR="$1"
STUNIR_DIR="/opt/stunir"

if [ -z "$BACKUP_DIR" ]; then
    echo "Usage: $0 <backup_directory>"
    exit 1
fi

# Stop service
systemctl stop stunir

# Restore configuration
cp -r "$BACKUP_DIR/config/"* "$STUNIR_DIR/config/"

# Restore receipts
tar -xzf "$BACKUP_DIR/receipts.tar.gz" -C "$STUNIR_DIR"

# Restore data (if exists)
if [ -f "$BACKUP_DIR/data.tar.gz" ]; then
    tar -xzf "$BACKUP_DIR/data.tar.gz" -C "$STUNIR_DIR"
fi

# Fix permissions
chown -R stunir:stunir "$STUNIR_DIR"

# Start service
systemctl start stunir

echo "Restore completed from: $BACKUP_DIR"
```

See Also

- User Guide (USER_GUIDE.md) - Getting started
- Architecture (ARCHITECTURE.md) - System design
- API Reference (API_REFERENCE.md) - API documentation
- Security Policy (../SECURITY.md) - Security guidelines
