USB Microscope Camera Web Interface

Table of Contents

1. Description

Docker container setup for streaming USB microscope camera via web interface on Raspberry Pi Zero.

Why?

- Had a few zeros hanging around (one used here for now)
- had a camera on a microscope (USB and HDMI)
- have some assorted pi cameras too (one with night vision)
- wanted web interface for easy access
- had a led hat for status display
- battery is there so why not use it
- have some other cameras too (HDMI)

What this is not designed to do is to be the go to interface for working with the stereo microscope.

Looking through, live, beats using a latency plagued USB/HDMI/Whatever abstraction layer. This is an augmentation to assist and not take over. HDMI still beats the pi zero for live viewing by a long way.

2. Quick Start

For fully automated setup, run the one-click script:

```
./run-me-first.sh
```

It will: - Prompt for your Pi hostname/IP - Set up SSH access (generate keys and copy if needed) - Copy required files to Pi - Run comprehensive testing and container build

Manual steps if preferred:

1. Copy `prep-scripts` and `docker` directories to your Raspberry Pi:

```
scp -r prep-scripts docker user@pi-host:~/
```

1. Run the comprehensive test script to install dependencies and verify hardware:

```
ssh user@pi-host
cd prep-scripts
./comprehensive-test.sh
```

1. Start the microscope camera service:

```
cd ../docker/docker-microscope
sudo docker-compose up -d
```

1. Access the web interface at `https://<pi-ip>:8443`

See detailed setup and configuration instructions below.

3. Hardware used

- Raspberry Pi Zero (or Pi Zero 2)
- MacroSilicon USB Video device (ID 534d:2109)
- USB microscope with HDMI output (and USB)
- HDMI to USB capture adapter
- Pimoroni Unicorn HAT (4x8 LED matrix) - optional for status display
- UPS-lite v1.3

4. LED Status Display (Unicorn HAT)

The Pimoroni Unicorn HAT provides visual feedback through a 4x8 LED matrix:

4.1. Row Layout (bottom to top)

Row 0 (Bottom): System Status

```
Green = All systems operational (camera + USB working)
Partial Green = Some systems operational
Off = Systems not ready
```

Row 1: Streaming Status

```
Red = Video streaming active
Off = Not streaming
```

Rows 2-3 (Top): Capture Flash

```
Bright White Flash = Image captured successfully
Duration: 0.5 seconds
```

4.2. Startup Sequence

On container start, LEDs animate bottom to top in blue, then show current system status.

4.3. Error Pattern

If camera fails to initialize, LEDs flash alternating red rows.

5. System Setup

See the included documentation below for detailed setup instructions.

5.1. Pi Zero Status Check

Verify system information:

```
uname -a
```

Output shows ARMv6 architecture:

```
Linux piz-micoscope 6.12.47+rpt-rpi-v6 #1 Raspbian 1:6.12.47-1+rpt1~bookworm (2025-09-16) armv6l GNU/Linux
```

5.1.1. Hardware Tests (before going full in)

5.1.1.1. Why This

Verify Pi Zero microscope hardware before container deployment.

Tests run on bare Pi. Production uses Docker container.

Make sure stuff works. Debug hardware issues early.

NOTE	The comprehensive test script `comprehensive-test.sh` automates setup and testing. IF you are running that then after it you can skip this section unless you run aground.

5.1.1.2. Why This Way

Container uses apt packages (`python3-opencv`). Pre-compiled. Fast.

Pip packages (`opencv-python`) compile from source. Hours on Pi Zero (I gave up).

Solution: Use apt for opencv. Pip for small packages. Venv with system access.

5.1.1.3. Quick Start

Enable hardware:

```
$ sudo raspi-config
# Interface Options → I2C → Enable
# Interface Options → SPI → Enable
# Finish → Reboot
```

Copy prep-scripts and docker to Pi:

```
$ scp -r prep-scripts docker sean@piz-microscope.fritz.box:~/
```

On Pi install packages via apt:

```
$ sudo apt-get update
$ sudo apt-get install -y python3-opencv python3-numpy python3-dev
```

Create venv with system packages access:

```
$ cd ~/prep-scripts
$ python3 -m venv --system-site-packages ~/test-env
$ source ~/test-env/bin/activate
$ pip install --upgrade pip setuptools wheel
$ pip install -r requirements.txt
```

Requirements (requirements.txt):

```
smbus2
RPi.GPIO
unicornhat
```

Test hardware:

```
$ source ~/test-env/bin/activate

# LED HAT
$ sudo -E python3 prep-scripts/function-test-hat.py

# Battery
$ sudo -E python3 prep-scripts/function-test-battery.py

# Cameras (USB/CSI)
$ python3 prep-scripts/function-test-cameras.py --list  # List available cameras
$ python3 prep-scripts/function-test-cameras.py /dev/video0  # Test specific camera
```

Cleanup:

```
$ deactivate
$ rm -rf ~/test-env ~/prep-scripts
```

5.1.1.4. Hardware Check

List USB devices:

```
$ lsusb
Bus 001 Device 002: ID 534d:2109 MacroSilicon USB Video
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

List video devices:

```
$ v4l2-ctl --list-devices
unicam (platform:20801000.csi):
        /dev/video0
```

```
USB Video: USB Video (usb-20980000.usb-1):
        /dev/video1
        /dev/video2
```

Check I2C and SPI:

```
$ ls /dev/i2c-* /dev/spi*
/dev/i2c-1  /dev/spidev0.0  /dev/spidev0.1
```

5.1.1.5. Tests

5.1.1.5.1. LED HAT

Pimoroni Unicorn HAT 4x8. Uses SPI.

Script: `function-test-hat.py` - Color sequence test

Run:

```
$ sudo -E python3 function-test-hat.py
Unicorn HAT initialized
Test 1: Red row
Test 2: Green row
Test 3: Blue row
Test 4: Yellow row
Test 5: All white
Test complete
```

Expected: Rows light red, green, blue, yellow, then all white.

5.1.1.5.2. Battery

UPS-Lite v1.3. CW2015 fuel gauge. I2C 0x62.

Script: `function-test-battery.py` - Comprehensive battery testing

Device detection, register dump, voltage/capacity reading, status monitoring:

```
$ sudo -E python3 function-test-battery.py
UPS-Lite v1.3 Battery Monitor Test
I2C Address: 0x62

Test 1: I2C Device Detection
PASS: Device found

Test 2: Register Dump
Register dump (first 16 registers):
```

```
  Reg 0x00: 0x00 (0)
  ...

Test 3: Single Readings
Voltage: 4.20V
Capacity: 100%
Charging: False

Test 4: Status Summary
Status: FULL
Voltage: 4.20V
Capacity: 100%
Charging: False

Test 5: Continuous Monitoring (Ctrl+C to stop)
Status: FULL | Voltage: 4.20V | Capacity: 100%
...
```

5.1.1.5.3. Camera

Test USB cameras (MacroSilicon HDMI capture) and CSI cameras (onboard Pi camera).

Script: `test-camera.py` - Unified camera testing

List available cameras:

```
$ python3 test-camera.py --list
Available video devices:
/dev/video0: CSI camera (640x480 @ 30fps)
/dev/video1: USB camera (1280x720 @ 30fps)
```

Test all cameras:

```
$ python3 test-camera.py
Testing camera: /dev/video0
Resolution: 640x480
FPS: 30.0
SUCCESS: Captured test-video0.jpg

Testing camera: /dev/video1
Resolution: 1280x720
FPS: 30.0
SUCCESS: Captured test-video1.jpg
```

Test specific camera:

```
$ python3 test-camera.py /dev/video1
Testing camera: /dev/video1
```

```
Resolution: 1280x720
FPS: 30.0
SUCCESS: Captured test-video1.jpg
```

5.1.1.6. Troubleshooting

I2C missing:

```
$ ls /dev/i2c-*
$ sudo i2cdetect -y 1  # Battery at 0x62
```

SPI missing:

```
$ ls /dev/spi*
```

Import errors:

```
$ source ~/test-env/bin/activate
$ python3 -c "import cv2; print(cv2.__version__)"
```

GPIO busy:

```
$ sudo docker stop microscope-camera
```

5.1.1.7. Notes

`sudo -E` preserves venv.

`--system-site-packages` lets venv use apt opencv while keeping pip packages isolated.

Tests verify hardware only. Container handles production. Two different beasts. This is only foundation.

6. Access Web Interface

Open browser on same network:

- https://<pi-zero-ip>:8443

Stream endpoint:

- https://<pi-zero-ip>:8443/?action=stream

7. Configuration

Edit environment variables in `docker-compose.yml`*:*

```
environment:
  - VIDEO_DEVICE=/dev/video0    # USB device path
  - VIDEO_WIDTH=1280            # Resolution width
  - VIDEO_HEIGHT=720            # Resolution height
  - VIDEO_FPS=30                # Frames per second
  - WEB_PORT=8443               # Web interface port
  - ENABLE_LEDS=true            # Enable Unicorn HAT display
```

To disable LED display (if no Unicorn HAT):

```
  - ENABLE_LEDS=false
```

Common resolutions for microscope cameras:

- 640x480 @ 30fps (default, lowest latency)
- 1280x720 @ 30fps (HD quality)
- 1920x1080 @ 15fps (Full HD, lower fps)

8. Troubleshooting

8.1. ARMv6 vs ARMv7 Architecture Error

Pi Zero uses ARMv6. If you see platform mismatch warnings:

```
[Warning] The requested image's platform (linux/arm/v7) does not match
the detected host platform (linux/arm/v6)
```

Solution:

- Dockerfile now uses `balenalib/raspberry-pi-debian` which supports ARMv6.

8.2. Build Fails with Code 139

Segmentation fault during build. potentially caused by:

- Wrong base image architecture
- Out of memory during compilation

Solution:

- Use the updated Dockerfile with minimal dependencies and pre-built packages.

8.3. Device Not Found

Check USB device permissions:

```
ls -la /dev/video0
sudo chmod 666 /dev/video0
```

Add user to video group:

```
sudo usermod -aG video $USER
```

8.4. Unsupported Format

List supported formats:

```
v4l2-ctl --device=/dev/video0 --list-formats-ext
```

Adjust resolution in `docker-compose.yml` to match supported format.

8.5. Container Fails to Start

Check container logs:

```
docker-compose logs microscope
```

Rebuild container:

```
docker-compose down
docker-compose build --no-cache
docker-compose up -d
```

8.6. Container Permissions

Container runs with privileged mode (ran with) and has access to:

- `/dev/video0` - USB video device
- `/dev/gpiomem` - GPIO memory for LED control
- `/dev/spidev0.0` - SPI interface for Unicorn HAT
- `/sys` - System files (read-only) for hardware detection

These permissions allow the container to control the Unicorn HAT LEDs and access the USB camera.

8.7. Low Performance on Pi Zero (if you hit that)

Pi Zero (ARMv6) has limited CPU. Reduce resolution and framerate:

```
environment:
  - VIDEO_WIDTH=320
  - VIDEO_HEIGHT=240
  - VIDEO_FPS=15
```

8.8. LEDs Not Working

Check Unicorn HAT is properly connected and SPI is enabled:

```
ls -la /dev/spidev0.0
# Should show: crw-rw---- 1 root spi
```

Enable SPI if needed:

```
sudo raspi-config
# Interface Options -> SPI -> Enable
```

Check container has device access:

```
sudo docker exec microscope-camera ls -la /dev/spidev0.0 /dev/gpiomem
```

8.9. Multiple Video Devices

If multiple video devices exist, specify correct device:

```
v4l2-ctl --list-devices
```

Update `VIDEO_DEVICE` environment variable accordingly.

9. Pi Zero Microscope Setup

Fresh Raspberry Pi OS setup to production deployment.

9.1. Build/Rebuild

To rebuild from scratch:

1. Flash fresh Raspberry Pi OS

a. Configure WiFi and user (in the step above with RPI imager)

2. Follow this document
3. Deploy container

No manual package installation. No system package breaking. Reproducible deployment.

9.2. Initial Setup

Fresh Raspberry Pi OS Lite installed. WiFi configured. User `sean` created.

Hostname: `piz-microscope`

SSH from workstation:

```
ssh-copy-id sean@piz-microscope.fritz.box
ssh sean@piz-microscope.fritz.box
```

9.3. Enable Hardware Interfaces

Enable I2C and SPI for hardware access.

```
sudo raspi-config
```

Steps:

1. Interface Options → I2C → Enable
2. Interface Options → SPI → Enable
3. Finish → Reboot

Verify:

```
ls /dev/i2c-* /dev/spi*
```

Expected:

```
/dev/i2c-1
/dev/spidev0.0
/dev/spidev0.1
```

9.4. Install Docker

Docker provides container runtime. No application packages on host.

Install Docker:

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo usermod -aG docker sean
```

Logout and login to apply group.

Verify:

```
docker --version
```

9.5. Install Docker Compose

Install Docker Compose:

```
sudo apt update
sudo apt install docker-compose
```

Verify:

```
docker-compose --version
```

9.6. Environment Configuration

Configuration via `.env` file.

9.6.1. Setup

Copy example file:

```
cd docker
cp .env.example .env
```

Edit `.env` with your values.

9.6.2. Required Changes

Change authentication credentials:

```
WEB_USERNAME=your_username
WEB_PASSWORD=your_secure_password
```

9.6.3. Configuration Variables

9.6.3.1. Camera

```
VIDEO_DEVICE=/dev/video1
VIDEO_WIDTH=1280
VIDEO_HEIGHT=720
VIDEO_FPS=30
```

9.6.3.2. Web Server

```
WEB_PORT=8080
ENABLE_HTTPS=true
```

9.6.3.3. Authentication

```
WEB_USERNAME=admin
WEB_PASSWORD=your_secure_password_here
```

9.6.3.4. Hardware

```
ENABLE_LEDS=true
ENABLE_BATTERY_MONITOR=true
```

9.6.3.5. Battery

```
BATTERY_LOW_THRESHOLD=20
BATTERY_CRITICAL_THRESHOLD=10
```

9.6.4. Deployment

Docker Compose reads `.env` automatically.

Start service:

```
cd docker
docker-compose up -d
```

Variables passed to container. Defaults used if not set.

9.6.5. Security

`.env` file contains credentials. Never commit to git. Added to `.gitignore`.

Share `.env.example` instead. Each deployment creates own `.env`.

Backup `.env` securely. Rotate credentials regularly.

9.7. Deploy Application

Copy project to Pi:

```
scp -r docker sean@piz-microscope.fritz.box:~/
```

On Pi:

```
cd docker
sudo docker-compose up -d
```

Access web interface:

- https://piz-microscope.fritz.box:8080 (HTTPS with self-signed cert)
- https://piz-microscope.fritz.box:8443 (HTTPS primary)

Default credentials:

- Username: admin
- Password: changeme_generate_secure_password (change in .env)

9.8. Security Configuration

Security features implemented in microscope system.

9.8.1. Authentication

Basic HTTP authentication protects all endpoints. Health endpoint remains unauthenticated for monitoring.

9.8.1.1. Credentials

Configure in `.env` file:

```
WEB_USERNAME=admin
WEB_PASSWORD=your_secure_password_here
```

Copy example file:

```
cd docker
cp .env.example .env
```

Edit credentials in `.env`.

Default username: `admin` Default password: `changeme_generate_secure_password`

Change before deployment. Never commit `.env` to git.

9.8.2. HTTPS/TLS

Self-signed certificate generated during build. Valid for 365 days.

9.8.2.1. Certificate Details

Location: `/app/certs/` Files: `cert.pem`, `key.pem`

Certificate info: - Country: DE - Organization: VSaGCR - Common Name: microscope.local

9.8.2.2. Disable HTTPS

Set in `.env` file:

```
ENABLE_HTTPS=false
```

9.8.3. Path Traversal Protection

All file operations validate paths. Only files in `/app/captures` accessible.

Validation uses `filepath.resolve().is_relative_to()`. Invalid paths rejected with 400 error.

9.8.4. Privileged Mode Removed

Container runs without privileged mode. Specific device access only.

Devices mapped: - `/dev/video1` - Camera - `/dev/gpiomem` - GPIO - `/dev/spidev0.0` - SPI - `/dev/i2c-1` - I2C

9.8.5. Health Check

Path: `/health`

Unauthenticated endpoint for monitoring. Returns 200 if camera operational. Returns 503 if camera error.

```
curl https://localhost:8443/health
```

9.8.6. Protected Endpoints

All require authentication:

- `/` - Main interface
- `/stream` - Video feed

- `/capture` - Image capture
- `/status` - System status
- `/captures` - List images
- `/captures/<filename>` - Get image
- `/batch/download` - Batch download
- `/batch/delete` - Batch delete

9.8.7. Recommendations

Store credentials in `.env` file. File excluded from git via `.gitignore`.

Generate strong password:

```
python3 -c "import secrets; print(secrets.token_urlsafe(32))"
```

Update `.env` with generated password.

Use reverse proxy for production. Terminate TLS with valid certificate. Add rate limiting.

9.8.8. File Security

`.env` contains sensitive credentials. Never commit to version control. Added to `.gitignore` automatically.

Share `.env.example` instead. Each deployment creates own `.env`.

Backup `.env` file securely. Rotate credentials regularly.

9.8.9. Known Limitations

Self-signed certificate triggers browser warnings. Accept certificate or add to trusted store.

No password complexity requirements. No account lockout mechanism. No audit logging.

Plan proper authentication system for production.

9.9. Verify Hardware

Container runs tests on startup.

Check logs:

```
sudo docker logs microscope-camera
```

Expected:

```
Unicorn HAT initialized successfully
CW2015 battery monitor initialized
Camera /dev/video1 opened
```

9.10. Host State

After setup, host has:

- Docker installed
- I2C enabled
- SPI enabled
- User in docker group

Host does NOT have:

- Python packages
- Application code (only in container)
- Manual dependencies

Clean host. Container has everything.

9.11. Planned Features Not Yet Implemented

Features designed in architecture but not implemented. See ADRs for detailed specifications:

- ADR-003: LED Event Queue Pattern
- ADR-004: Battery Monitoring Architecture
- ADR-005: Multi-Camera Source Management

9.11.1. Current Implementation Status

- Battery monitoring module exists but not integrated
- Event queue architecture designed but not implemented
- Multi-camera support planned but single camera only
- LED admin interface designed but endpoint missing
- Stream deck UI mockup created but not built
- Health endpoint referenced but not implemented

9.11.2. Implementation Priority

1. Fix security issues first ✔
2. Add health endpoint for monitoring

3. Integrate battery monitoring
4. Implement event queue pattern
5. Add multi-camera support
6. Build admin interface
7. Create stream deck UI

9.12. Security Fixes and Improvements Applied

Security vulnerabilities fixed. All changes documented.

9.12.1. Critical Fixes

9.12.1.1. Path Traversal Vulnerability

Issue: `batch_delete()` and file operations accepted user filenames without validation.

Fix: All file operations now validate paths using `filepath.resolve().is_relative_to()`.

Files: * `docker/docker-microscope/web_server.py`

Endpoints protected: * `/captures/<filename>` * `/batch/download` * `/batch/delete`

9.12.1.2. Authentication Added

Issue: No authentication on any endpoint.

Fix: Basic HTTP authentication on all endpoints except health check.

Configuration: * Username: `WEB_USERNAME` environment variable * Password: `WEB_PASSWORD` environment variable

Protected endpoints: * `/` - Main interface * `/stream` - Video feed * `/capture` - Image capture * `/status` - System status * `/captures` - List images * All batch operations

9.12.1.3. HTTPS Support

Issue: Unencrypted HTTP traffic.

Fix: Self-signed certificate generated during build.

Files modified: * `docker/docker-microscope/Dockerfile` - Certificate generation * `docker/docker-microscope/web_server.py` - SSL context handling

Configuration: * `ENABLE_HTTPS` environment variable * Certificate path: `/app/certs/`

9.12.1.4. Privileged Mode Removed

Issue: Container ran with `privileged: true`.

Fix: Removed privileged mode from compose file.

Files: * `docker/docker-compose.yml`

Container now uses specific device mappings only.

9.12.2. Configuration Changes

9.12.2.1. Environment Variables Centralized

All configuration moved to `docker-compose.yml`.

Variables added: * `WEB_USERNAME` - Default: admin * `WEB_PASSWORD` - Default: changeme_generate_secure_password * `ENABLE_HTTPS` - Default: true * `BATTERY_LOW_THRESHOLD` - Default: 20 * `BATTERY_CRITICAL_THRESHOLD` - Default: 10

9.12.2.2. Health Check Added

Endpoint: `/health`

Unauthenticated for monitoring systems. Returns 200 if camera operational. Returns 503 if camera error.

Docker healthcheck configured in compose file.

9.12.2.3. Ports Updated

Exposed ports: * 8080 - HTTP/HTTPS web interface * 8443 - Additional HTTPS port

9.12.3. Documentation Created

9.12.3.1. New Files

`docker/PLANNED_FEATURES.adoc` * Lists unimplemented features from architecture * Battery monitoring integration * Event queue system * Multi-camera switching * LED admin interface * Stream deck interface

`docker/SECURITY.adoc` * Security configuration guide * Authentication setup * HTTPS configuration * Path protection details * Recommendations

`adr/adr-002-security-model.adoc` * Architecture decision record * Security model rationale * Implementation details * Trade-offs documented

9.12.3.2. Updated Files

`docker/README.adoc` * Updated deployment instructions * Added authentication details * Fixed paths after restructure

`include-setup.adoc` * Fixed file structure documentation * Removed references to nonexistent examples directory * Content consolidated into main `README.asciidoc`

`prep-scripts/README.adoc` * Fixed test deployment paths

`adr/README.adoc` * Added ADR-002 include

9.12.4. Files Modified

9.12.4.1. Core Application

`docker/docker-microscope/web_server.py` * Added authentication system * Added HTTPS support * Fixed path traversal vulnerabilities * Added health endpoint * Environment-based configuration

`docker/docker-microscope/Dockerfile` * Added openssl package * Generate self-signed certificate * Expose port 8443

`docker/docker-microscope/docker_entrypoint.sh` * Use VIDEO_DEVICE from environment consistently * Remove hardcoded /dev/video0 references

9.12.5. Deployment Changes

Default password must be changed. Generate secure password before deployment.

Access interface at: * https://hostname:8080 (HTTPS) * https://hostname:8443 (HTTPS primary)

Browser shows certificate warning. Accept self-signed certificate or add to trusted store.

9.12.6. Security Recommendations

Change default password in compose file. Use reverse proxy for production. Terminate TLS with valid certificate. Add rate limiting at proxy level.

Generate password:

```
python3 -c "import secrets; print(secrets.token_urlsafe(32))"
```

9.12.7. Testing

Test authentication:

```
curl -u admin:password https://hostname:8080/
```

Test health endpoint:

```
curl https://hostname:8443/health
```

Test path traversal protection:

```
curl -u admin:password https://hostname:8080/captures/../etc/passwd
# Should return 400 Bad Request
```

9.12.8. Next Steps

Security fixes complete. System ready for deployment.

Consider: * Deploy and test authentication * Generate production password * Set up reverse proxy * Monitor health endpoint * Review security documentation

Appendix A: References

- mjpg-streamer GitHub
- Video4Linux2 API

Appendix B: images

B.1. Infrastructure Diagrams

PlantUML diagrams for the infrastructure management system.

B.1.1. System Architecture Diagrams

B.1.1.1. VSaGCR Microscopy System Architecture

[architecture] | *architecture.png*

Figure 1. architecture infrastructure

Appendix C: Architecture Decision Records

C.1. What is an ADR

Architecture Decision Record captures significant design choices.

TOGAF defines ADRs as:

- Document key architectural decisions
- Record context and rationale
- Track consequences
- Enable future understanding

C.2. Format

- Title: Decision identifier
- Status: Proposed, Accepted, Deprecated, Superseded
- Context: The problem
- Decision: The choice made
- Consequences: Impact and trade-offs

C.3. Records

C.3.1. ADR-001: Container-First Architecture

Status: Accepted

Date: 2025-12-14

C.3.1.1. Context

Pi Zero microscope project requires portability across hardware variants (Zero, Zero 2, etc.).

System dependencies create deployment friction:

- Python packages scattered on host
- Manual I2C/SPI configuration
- Version conflicts
- Rebuild complexity

Testing on bare metal pollutes host system. Direct pip install with `--break-system-packages` breaks system integrity.

C.3.1.2. Decision

All application code runs in Docker containers.

Host system configuration limited to:

- Docker installation
- I2C enabled (`dtparam=i2c_arm=on`)
- SPI enabled (`dtparam=spi=on`)
- Device permissions (i2c, spi, video groups)

No Python packages installed on host. No application dependencies on host.

Container provides:

- Complete runtime environment

- All Python dependencies
- Reproducible builds
- Hardware device access via volume mounts

Testing on bare Pi is interim only. All Pi testing is throwaway. Pi will be rebuilt from clean image.

C.3.1.3. Consequences

C.3.1.3.1. Positive

- Hardware portability guaranteed
- Clean host system
- Reproducible deployments
- Version control of full stack
- Easy migration to Zero 2 or other hardware
- No manual dependency management
- Container is deployment artifact

C.3.1.3.2. Negative

- Docker required on Pi
- Container build time overhead
- Privileged mode required for hardware access
- Docker is an abstraction

C.3.1.3.3. Implementation

- Dockerfile defines complete environment
- docker-compose.yml maps devices and volumes
- Deployment script handles container lifecycle
- Host setup documented in single guide
- No host package pollution

C.3.2. ADR-008: AsciiDoc Conditional Settings for Multi-Format Publishing

Status: Accepted

Date: 2025-12-15

C.3.2.1. Context

Documentation uses AsciiDoc format for multiple output targets: GitHub rendering, PDF compilation, and central documentation site.

GitHub renders AsciiDoc directly but does not process include directives. This limits settings to

inline declarations.

Compiled AsciiDoc processes includes, allowing settings in separate files for PDF generation.

Images require conditional imagesdir setting for book format to display correctly in compiled docs while maintaining GitHub compatibility.

Notes and other blocks need inline settings for GitHub visibility, but can use included settings for compiled formats.

C.3.2.2. Decision

Use conditional blocks for imagesdir management:

- Set imagesdir to local images directory when flag-book is defined
- Store original imagesdir before changing
- Reset imagesdir after image blocks

Keep settings for notes and blocks inline in main documents for GitHub compatibility.

Place PDF-specific settings in included files for compiled documentation.

C.3.2.3. Consequences

C.3.2.3.1. Positive

- Images display correctly in GitHub and compiled formats
- Notes and blocks render properly on GitHub
- PDF settings remain organized in separate files
- Maintains clean separation between formats

C.3.2.3.2. Negative

- Adds complexity to document structure
- Requires careful management of conditional blocks
- Inline settings reduce modularity for GitHub-focused docs

C.3.3. ADR-002: Security Model and Authentication

Status: Accepted

Date: 2025-12-15

C.3.3.1. Context

System provides web interface for microscope camera control. Might handle sensitive microscope images and captures. Network exposure requires protection against unauthorized access.

Security requirements:

- Prevent unauthorized access to camera controls
- Protect captured images from unauthorized viewing or deletion
- Encrypt network traffic to prevent interception
- Prevent file system attacks through path manipulation
- Limit container privileges to reduce attack surface
- Enable monitoring without compromising security

C.3.3.2. Decision

Implement layered security approach:

C.3.3.2.1. Authentication

Basic HTTP authentication on all endpoints except health check. Credentials configured via environment variables (in MVP). Health endpoint remains unauthenticated for monitoring systems.

C.3.3.2.2. HTTPS/TLS

- Self-signed certificate generated during container build.
- HTTPS enabled by default.
- HTTP fallback available via configuration.
- Possible next step with let's encrypt and forward proxy (as opposed to reverse).
    - Forward proxy allows encapsulated environment with own domain not exposed to internet.

C.3.3.2.3. Path Traversal Protection

All file operations validate resolved paths. Python `pathlib.resolve().is_relative_to()` prevents directory traversal. Only files within `/app/captures` accessible.

C.3.3.2.4. Least Privilege

- Remove privileged container mode (needs to be tested and might return).
- Map specific devices only.
- No unnecessary host access.

C.3.3.3. Consequences

C.3.3.3.1. Positive

- Unauthorized access prevented
- Network traffic encrypted
- File system attacks blocked
- Reduced attack surface
- Health monitoring still functional

- Simple credential management
  - MVP for one user

C.3.3.3.2. Negative

- Self-signed certificate triggers browser warnings
- Basic auth not suitable for high-security environments
- Credentials stored in compose file
- No multi-user support
- No audit logging
- No rate limiting

C.3.3.3.3. Mitigation

- Document security limitations.
- Recommend reverse/forward proxy for production.
- Provide credential generation guidance.
- Plan proper authentication for production use.

C.3.3.3.4. Implementation

Environment variables:

- `WEB_USERNAME` - Authentication username
- `WEB_PASSWORD` - Authentication password
- `ENABLE_HTTPS` - Enable HTTPS mode

Certificate location: `/app/certs/`

Protected endpoints: All except `/health`

C.3.3.4. Future Improvements

- OAuth2/OIDC integration
- Certificate management via Let's Encrypt
- Audit logging
- Rate limiting
- Session management
- Multi-user support
- Role-based access control

C.3.4. ADR-003: LED Event Queue Pattern

Status: Proposed

Date: 2025-12-15

C.3.4.1. Context

LED control needed from multiple sources: * Web server on image capture * Battery monitor for status * Camera service for streaming indicator * System health checks * Admin interface for testing

Direct function calls create tight coupling. LED operations block calling code. Multiple sources compete for LED control.

Event queue pattern designed in architecture docs. Worker thread processes events async. Decouples producers from LED hardware.

Current code uses direct LED function calls.

C.3.4.2. Decision

Event queue pattern is design goal. Direct calls acceptable for current simple use case.

Implement event queue when: * Battery monitoring integrated * Multi-camera switching added * LED admin interface created * Animation conflicts occur

C.3.4.3. Consequences

C.3.4.3.1. Positive - Direct Calls (Current)

- Simple implementation
- Fewer moving parts
- Easy to debug
- No queue overhead
- Adequate for single camera

C.3.4.3.2. Negative - Direct Calls (Current)

- Tight coupling between modules
- LED calls may block
- Cannot prioritize events
- Hard to add new LED features
- Testing requires hardware

C.3.4.3.3. Positive - Event Queue (Future)

- Async non-blocking operations
- Multiple producers supported
- Priority queue possible
- Centralized LED logic

- Easy testing via event injection
- Animations don't conflict

C.3.4.3.4. Negative - Event Queue (Future)

- More complex
- Additional thread management
- Queue full scenarios
- Event ordering considerations

C.3.4.4. Implementation Plan

Phase 1 (current):

- Direct LED calls in web server
- Simple flash on capture
- System status indication

Phase 2 (when battery added):

- Introduce event queue
- Worker thread processes events
- Battery publishes status events
- Web server publishes capture events

Phase 3 (multi-camera):

- Streaming events per camera
- Source indicator on LEDs
- Camera switching animations

C.3.4.5. Technical Details

C.3.4.5.1. Event Types

```
EventType.SYSTEM_STATUS  -> Row 0: System health (green)
EventType.STREAMING      -> Row 1: Streaming active (red)
EventType.CAPTURE        -> Row 2: Capture flash (white)
EventType.BATTERY        -> Row 3: Battery gauge (8 LEDs)
EventType.ANIMATION      -> Startup/shutdown/rainbow/pulse
EventType.TEST           -> Test patterns for admin page
EventType.SHUTDOWN       -> Low battery warning (blink red)
```

C.3.4.5.2. LED Row Assignment

- Row 3 (Top): Battery Level (8-LED gauge)
  - Green=full, Yellow=med, Orange=low, Red=critical

    - Cyan=charging
- Row 2: Capture Flash (white, 2s)
- Row 1: Streaming Status (red when active)
    - Source indicator possible
- Row 0 (Bottom): System Status (green when OK)
    - Left half=camera, Right half=system

C.3.4.5.3. Benefits

- Async/non-blocking - no delays in main app
- Multiple sources can send events
- Centralized LED control logic
- Easy to add battery, temp, etc.
- Test mode can inject events
- Python stdlib only (no MQTT overhead)

C.3.4.6. LED Test Interface

Admin page at `/admin/leds` *with:*

- Event injection controls
- Animation triggers
- Current state display
- Test patterns
- Battery simulation

C.3.4.7. References

Pattern documented but deferred until needed.

C.3.5. ADR-004: Battery Monitoring Architecture

Status: Proposed

Date: 2025-12-15

C.3.5.1. Context

Pi Zero runs on UPS PIco battery HAT. System must protect against data loss on power failure.

Battery status needed for:

- Graceful shutdown below threshold
- LED battery gauge display
- System health monitoring

C.3.5.2. Decision

Integrate battery monitoring when LED event queue implemented. Monitor UPS PIco via I2C at address 0x69. Implement three-tier threshold system.

C.3.5.2.1. Thresholds

Low warning: 20% * Yellow/orange LEDs * Log warning

Critical: 10% * Red blinking LEDs * Initiate shutdown countdown

Emergency: <5% * Immediate shutdown * Flush buffers * Safe camera stop

C.3.5.2.2. Shutdown Sequence

1. Detect battery below critical threshold
2. Send shutdown event to LED (red blink 5x)
3. Wait 3 seconds for visual warning
4. Stop camera gracefully
5. Flush capture buffer
6. Execute system shutdown

C.3.5.3. Consequences

C.3.5.3.1. Positive

- Data loss prevented
- Clear battery warnings
- Graceful shutdown
- User sees battery status
- System protected

C.3.5.3.2. Negative

- Additional monitoring thread
- Shutdown may interrupt captures
- Battery HAT required for full function

C.3.5.3.3. Implementation

Environment variables: * `ENABLE_BATTERY_MONITOR` - Enable/disable * `BATTERY_LOW_THRESHOLD` - Default 20 * `BATTERY_CRITICAL_THRESHOLD` - Default 10

Battery monitor runs in separate thread. Polls every 5 seconds. Publishes events to LED queue.

C.3.5.4. Integration Points

Battery service:

- `battery_status.py` - Already exists
- I2C communication with UPS-lite v1.3
- Voltage and capacity reading
- Charging status detection

LED display:

- Row 3 (top) - 8-LED battery gauge
- Green when full
- Yellow when low
- Red when critical
- Cyan when charging

Web interface:

- Battery status in system info
- Charge percentage displayed
- Charging indicator

Health endpoint:

- Include battery status
- Warning if below threshold

C.3.5.5. Safety Considerations

Shutdown initiated by container. Requires host shutdown capability. May need privileged mode or specific capability.

Alternative: Notify external monitor. External system handles shutdown. Container only reports status.

C.3.5.6. Current Status

Module exists: `docker/docker-microscope/battery_status.py`

Not imported in web server. Not integrated with LED system. Environment variables configured in compose.

Waiting for LED event queue implementation.

C.3.6. ADR-005: Multi-Camera Source Management

Status: Proposed

Date: 2025-12-15

C.3.6.1. Context

System designed for three (or more) camera sources:

- HDMI capture via MacroSilicon USB adapter
  - For now… next step is to use that for an additional camera
- Direct USB microscope camera
  - currently the usb capture device as no HUB used
- Raspberry Pi Camera Module (CSI) (with night vision)

Current

- Current implementation supports single camera only.
- Camera device hardcoded in configuration.
- No source detection or switching.

Goal

- Architecture documents stream deck interface.
- User can switch between sources.
- Each source requires different configuration.

Potentail further cameras:

- Additional USB microscopes via powered hub
  - eg thermal camera

C.3.6.2. Decision

Defer multi-camera until single camera stable.

When implemented:

- Auto-detect available cameras on startup
  - should do now even for single camera
- Maintain separate camera service per source
- Switch active stream via API endpoint
- Show source thumbnails in UI

C.3.6.2.1. Detection Strategy

Scan video devices on startup:

- `/dev/video0` through `/dev/video4`
- Query capabilities with v4l2
- Identify device by USB ID or driver
- Store available sources

C.3.6.2.2. Switching Approaches

Option 1: V4L2 switching (simpler)

- Single Flask stream endpoint
- Switch camera service backend
- One stream at a time
- Lower bandwidth

Option 2: Multiple streams (complex)

- Separate endpoint per camera
- Client-side switching
- Simultaneous preview possible
- Higher bandwidth

Decision:

- Start with Option 1.
- Upgrade to Option 2 if preview needed.

C.3.6.3. Consequences

C.3.6.3.1. Positive - Deferred

- Simpler current implementation
- Single camera well-tested
- Easier deployment
- Lower resource usage

C.3.6.3.2. Negative - Deferred

- Cannot use multiple microscopes
- Manual config to switch sources
- No camera comparison
- Limited use cases

C.3.6.3.3. Positive - When Implemented

- Flexible microscope setup
- Easy source comparison
- Support different microscopes
- Stream deck interface

C.3.6.3.4. Negative - When Implemented

- More complex device management

- Camera initialization overhead
- Source switching latency
- Higher resource usage
    - Could well be beyond pi zero

C.3.6.4. Implementation Plan

Phase 1 (current):

- Single camera via VIDEO_DEVICE env var
- Manual configuration
- Restart to switch

Phase 2 (detection):

- Scan available cameras
- Report in /status endpoint
- UI shows available sources

Phase 3 (switching):

- `/switch/<source>` endpoint
- Stop current camera
- Start new camera
- Update LED streaming indicator

Phase 4 (stream deck):

- Thumbnail previews
- Click to switch
- Source status indicators

NOTE	slowly slowly catchy monkey.

C.3.6.5. Stream Deck Interface

C.3.6.5.1. Features

- Preview tiles for each source (thumbnail grid)
- Click to switch active source
- Capture from any source
- Download captured images
- Source status indicators

C.3.6.5.2. UI Layout

```
┌──────────────────────────────────────────────────────────────────────────────┐
│ VSaGCR - Microscopy Interface                                                │
├──────────────────────────────────────────────────────────────────────────────┤
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐                        │
│  │ HDMI  │  │ USB  │  │ Pi  │ Sources │
│  │ ▯    │  │      │  │ Cam  │         │
│  └──────────────┘  └──────────────┘  └──────────────┘                        │
├──────────────────────────────────────────────────────────────────────────────┤
│ [Main Preview - Active Source]                                               │
│                                                                              │
├──────────────────────────────────────────────────────────────────────────────┤
│ [Capture] [Record] [Switch]                                                  │
│ Battery: ▯▯▯▯▯▯▯▯ 85%                                                        │
└──────────────────────────────────────────────────────────────────────────────┘
```

C.3.6.6. Configuration

Environment variables per source:

```
VIDEO_DEVICE_HDMI=/dev/video0
VIDEO_DEVICE_USB=/dev/video1
VIDEO_DEVICE_CSI=/dev/video2
```

Auto-detection overrides if devices found. Maybe even just forgo config entirely if dynamic is good enough.

C.3.6.7. LED Integration

Row 1 streaming indicator shows active source: * Left side: HDMI * Center: USB * Right: CSI

Animations on source switch.

C.3.6.8. Current Status

Single camera only. Configured via VIDEO_DEVICE. Multi-camera documented in architecture.

Waiting for: * Production deployment experience * User feedback on single camera (me) * LED event queue implementation

C.3.7. ADR-006: Directory Structure

Status: Accepted

Date: 2025-12-15

C.3.7.1. Context

- Need a clear directory structure for project.
- Want to have this as a well documented project.
- People should find files easily (others and me).

C.3.7.2. Decision

```
/docker/
  /docker-microscope/     # Container application
  docker-compose.yml      # Service definition
  README.adoc             # Docker deployment docs

/adr/                     # Architecture decisions

/prep-scripts/           # Hardware prep and test scripts

/images/                  # Diagrams and screenshots

Root level files:
  README.asciidoc         # Main project docs (consolidated)
```

C.3.7.3. Consequences

C.3.7.3.1. Positive

- Clear separation of concerns
- Docker files grouped logically
- Build context simplified
- Easier to find components

C.3.7.4. Rationale

`/docker` groups all container-related files. Sub-directory per container project. Multiple containers possible in future.

Example expansion:

```
/docker/
  /docker-microscope/
  /docker-analysis/       # Future: image processing
  /docker-storage/        # Future: backup service
  docker-compose.yml      # Orchestrates all
```

C.3.8. ADR-007: Privileged Container Mode Removal

Status: Accepted

Date: 2025-12-15

C.3.8.1. Context

Initial container configuration used `privileged: true`.

Original justification:

- Hardware access for GPIO
- SPI interface for LEDs
- I2C for battery monitor
- Video device access

Privileged mode grants full host access.

Container can:

- Access all devices
- Modify kernel settings
- Escape container isolation
- Bypass security controls

Problem?

- Violates least-privilege principle.
    - But does it matter here as the things is dedicated anyhow

C.3.8.2. Decision

Remove privileged mode completely. Use specific device mappings only.

Devices explicitly mapped:

- `/dev/video1` - Camera capture
- `/dev/gpiomem` - GPIO memory (LEDs)
- `/dev/spidev0.0` - SPI interface (Unicorn HAT)
- `/dev/i2c-1` - I2C bus (battery monitor)
- `/dev/vchiq` - VideoCore interface
- `/dev/vcsm-cma` - VideoCore shared memory

Read-only volumes:

- `/sys:ro` - System information
- `/run/udev:ro` - Device events

C.3.8.3. Consequences

C.3.8.3.1. Positive

- Reduced attack surface
- Container isolation maintained
- Host protection enhanced
  - not as important right now but maybe in future
- Follows security best practices
- Complies with security standards

C.3.8.3.2. Negative

- More verbose device configuration
- Must update for new hardware
- Device permissions still required on host
- Some features may need capabilities

C.3.8.4. Host Configuration Required

Devices must exist with proper permissions:

```
# Enable I2C and SPI
sudo raspi-config
# Interface Options → I2C → Enable
# Interface Options → SPI → Enable

# User in required groups
sudo usermod -aG video,spi,i2c,gpio $USER
```

Device permissions checked on startup. Container should fail fast if devices inaccessible.

C.3.8.5. Capabilities Considered

Alternative to privileged mode: Linux capabilities.

Potential:

- `CAP_SYS_ADMIN` - Not required
- `CAP_SYS_RAWIO` - Not required
- `CAP_NET_ADMIN` - Not required

Device access sufficient.

If future features need capabilities:

```
cap_add:
  - SPECIFIC_CAPABILITY
```

Document in ADR with justification.

C.3.8.6. Shutdown Capability

Battery monitoring may need system shutdown. Options:

1. External shutdown monitor (chosen)
    - Container reports battery status
    - External service handles shutdown
    - No container privileges needed
2. Shutdown capability
    - Add `CAP_SYS_BOOT`
    - Container can shutdown host
    - Security consideration

Current: Battery monitoring deferred.

C.3.8.7. Testing Results

Container tested without privileged mode: * Camera access: Works * LED control: Works * SPI communication: Works * I2C access: Works (when enabled - Duh) * Video capture: Works

All functionality maintained. No privileged mode required.