Astra Raspberry Pi Setup Guide

1) Hardware

- Raspberry Pi Zero 2 W
- microSD card, 4 GB minimum, 8 GB or larger recommended
- Raspberry Pi Camera
- Power supply

2) Flash Raspberry Pi OS Lite

Open Raspberry Pi Imager and configure it as follows:

- **Device:** Raspberry Pi Zero 2 W
- **Operating System:** Raspberry Pi OS Lite (32-bit)
- **Storage:** Select your microSD card

Before clicking **Write**, open the advanced options (⚙ or **Edit Settings**) and configure:

- Hostname: `astra`
- Enable SSH
- Set a username and password
- Configure your Wi-Fi SSID and password
- Set your locale, keyboard layout, and timezone

Click **Write**, wait for the flashing and verification process to finish, then safely eject the card.

Insert the card into the Raspberry Pi, power it on, and wait about 1–2 minutes for the first boot.

You are now ready to continue with the setup below.

3) First boot and update

SSH into the Pi from your pc:

```
ssh <username>@astra.local
```

And run:

```
sudo apt update
sudo apt full-upgrade -y
sudo reboot
```

4) Install packages

Run:

```
sudo apt install python3-flask python3-waitress avahi-daemon
```

5) Create the project folder

```
mkdir -p ~/astra
cd ~/astra
```

6) Create `app.py`

Create the file with your editor:

```
nano ~/astra/app.py
```

Paste the following code:

```
import atexit
import subprocess
import threading
import time
from datetime import datetime
from io import BytesIO

from flask import Flask, Response, redirect, render_template_string, request,
send_file, url_for
from waitress import serve

app = Flask(__name__)

WIDTH = 1920
HEIGHT = 1080
FPS = 15

state_lock = threading.Lock()
frame_cond = threading.Condition()

# Default settings (Moon mode)
DEFAULTS = {
    "hflip": False,
    "vflip": True,
    "exposure_us": 1000,
    "gain": 1.0,
}

camera_settings = DEFAULTS.copy()

latest_frame = None
latest_frame_id = 0

camera_proc = None
stop_event = threading.Event()
restart_event = threading.Event()

def clamp_int(value, low, high, default):
    try:
        value = int(value)
    except (TypeError, ValueError):
        return default
    return max(low, min(high, value))

def clamp_float(value, low, high, default):
    try:
        value = float(value)
    except (TypeError, ValueError):
        return default
```

```
    return max(low, min(high, value))

def build_camera_cmd():
    with state_lock:
        hflip = camera_settings["hflip"]
        vflip = camera_settings["vflip"]
        exposure_us = camera_settings["exposure_us"]
        gain = camera_settings["gain"]

    cmd = [
        "rpicam-vid",
        "-n",
        "-t", "0",
        "--width", str(WIDTH),
        "--height", str(HEIGHT),
        "--framerate", str(FPS),
        "--codec", "mjpeg",
        "--gain", f"{gain}",
        "--shutter", str(exposure_us),
        "-o", "-",
    ]

    if hflip:
        cmd.append("--hflip")
    if vflip:
        cmd.append("--vflip")

    return cmd

def camera_worker():
    global latest_frame, latest_frame_id, camera_proc

    while not stop_event.is_set():
        cmd = build_camera_cmd()

        camera_proc = subprocess.Popen(
            cmd,
            stdout=subprocess.PIPE,
            stderr=subprocess.DEVNULL,
            bufsize=0,
        )

        buffer = bytearray()

        try:
            while not stop_event.is_set() and not restart_event.is_set():
                chunk = camera_proc.stdout.read(4096)
                if not chunk:
                    break

                buffer.extend(chunk)

                while True:
                    start = buffer.find(b"\xff\xd8")
                    if start == -1:
                        if len(buffer) > 2_000_000:
                            del buffer[:-2_000_000]
                        break

                    end = buffer.find(b"\xff\xd9", start + 2)
                    if end == -1:
                        if start > 0:
                            del buffer[:start]
                        break
```

```
                    frame = bytes(buffer[start:end + 2])
                    del buffer[:end + 2]

                    with frame_cond:
                        latest_frame = frame
                        latest_frame_id += 1
                        frame_cond.notify_all()

        finally:
            try:
                if camera_proc and camera_proc.poll() is None:
                    camera_proc.terminate()
                    camera_proc.wait(timeout=2)
            except Exception:
                pass

            restart_event.clear()
            time.sleep(0.2)

def mjpeg_generator():
    boundary = b"--frame\r\nContent-Type: image/jpeg\r\n\r\n"
    last_seen_id = -1

    while not stop_event.is_set():
        with frame_cond:
            frame_cond.wait_for(
                lambda: latest_frame_id != last_seen_id or stop_event.is_set(),
                timeout=5,
            )

            if stop_event.is_set():
                break

            if latest_frame is None:
                continue

            frame = latest_frame
            last_seen_id = latest_frame_id

        yield boundary + frame + b"\r\n"

INDEX_HTML = """
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Astra</title>
<style>
    * {
        box-sizing: border-box;
    }

    body {
        margin: 0;
        background: #141414;
        color: #e5e5e5;
        font-family: system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI",
Roboto, sans-serif;
    }

    header {
        height: 56px;
```

```
        display: flex;
        align-items: center;
        padding: 0 20px;
        background: #1b1b1b;
        border-bottom: 1px solid #2f2f2f;
        font-size: 22px;
        font-weight: 600;
        letter-spacing: 1px;
    }

    .container {
        max-width: 1100px;
        margin: 24px auto;
        padding: 0 20px 24px 20px;
    }

    .viewer {
        background: #000;
        border: 1px solid #333;
        overflow: hidden;
        border-radius: 8px;
    }

    .viewer img {
        display: block;
        width: 100%;
        height: auto;
        transform-origin: center center;
        transition: transform 0.15s ease-out;
    }

    .panel {
        margin-top: 18px;
        padding: 16px 0 0 0;
    }

    .controls {
        display: grid;
        grid-template-columns: repeat(2, minmax(240px, 1fr));
        gap: 18px 28px;
        align-items: end;
    }

    .control-group {
        min-width: 0;
    }

    .control-label {
        display: flex;
        justify-content: space-between;
        align-items: baseline;
        margin-bottom: 8px;
        font-size: 14px;
        color: #d8d8d8;
    }

    .control-value {
        color: #a9a9a9;
        font-variant-numeric: tabular-nums;
    }

    .slider {
        width: 100%;
        margin: 0;
```

```
}

.toggle-row {
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 10px 0;
    font-size: 15px;
    color: #e5e5e5;
}

.switch {
    position: relative;
    display: inline-block;
    width: 52px;
    height: 30px;
    flex: 0 0 auto;
}

.switch input {
    opacity: 0;
    width: 0;
    height: 0;
}

.slider-ui {
    position: absolute;
    cursor: pointer;
    inset: 0;
    background: #444;
    transition: 0.2s;
    border-radius: 999px;
    border: 1px solid #555;
}

.slider-ui:before {
    position: absolute;
    content: "";
    height: 22px;
    width: 22px;
    left: 3px;
    top: 3px;
    background: #ddd;
    transition: 0.2s;
    border-radius: 50%;
}

.switch input:checked + .slider-ui {
    background: #2f6feb;
    border-color: #2f6feb;
}

.switch input:checked + .slider-ui:before {
    transform: translateX(22px);
}

.modes-bar {
    display: flex;
    gap: 10px;
    margin-bottom: 24px;
    flex-wrap: wrap;
    padding-bottom: 16px;
    border-bottom: 1px solid #333;
}
```

```
    .buttons {
        display: flex;
        gap: 12px;
        align-items: center;
        margin-top: 24px;
        flex-wrap: wrap;
    }

    button, a.capture {
        appearance: none;
        border: 1px solid #444;
        border-radius: 8px;
        padding: 10px 18px;
        font-size: 14px;
        font-weight: 500;
        cursor: pointer;
        color: white;
        background: #2d2d2d;
        text-decoration: none;
    }

    button:hover, a.capture:hover {
        background: #3b3b3b;
    }

    .btn-apply { background: #2f6feb; border-color: #2f6feb; }
    .btn-apply:hover { background: #2458c2; }

    .btn-capture { border-color: #059669; background: #059669; }
    .btn-capture:hover { background: #047857; }

    .btn-reset { margin-left: auto; border-color: #7f1d1d; background: #450a0a;
color: #fecaca; }
    .btn-reset:hover { background: #7f1d1d; }

    .footer {
        margin-top: 18px;
        color: #999;
        font-size: 13px;
        font-variant-numeric: tabular-nums;
    }

    @media (max-width: 720px) {
        .controls {
            grid-template-columns: 1fr;
        }
        .btn-reset { margin-left: 0; width: 100%; text-align: center; }
    }
</style>
</head>
<body>
<header>ASTRA</header>

<div class="container">
    <div class="viewer">
        <img id="liveStream" src="{{ url_for('video_feed') }}" alt="Astra live
feed">
    </div>

    <form class="panel" method="post" action="{{ url_for('settings') }}">

        <div class="modes-bar">
            <span style="display:flex; align-items:center; margin-right:8px;
```

```
color:#aaa; font-size:14px;">Presets:</span>
            <button type="submit" formaction="{{ url_for('preset',
mode='moon') }}">○ Moon</button>
            <button type="submit" formaction="{{ url_for('preset',
mode='planets') }}">🪐 Planets</button>
            <button type="submit" formaction="{{ url_for('preset', mode='stars')
}}">▫ Deep Sky</button>
        </div>

        <div class="controls">
            <div class="control-group">
                <div class="toggle-row">
                    <span>Horizontal Flip</span>
                    <label class="switch">
                        <input type="checkbox" name="hflip" {% if hflip %}
checked{% endif %}>
                        <span class="slider-ui"></span>
                    </label>
                </div>
            </div>

            <div class="control-group">
                <div class="toggle-row">
                    <span>Vertical Flip</span>
                    <label class="switch">
                        <input type="checkbox" name="vflip" {% if vflip %}
checked{% endif %}>
                        <span class="slider-ui"></span>
                    </label>
                </div>
            </div>

            <div class="control-group">
                <div class="control-label">
                    <span>Exposure (µs)</span>
                    <span class="control-value">{{ exposure_us }}</span>
                </div>
                <input
                    class="slider"
                    type="range"
                    name="exposure_us"
                    min="100"
                    max="2000000"
                    step="500"
                    value="{{ exposure_us }}"
                >
            </div>

            <div class="control-group">
                <div class="control-label">
                    <span>Gain</span>
                    <span class="control-value">{{ "%.1f"|format(gain) }}</span>
                </div>
                <input
                    class="slider"
                    type="range"
                    name="gain"
                    min="1.0"
                    max="16.0"
                    step="0.1"
                    value="{{ "%.1f"|format(gain) }}"
                >
            </div>
```

```
            <div class="control-group">
                <div class="control-label">
                    <span>Digital Focus Zoom</span>
                    <span class="control-value" id="zoomVal">0x</span>
                </div>
                <input class="slider" type="range" id="zoomSlider" min="0"
max="3" step="0.5" value="0">
            </div>
        </div>

        <div class="buttons">
            <button type="submit" class="btn-apply">Apply Adjustments</button>
            <a class="capture btn-capture" href="{{ url_for('capture') }}"
target="_blank" download>⛶ Capture Frame</a>
            <button type="submit" formaction="{{ url_for('preset', mode='reset')
}}" class="btn-reset">Reset Defaults</button>
        </div>

        <div class="footer">
            1920 × 1080   •   15 FPS
        </div>
    </form>
</div>

<script>
    // Digital Zoom Frontend Handler
    const zoomSlider = document.getElementById('zoomSlider');
    const liveStream = document.getElementById('liveStream');
    const zoomVal = document.getElementById('zoomVal');

    zoomSlider.addEventListener('input', function() {
        // A slider value of 0 means scale(1). A value of 3 means scale(4).
        const scaleLevel = 1 + parseFloat(this.value);
        liveStream.style.transform = `scale(${scaleLevel})`;
        zoomVal.innerText = `${this.value}x`;
    });
</script>
</body>
</html>
"""

@app.route("/")
def index():
    with state_lock:
        hflip = camera_settings["hflip"]
        vflip = camera_settings["vflip"]
        exposure_us = camera_settings["exposure_us"]
        gain = camera_settings["gain"]

    return render_template_string(
        INDEX_HTML,
        hflip=hflip,
        vflip=vflip,
        exposure_us=exposure_us,
        gain=gain,
    )

@app.route("/video_feed")
def video_feed():
    return Response(
        mjpeg_generator(),
        mimetype="multipart/x-mixed-replace; boundary=frame",
    )
```

```
@app.route("/settings", methods=["POST"])
def settings():
    hflip = "hflip" in request.form
    vflip = "vflip" in request.form
    exposure_us = clamp_int(request.form.get("exposure_us"), 100, 2000000, 1000)
    gain = clamp_float(request.form.get("gain"), 1.0, 16.0, 1.0)

    with state_lock:
        camera_settings["hflip"] = hflip
        camera_settings["vflip"] = vflip
        camera_settings["exposure_us"] = exposure_us
        camera_settings["gain"] = gain

    restart_event.set()
    return redirect(url_for("index"))

@app.route("/preset/<mode>", methods=["POST"])
def preset(mode):
    # Keep flip settings as they are, but update exposure and gain
    with state_lock:
        if mode == 'moon' or mode == 'reset':
            camera_settings["exposure_us"] = 1000
            camera_settings["gain"] = 1.0
        elif mode == 'planets':
            camera_settings["exposure_us"] = 3000
            camera_settings["gain"] = 1.5
        elif mode == 'stars':
            camera_settings["exposure_us"] = 500000
            camera_settings["gain"] = 10.0

    restart_event.set()
    return redirect(url_for("index"))

@app.route("/capture")
def capture():
    with frame_cond:
        frame = latest_frame

    if frame is None:
        return "No frame available yet. Wait a second and try again.", 503

    filename = datetime.now().strftime("astra_%Y%m%d_%H%M%S.jpg")
    fileobj = BytesIO(frame)
    fileobj.seek(0)

    return send_file(
        fileobj,
        mimetype="image/jpeg",
        as_attachment=True,
        download_name=filename,
    )

def cleanup():
    stop_event.set()
    restart_event.set()

    try:
        if camera_proc and camera_proc.poll() is None:
            camera_proc.terminate()
    except Exception:
        pass

atexit.register(cleanup)
```

```
threading.Thread(target=camera_worker, daemon=True).start()

if __name__ == "__main__":
    serve(app, host="0.0.0.0", port=80)
```

7) Create the systemd service

Create the file:

```
sudo nano /etc/systemd/system/astra.service
```

Paste this service file, replacing `YOUR_USER` with your Linux username:

```
[Unit]
Description=Astra Telescope Server
After=network-online.target avahi-daemon.service
Wants=network-online.target

[Service]
Type=simple
User=YOUR_USER
WorkingDirectory=/home/YOUR_USER/astra
ExecStart=/usr/bin/python3 /home/YOUR_USER/astra/app.py
Restart=always
RestartSec=2
AmbientCapabilities=CAP_NET_BIND_SERVICE
CapabilityBoundingSet=CAP_NET_BIND_SERVICE

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

Then run:

```
sudo systemctl daemon-reload
sudo systemctl enable astra.service
sudo systemctl restart astra.service
sudo systemctl status astra.service
```

8) Verify the web server

Check that Astra is listening on port 80:

```
sudo ss -tlnp | grep :80
```

Open the page from another device on the same network:

- `http://astra.local`
- or `http://<pi-ip>` if mDNS is not available

9) Create the Astra hotspot

Run these commands when you are ready to switch Astra from home Wi-Fi to hotspot mode. Replace `YOUR_STRONG_PASSWORD` with your own password.

```
sudo nmcli connection add     type wifi     ifname wlan0     con-name Astra
autoconnect yes     ssid Astra

sudo nmcli connection modify Astra     802-11-wireless.mode ap     802-11-
```

```
wireless.band bg

sudo nmcli connection modify Astra     wifi-sec.key-mgmt wpa-psk     wifi-
sec.psk "YOUR_STRONG_PASSWORD"

sudo nmcli connection modify Astra     ipv4.method shared     ipv4.addresses
192.168.4.1/24

sudo nmcli connection modify Astra connection.autoconnect yes
```

If you want Astra to stop joining the home Wi-Fi network automatically, disable that profile's autoconnect setting:

```
sudo nmcli connection modify YOUR_HOME_WIFI connection.autoconnect no
```

Activate the hotspot only when you are ready to disconnect from the current network:

```
sudo nmcli connection up Astra
```

10) Use Astra

Connect a phone, tablet, or laptop to the Wi-Fi network named `Astra`.

Then open one of these URLs:

- `http://astra.local`
- `http://192.168.4.1`

The page shows the live feed, flip toggles, exposure, gain, brightness, and the capture button.

Note: If both of those URLs don't work, try `http://10.42.0.1`