Absolutely! I'll extend your existing script with these features:

1. **Copy and Move Selected Shapes (including multiple selection).**
2. **Background selection button** with preset options plus an input for a custom TMS URL.

**How it works:**

* **Selection & Multi-selection:** Click shapes to select. Shift-click to select multiple.
* **Move:** When shapes are selected, a "Move Selected" button appears. Drag the mouse on the map to move selected shapes.
* **Copy:** A "Copy Selected" button duplicates selected shapes with offset.
* **Background selection:** A button opens a popup with preset backgrounds and a field to input custom TMS URL.

Here's the **updated** leaflet-editor-with-attributes.html with your requested features integrated.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title>Leaflet Editor with Attributes, Copy/Move, and Background Selector</title>

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<!-- Leaflet CSS -->

<link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.css" />

<link rel="stylesheet" href="https://unpkg.com/leaflet-draw@1.0.4/dist/leaflet.draw.css" />

<style>

body, html { margin: 0; padding: 0; height: 100%; }

#map { width: 100%; height: 100%; }

.leaflet-bar a.square-icon {

background-image: url('data:image/svg+xml;utf8,<svg fill="%23000" viewBox="0 0 32 32" xmlns="http://www.w3.org/2000/svg"><rect x="6" y="6" width="20" height="20" stroke="black" fill="none"/></svg>');

background-repeat: no-repeat;

background-position: center;

}

#export-btn, #copy-btn, #move-btn, #bg-btn {

position: absolute;

top: 10px;

z-index: 999;

background: white;

padding: 6px 12px;

border: 1px solid #ccc;

cursor: pointer;

font-family: sans-serif;

user-select: none;

}

#export-btn { right: 10px; }

#copy-btn { right: 120px; display: none; }

#move-btn { right: 220px; display: none; }

#bg-btn { left: 10px; }

#bg-selector {

position: absolute;

top: 40px;

left: 10px;

z-index: 1000;

background: white;

border: 1px solid #ccc;

padding: 10px;

display: none;

font-family: sans-serif;

max-width: 300px;

}

#bg-selector label {

display: block;

margin-bottom: 6px;

font-weight: bold;

}

#bg-selector input[type=text] {

width: 100%;

margin-bottom: 6px;

padding: 4px;

box-sizing: border-box;

}

#bg-selector button {

padding: 4px 8px;

cursor: pointer;

}

/\* Selected shape style \*/

.selected {

outline: 3px solid red;

/\* Leaflet layers don't support outline, so we add a red dash pattern \*/

stroke: red !important;

weight: 4 !important;

}

</style>

</head>

<body>

<div id="map"></div>

<button id="export-btn">Export GeoJSON</button>

<button id="copy-btn">Copy Selected</button>

<button id="move-btn">Move Selected</button>

<button id="bg-btn">Background</button>

<div id="bg-selector">

<label for="preset-select">Choose Background:</label>

<select id="preset-select">

<option value="osm">OpenStreetMap</option>

<option value="satellite">Satellite (Esri)</option>

<option value="topo">Topo Map (OpenTopoMap)</option>

</select>

<label for="custom-tms">Or enter custom TMS URL:</label>

<input type="text" id="custom-tms" placeholder="e.g. https://server/{z}/{x}/{y}.png" />

<button id="set-bg-btn">Set Background</button>

<button id="close-bg-btn" style="margin-left:8px;">Close</button>

</div>

<script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"></script>

<script src="https://unpkg.com/leaflet-draw@1.0.4/dist/leaflet.draw.js"></script>

<script>

const map = L.map('map').setView([0, 0], 2);

// Background layers

const backgrounds = {

osm: L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {

attribution: '&copy; OpenStreetMap contributors'

}),

satellite: L.tileLayer('https://server.arcgisonline.com/ArcGIS/rest/services/World\_Imagery/MapServer/tile/{z}/{y}/{x}', {

attribution: 'Tiles &copy; Esri'

}),

topo: L.tileLayer('https://{s}.tile.opentopomap.org/{z}/{x}/{y}.png', {

attribution: 'Map data: &copy; OpenTopoMap contributors'

})

};

// Start with OSM

let currentBackground = backgrounds.osm.addTo(map);

// FeatureGroup for drawn items

const drawnItems = new L.FeatureGroup();

map.addLayer(drawnItems);

// Custom Square tool

L.Draw.Square = L.Draw.Rectangle.extend({

statics: { TYPE: 'square' },

\_drawShape: function (latlng) {

const start = this.\_startLatLng;

const dx = latlng.lng - start.lng;

const dy = latlng.lat - start.lat;

const size = Math.max(Math.abs(dx), Math.abs(dy));

const bounds = new L.LatLngBounds(start, [

start.lat + Math.sign(dy) \* size,

start.lng + Math.sign(dx) \* size

]);

if (!this.\_shape) {

this.\_shape = new L.Rectangle(bounds, this.options.shapeOptions);

this.\_map.addLayer(this.\_shape);

} else {

this.\_shape.setBounds(bounds);

}

}

});

// Draw Control

const drawControl = new L.Control.Draw({

edit: {

featureGroup: drawnItems,

remove: true

},

draw: {

polygon: true,

polyline: true,

rectangle: true,

circle: true,

marker: true,

circlemarker: false

}

});

map.addControl(drawControl);

// Custom draw tools for keyboard shortcut usage

const drawTools = {

marker: new L.Draw.Marker(map),

polyline: new L.Draw.Polyline(map),

polygon: new L.Draw.Polygon(map),

rectangle: new L.Draw.Rectangle(map),

square: new L.Draw.Square(map),

circle: new L.Draw.Circle(map)

};

let activeTool = null;

function activateTool(toolName) {

if (activeTool) activeTool.disable();

if (toolName && drawTools[toolName]) {

activeTool = drawTools[toolName];

activeTool.enable();

console.log(`Activated tool: ${toolName}`);

}

}

// Keyboard shortcuts

document.addEventListener('keydown', function (e) {

if (e.target.tagName === 'INPUT' || e.target.tagName === 'TEXTAREA') return;

switch (e.key.toLowerCase()) {

case 'm': activateTool('marker'); break;

case 'l': activateTool('polyline'); break;

case 'p': activateTool('polygon'); break;

case 'r': activateTool('rectangle'); break;

case 's': activateTool('square'); break;

case 'c': activateTool('circle'); break;

case 'escape':

if (activeTool) {

activeTool.disable();

activeTool = null;

}

break;

}

});

// Prompt for attributes

function promptAttributes() {

const name = prompt("Enter feature name:");

const type = prompt("Enter feature type (e.g. building, road):");

return { name, type };

}

// Handle new feature

map.on(L.Draw.Event.CREATED, function (e) {

const layer = e.layer;

const props = promptAttributes();

layer.feature = {

type: "Feature",

properties: props,

geometry: {}

};

drawnItems.addLayer(layer);

if (activeTool) {

activeTool.disable();

activeTool = null;

}

});

// Export to GeoJSON

document.getElementById("export-btn").addEventListener("click", function () {

const geojson = drawnItems.toGeoJSON();

const dataStr = "data:text/json;charset=utf-8," + encodeURIComponent(JSON.stringify(geojson, null, 2));

const downloadAnchorNode = document.createElement('a');

downloadAnchorNode.setAttribute("href", dataStr);

downloadAnchorNode.setAttribute("download", "features.geojson");

document.body.appendChild(downloadAnchorNode);

downloadAnchorNode.click();

downloadAnchorNode.remove();

});

/\* -------------- New feature: Select & Multi-select shapes -------------- \*/

let selectedLayers = new Set();

function setLayerSelected(layer, selected) {

if (selected) {

selectedLayers.add(layer);

if (layer.setStyle) {

layer.setStyle({ color: 'red', weight: 4 });

} else if (layer.\_icon) {

// Marker: add red border

layer.\_icon.style.outline = '3px solid red';

}

} else {

selectedLayers.delete(layer);

if (layer.setStyle) {

// Reset style - use default

layer.setStyle({ color: '#3388ff', weight: 3 });

} else if (layer.\_icon) {

if (layer.\_icon.style) layer.\_icon.style.outline = '';

}

}

}

function clearAllSelection() {

selectedLayers.forEach(layer => setLayerSelected(layer, false));

selectedLayers.clear();

updateButtonsVisibility();

}

function updateButtonsVisibility() {

const copyBtn = document.getElementById('copy-btn');

const moveBtn = document.getElementById('move-btn');

if (selectedLayers.size > 0) {

copyBtn.style.display = 'inline-block';

moveBtn.style.display = 'inline-block';

} else {

copyBtn.style.display = 'none';

moveBtn.style.display = 'none';

}

}

// Select/deselect on layer click

drawnItems.on('layeradd', function(e) {

const layer = e.layer;

layer.on('click', function(ev) {

// Prevent map click event propagation

L.DomEvent.stopPropagation(ev);

if (ev.originalEvent.shiftKey) {

// Multi-select toggle

if (selectedLayers.has(layer)) {

setLayerSelected(layer, false);

} else {

setLayerSelected(layer, true);

}

} else {

// Single select - clear others

clearAllSelection();

setLayerSelected(layer, true);

}

updateButtonsVisibility();

});

});

// Clicking on map clears selection

map.on('click', function() {

clearAllSelection();

});

/\* ----------- Copy Selected Shapes ------------- \*/

document.getElementById('copy-btn').addEventListener('click', function () {

if (selectedLayers.size === 0) return;

const offsetLat = 0.01; // small offset

const offsetLng = 0.01;

const newLayers = [];

selectedLayers.forEach(layer => {

let newLayer;

if (layer instanceof L.Marker) {

const latlng = layer.getLatLng();

newLayer = new L.Marker([latlng.lat + offsetLat, latlng.lng + offsetLng]);

} else if (layer instanceof L.Polygon) {

const latlngs = layer.getLatLngs();

const newLatlngs = offsetLatLngs(latlngs, offsetLat, offsetLng);

newLayer = new L.Polygon(newLatlngs, layer.options);

} else if (layer instanceof L.Polyline) {

const latlngs = layer.getLatLngs();

const newLatlngs = offsetLatLngs(latlngs, offsetLat, offsetLng);

newLayer = new L.Polyline(newLatlngs, layer.options);

} else if (layer instanceof L.Rectangle) {

const bounds = layer.getBounds();

const newBounds = new L.LatLngBounds(

[bounds.getSouth() + offsetLat, bounds.getWest() + offsetLng],

[bounds.getNorth() + offsetLat, bounds.getEast() + offsetLng]

);

newLayer = new L.Rectangle(newBounds, layer.options);

} else if (layer instanceof L.Circle) {

const center = layer.getLatLng();

const radius = layer.getRadius();

newLayer = new L.Circle([center.lat + offsetLat, center.lng + offsetLng], { radius: radius, ...layer.options });

} else {

console.warn('Unknown layer type for copy:', layer);

return;

}

// Copy feature properties if exist

if (layer.feature && layer.feature.properties) {

newLayer.feature = {

type: "Feature",

properties: Object.assign({}, layer.feature.properties)

};

} else {

newLayer.feature = {

type: "Feature",

properties: {}

};

}

drawnItems.addLayer(newLayer);

newLayers.push(newLayer);

});

clearAllSelection();

newLayers.forEach(l => setLayerSelected(l, true));

updateButtonsVisibility();

});

// Helper to offset LatLngs (handles nested arrays for polygons)

function offsetLatLngs(latlngs, dLat, dLng) {

return latlngs.map(item => {

if (Array.isArray(item)) {

return offsetLatLngs(item, dLat, dLng);

} else {

return L.latLng(item.lat + dLat, item.lng + dLng);

}

});

}

/\* ----------- Move Selected Shapes ------------- \*/

let isMoving = false;

let moveStartPoint = null;

document.getElementById('move-btn').addEventListener('click', function () {

if (selectedLayers.size === 0) return;

alert('Click and drag on the map to move selected shapes. Release mouse button to finish.');

isMoving = true;

moveStartPoint = null;

map.getContainer().style.cursor = 'move';

});

map.on('mousedown', function (e) {

if (!isMoving) return;

moveStartPoint = e.latlng;

});

map.on('mousemove', function (e) {

if (!isMoving || !moveStartPoint) return;

const deltaLat = e.latlng.lat - moveStartPoint.lat;

const deltaLng = e.latlng.lng - moveStartPoint.lng;

selectedLayers.forEach(layer => {

if (layer instanceof L.Marker) {

const pos = layer.getLatLng();

layer.setLatLng([pos.lat + deltaLat, pos.lng + deltaLng]);

} else if (layer instanceof L.Polygon) {

const latlngs = layer.getLatLngs();

const newLatlngs = offsetLatLngs(latlngs, deltaLat, deltaLng);

layer.setLatLngs(newLatlngs);

} else if (layer instanceof L.Polyline) {

const latlngs = layer.getLatLngs();

const newLatlngs = offsetLatLngs(latlngs, deltaLat, deltaLng);

layer.setLatLngs(newLatlngs);

} else if (layer instanceof L.Rectangle) {

const bounds = layer.getBounds();

const newBounds = new L.LatLngBounds(

[bounds.getSouth() + deltaLat, bounds.getWest() + deltaLng],

[bounds.getNorth() + deltaLat, bounds.getEast() + deltaLng]

);

layer.setBounds(newBounds);

} else if (layer instanceof L.Circle) {

const center = layer.getLatLng();

layer.setLatLng([center.lat + deltaLat, center.lng + deltaLng]);

}

});

moveStartPoint = e.latlng;

});

map.on('mouseup', function (e) {

if (isMoving) {

isMoving = false;

moveStartPoint = null;

map.getContainer().style.cursor = '';

// Update GeoJSON feature coordinates after move

selectedLayers.forEach(layer => {

if (!layer.feature) layer.feature = { type: "Feature", properties: {} };

layer.feature.geometry = layer.toGeoJSON().geometry;

});

}

});

/\* ---------------- Background Selector ------------------ \*/

const bgBtn = document.getElementById('bg-btn');

const bgSelector = document.getElementById('bg-selector');

const presetSelect = document.getElementById('preset-select');

const customTmsInput = document.getElementById('custom-tms');

const setBgBtn = document.getElementById('set-bg-btn');

const closeBgBtn = document.getElementById('close-bg-btn');

bgBtn.addEventListener('click', () => {

bgSelector.style.display = 'block';

// Reset inputs to current background

if (currentBackground === backgrounds.osm) presetSelect.value = 'osm';

else if (currentBackground === backgrounds.satellite) presetSelect.value = 'satellite';

else if (currentBackground === backgrounds.topo) presetSelect.value = 'topo';

else presetSelect.value = '';

customTmsInput.value = '';

});

closeBgBtn.addEventListener('click', () => {

bgSelector.style.display = 'none';

});

setBgBtn.addEventListener('click', () => {

let layer;

if (customTmsInput.value.trim()) {

try {

layer = L.tileLayer(customTmsInput.value.trim(), {

attribution: 'Custom TMS Layer'

});

} catch (err) {

alert('Invalid TMS URL.');

return;

}

} else {

const preset = presetSelect.value;

layer = backgrounds[preset] || backgrounds.osm;

}

if (currentBackground) map.removeLayer(currentBackground);

currentBackground = layer;

currentBackground.addTo(map);

bgSelector.style.display = 'none';

});

</script>

</body>

</html>

Great question! Let me break it down for you:

**1. Public TMS Examples for High-Resolution Imagery**

Some popular public TMS (Tile Map Service) providers offer high-res imagery:

* **Esri Satellite** (already in example):

https://server.arcgisonline.com/ArcGIS/rest/services/World\_Imagery/MapServer/tile/{z}/{y}/{x}

* **Google Satellite**: *Not officially available as TMS and restricted by Google’s ToS* (so avoid direct usage).
* **NASA GIBS** (Global Imagery Browse Services) — provides satellite imagery from MODIS etc.:

Example tile URL:

https://gibs.earthdata.nasa.gov/wmts/epsg3857/best/VIIRS\_SNPP\_CorrectedReflectance\_TrueColor/default/2023-01-01/GoogleMapsCompatible\_Level9/{z}/{y}/{x}.jpg

* **OpenAerialMap**:

They host various aerial imagery; tile URLs vary by dataset.

* **OpenMapSurfer Satellite** (not always high res but decent):

https://korona.geog.uni-heidelberg.de/tiles/roads/x={x}&y={y}&z={z}

**2. Converting Your Own High-Resolution Image into TMS**

If you have a large, high-res image and want to serve it as a TMS (tiles for Leaflet or any slippy map), you generally need to:

* **Georeference** your image: assign coordinates (lat/lon) or projection to it.
* **Tile it**: generate zoom-level folders and XYZ tiles.

**Tools for this:**

* **GDAL** (open-source GIS toolkit):

Use gdal2tiles.py to convert georeferenced images to tiles.

Example:

gdal2tiles.py -z 0-5 -w leaflet your\_image.tif output\_folder

* -z 0-5 generates zoom levels 0 to 5 (adjust as needed)
* -w leaflet generates an index.html for Leaflet demo

**Steps:**

1. **Georeference your image** if not georeferenced already:

* Use QGIS or gdal\_translate with a world file to assign coordinates.

1. **Run gdal2tiles** to generate tiles:

gdal2tiles.py your\_image.tif output\_folder

1. **Serve the tiles folder with a simple HTTP server:**

cd output\_folder

python3 -m http.server 8080

1. **Use the local tile URL in Leaflet:**

L.tileLayer('http://localhost:8080/{z}/{x}/{y}.png').addTo(map);

**3. Quick Example Using a Public TMS**

You can try NASA GIBS high-res tiles like this:

L.tileLayer('https://gibs.earthdata.nasa.gov/wmts/epsg3857/best/VIIRS\_SNPP\_CorrectedReflectance\_TrueColor/default/2023-01-01/GoogleMapsCompatible\_Level9/{z}/{y}/{x}.jpg', {

attribution: 'NASA GIBS',

maxZoom: 9

}).addTo(map);

This provides true-color satellite imagery for Jan 1, 2023, zoom up to 9.

**Summary**

* **Public TMS:** Esri, NASA GIBS, OpenAerialMap for high-res satellite/aerial imagery
* **Custom TMS:** Georeference your image + use gdal2tiles.py to generate tiles + serve tiles via HTTP + use Leaflet’s L.tileLayer with URL pattern

If you want, I can help you prepare a simple script or workflow for your own image conversion too! Just let me know what format your image is in and what software you use.

Absolutely! Here's a step-by-step guide on how to convert your high-resolution image into TMS tiles using **GDAL's gdal2tiles.py**. This process assumes your image is **georeferenced** or you will georeference it first.

**Step-by-Step Guide to Convert a High-Resolution Image to TMS Tiles**

**Step 1: Prepare Your Image**

Your image should be **georeferenced** so that Leaflet knows how to place it on the map. Georeferencing means the image has embedded coordinates or a separate world file (.wld) describing where it fits on the globe.

* Common georeferenced formats: GeoTIFF (.tif), or any raster with a .tfw (world file).
* If your image is just a plain JPEG or PNG with no georeferencing, you will need to georeference it first (I'll explain below).

**Step 2: Install GDAL**

You need GDAL installed on your computer.

* **Windows**: Use OSGeo4W installer from <https://trac.osgeo.org/osgeo4w/>
* **Mac**: Use Homebrew

brew install gdal

* **Linux**: Use your package manager, e.g.

sudo apt-get install gdal-bin python3-gdal

**Step 3: Georeference Your Image (If Needed)**

If your image is **not georeferenced**, you can:

* Use **QGIS** (free GIS software) to georeference the image manually.
  1. Open QGIS
  2. Open *Georeferencer* (under Raster menu)
  3. Load your image
  4. Add control points linking image pixels to known coordinates (e.g., lat/lon)
  5. Set the transformation settings and output file format (GeoTIFF recommended)
  6. Run the georeferencing and save the georeferenced image
* Or create a **world file** (.wld) with pixel size and origin info (less accurate).

**Step 4: Use gdal2tiles.py to Generate Tiles**

Run this command in your terminal or command prompt:

gdal2tiles.py -z 0-5 -w leaflet path/to/your\_georeferenced\_image.tif output\_tiles\_folder

Explanation:

* -z 0-5: generate zoom levels 0 through 5 (adjust for your needs)
* -w leaflet: generate an HTML viewer compatible with Leaflet (optional, for testing)
* path/to/your\_georeferenced\_image.tif: path to your georeferenced image file
* output\_tiles\_folder: directory where tiles and the HTML viewer will be saved

**Step 5: Serve the Tiles**

The output folder will contain:

* A folder structure with tiles: {z}/{x}/{y}.png
* leaflet.html demo file (if you used -w leaflet)

To view tiles in your Leaflet app, you need to serve them via an HTTP server (because Leaflet can't load files locally due to browser restrictions):

Run:

cd output\_tiles\_folder

python3 -m http.server 8080

This will serve files at http://localhost:8080

**Step 6: Use the Tiles in Leaflet**

In your Leaflet map, add a tile layer like this:

L.tileLayer('http://localhost:8080/{z}/{x}/{y}.png', {

maxZoom: 5,

attribution: 'Custom Tiles'

}).addTo(map);

**Optional: Example Workflow for Georeferencing + Tiling in QGIS + GDAL**

1. **Open QGIS** and load your image in the *Georeferencer* tool.
2. Add at least 3-4 control points linking image pixels to real-world coordinates.
3. Export the georeferenced image as GeoTIFF.
4. Open your terminal, run gdal2tiles.py on the GeoTIFF.
5. Serve the tiles and use in your Leaflet project.

**Summary**

| **Step** | **Tool/Command** | **Purpose** |
| --- | --- | --- |
| Install GDAL | brew install gdal or OSGeo4W installer | Get gdal2tiles.py |
| Georeference image | QGIS Georeferencer / Create world file | Assign coordinates |
| Generate tiles | gdal2tiles.py -z 0-5 -w leaflet image.tif out | Convert image into TMS tiles |
| Serve tiles | python3 -m http.server 8080 | Serve tiles locally |
| Add tile layer | L.tileLayer('http://localhost:8080/{z}/{x}/{y}.png') | Use tiles in Leaflet |

If you want, I can also provide a quick QGIS georeferencing tutorial or help you with a sample command — just ask!

Perfect! Since you already have **QGIS installed on Windows**, you're in great shape. 🎉

Let’s now go through the **exact steps** using **QGIS and GDAL’s gdal2tiles.py**, which you already have thanks to QGIS.

**✅ Quick Overview of What We'll Do**

1. ✔️ Georeference your image (if not already)
2. ✔️ Locate and run gdal2tiles.py to generate TMS tiles
3. ✔️ Serve your tiles using a local server
4. ✔️ Load the tiles in your Leaflet map

**🔧 STEP 1: Georeference Your Image (if needed)**

If your image is **not** georeferenced (e.g., it’s a scanned map, drone photo, satellite still):

**👉 Use QGIS Georeferencer**

1. Open **QGIS**
2. Go to Raster → Georeferencer → Georeferencer...
3. Click Open Raster and choose your image
4. Click the **"Add Point"** tool and click on known locations
5. For each point, enter **latitude and longitude** (or use map canvas to pick coordinates)
6. When done, click the **gear icon ⚙️** for *Transformation Settings*:
   * **Output raster**: Save location for the new image
   * **Target SRS**: Use EPSG:4326 (WGS 84) for Leaflet compatibility
   * Choose transformation type: **Linear** or **Polynomial 1**
7. Click **Start Georeferencing**

✅ You now have a **GeoTIFF** file (e.g. my\_georef\_image.tif)

**🔍 STEP 2: Locate gdal2tiles.py on Windows**

Since you have QGIS installed, gdal2tiles.py is already available.

**✔️ Use the “OSGeo4W Shell”:**

1. Search for OSGeo4W Shell in the Start Menu (installed with QGIS)
2. Right-click → **Run as Administrator**

This shell is configured to find gdal2tiles.py and other GDAL tools.

**📦 STEP 3: Run gdal2tiles.py to Generate Tiles**

In the OSGeo4W shell, run:

gdal2tiles.py -z 0-5 -w leaflet C:\path\to\my\_georef\_image.tif C:\path\to\output\_tiles

Replace the paths with the full file paths to your .tif and output folder.

**Options Explained:**

* -z 0-5 = zoom levels (adjust for your needs)
* -w leaflet = generate a sample Leaflet HTML file
* my\_georef\_image.tif = the georeferenced image
* output\_tiles = where your tiles will go

✅ When complete, the output folder will contain:

output\_tiles/

├── leaflet.html

├── 0/

├── 1/

├── ...

**🌐 STEP 4: Serve the Tiles with a Local Web Server**

From **Command Prompt** or **PowerShell** (not OSGeo shell):

cd C:\path\to\output\_tiles

python -m http.server 8080

(Requires Python installed — QGIS usually includes it.)

Now open your browser to:

http://localhost:8080/leaflet.html

✅ You will see your tiled image in Leaflet.

**🗺️ STEP 5: Use the Tiles in Your Own Leaflet App**

Add this tile layer in your HTML:

L.tileLayer('http://localhost:8080/{z}/{x}/{y}.png', {

maxZoom: 5,

attribution: 'My Custom Imagery'

}).addTo(map);

**📁 Example Folder Output**

output\_tiles/

├── 0/

│ ├── 0/

│ │ ├── 0.png

├── 1/

├── 2/

├── ...

├── leaflet.html

**💡 Tips**

* If you want **higher detail**, increase -z:
* gdal2tiles.py -z 5-8 -w leaflet my\_georef\_image.tif output\_tiles
* For **faster generation**, add:
* --processes 4

**✅ Summary**

| **Task** | **Tool** |
| --- | --- |
| Georeference image | QGIS Georeferencer |
| Generate tiles | gdal2tiles.py via OSGeo4W Shell |
| Serve tiles | python -m http.server |
| View in browser | http://localhost:8080/leaflet.html |
| Use in Leaflet app | L.tileLayer('http://localhost:8080/{z}/{x}/{y}.png') |

Would you like a **ready-made HTML viewer template** that loads your tiles with custom tools, like in your earlier project? I can provide that too.