 **NASA FIRMS** → Provides near real-time wildfire/thermal anomaly data via API (CSV/JSON).

 **USGS Earthquakes** → Real-time earthquake alerts available in GeoJSON (open).

 **Weather.gov / NWS** → Official weather warnings, watches, advisories via CAP/GeoJSON feeds.

 **WMATA Real-time** → JSON APIs for bus positions, rail predictions, and incidents (API key).

 **Waze CCP** → Partner-only JSON feeds for accidents, jams, and road closures.  
applied for key

 **USGS NWIS** → Streamflow and gage height data for local rivers in real time (JSON).

 **Alexandria Open Data** → Police/fire datasets updated bi-annually, not real-time.

 **PulsePoint** → Alexandria not onboard; only neighboring counties visible (web only).

 **Broadcastify / IPN** → Live Fire/EMS radio audio (free) and structured IPN alerts (paid).

 **City Mass Notification / IPAWS** → Official emergency alerts (WEA/EAS), no public API; CAP feeds via NWS.

 **DASH GTFS-RT** → Transit positions and alerts, already covered under WMATA APIs.

 **Alexandria Flood Sensors** → Local flood gauges exist but no open API (dashboard only).

 **Weather APIs (OWM/AccuWeather/WeatherAPI.com)** → Supplemental JSON for conditions/forecasts, not primary alerts.

 **HERE Maps API** → Optional commercial service for traffic/road conditions (API key).

**NASA FIRMS API – Area Request Documentation**

**Endpoint**

https://firms.modaps.eosdis.nasa.gov/api/area/csv/YOUR\_KEY/VIIRS\_SNPP\_NRT/-77.5,38.6,-76.8,39.1/7

**Parameters**

* **YOUR\_KEY** → your FIRMS MAP\_KEY (personal API key from NASA FIRMS).
* **VIIRS\_SNPP\_NRT** → the sensor/dataset to use. Options include:
  + VIIRS\_SNPP\_NRT (Suomi NPP satellite, near real-time)
  + VIIRS\_NOAA20\_NRT / VIIRS\_NOAA21\_NRT (other VIIRS satellites)
  + MODIS\_NRT (MODIS sensor, near real-time)
* **-77.5,38.6,-76.8,39.1** → bounding box for the area of interest in **minLon, minLat, maxLon, maxLat** format.
  + This example covers Alexandria / DC region.
* **7** → number of days back to retrieve detections.

The request returns a **CSV file** containing all detected fire/hotspot pixels inside the bounding box and time window.

**Sample Output (truncated)**

latitude,longitude,bright\_ti4,scan,track,acq\_date,acq\_time,satellite,instrument,confidence,version,bright\_ti5,frp,daynight

66.42283,58.04228,328.91,0.32,0.55,2025-09-11,35,N,VIIRS,n,2.0NRT,274.17,1.84,N

67.13786,56.77925,328.51,0.57,0.52,2025-09-11,35,N,VIIRS,n,2.0NRT,269.84,4.43,N

**Field Descriptions**

* **latitude / longitude** → geographic coordinates of the detected hotspot.
* **bright\_ti4 / bright\_ti5** → brightness temperature (Kelvin) of VIIRS channels I4/I5, used to detect thermal anomalies. Higher values often indicate stronger fires.
* **scan / track** → size of the pixel footprint in degrees (across-scan and along-track). Indicates the spatial resolution of the detection.
* **acq\_date / acq\_time** → date and time of acquisition (UTC). Time is given as HHMM.
* **satellite** → satellite name (N = Suomi NPP, J = NOAA-20/21, A = Aqua, T = Terra).
* **instrument** → sensor used, e.g., VIIRS or MODIS.
* **confidence** → qualitative confidence in the detection (low, nominal, high, or n = not specified for VIIRS).
* **version** → data processing version (e.g., 2.0NRT = VIIRS near real-time version 2.0).
* **frp** → Fire Radiative Power (MW). Proxy for fire intensity and energy release.
* **daynight** → D = day detection, N = night detection.

**USGS Earthquake API – GeoJSON Feed Documentation**

**Endpoint**

https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/significant\_week.geojson

**Variants**

The USGS feed has multiple predefined endpoints depending on **time window** and **magnitude threshold**. Format is:

https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/[MAG]\_[TIME].[FORMAT]

* **[MAG]** → all, 1.0, 2.5, 4.5, significant
* **[TIME]** → hour, day, week, month
* **[FORMAT]** → geojson, kml, csv, quakeml

**Example:**

* All earthquakes past week:  
  https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all\_week.geojson
* Magnitude 2.5+ past day:  
  https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/2.5\_day.geojson

**Sample Output (GeoJSON excerpt)**

{

"type": "Feature",

"properties": {

"mag": 2.7,

"place": "7km NW of Ashburn, Virginia",

"time": 1694491320000,

"updated": 1694491620000,

"tz": null,

"url": "https://earthquake.usgs.gov/earthquakes/eventpage/se60473922",

"felt": 23,

"cdi": 3.5,

"mmi": null,

"alert": null

},

"geometry": {

"type": "Point",

"coordinates": [ -77.53, 39.06, 10.0 ]

},

"id": "se60473922"

}

**Field Descriptions**

* **mag** → Magnitude of the earthquake.
* **place** → Human-readable location (relative to nearest town/city).
* **time / updated** → Epoch timestamp (ms).
* **url** → Link to full event details on USGS website.
* **felt** → Number of “Did You Feel It?” reports submitted.
* **cdi** → Community Determined Intensity (shaking felt by the public, 1–10).
* **mmi** → Instrumental shaking intensity (Modified Mercalli Index).
* **alert** → USGS PAGER alert level (green/yellow/orange/red).
* **geometry.coordinates** → [longitude, latitude, depth\_km].
* **id** → Unique event identifier.

**Usage in Your Project**

* Directly map geometry.coordinates to show earthquake epicenters.
* Use **mag + place** for alerts (e.g., “Magnitude 3.2 near Alexandria, VA”).
* Combine **felt / cdi** to adjust severity messaging.
* The **feed updates every minute**, so it’s real-time ready for your dashboard.

**NOAA / Weather.gov Alerts API Documentation**

**Endpoint**

https://api.weather.gov/alerts

**Parameters**

This API supports filters to narrow alerts to your region of interest.

* **area** → 2-letter U.S. state code.
  + Example: area=VA → returns Virginia alerts.
* **zone** → specific NWS forecast zone or county code (FIPS).
* **status** → actual, exercise, system, test.
* **event** → filter by type (e.g., Tornado Warning).

**Example URLs:**

* All active Virginia alerts:  
  https://api.weather.gov/alerts/active?area=VA
* Active alerts in Alexandria County zone (e.g., VAZ054):  
  https://api.weather.gov/alerts/active?zone=VAZ054

**Sample Output (GeoJSON excerpt)**

{

"type": "Feature",

"properties": {

"id": "https://api.weather.gov/alerts/NWS-IDP-PROD-123456-7890",

"areaDesc": "Arlington; Alexandria",

"sent": "2025-09-17T14:00:00-04:00",

"effective": "2025-09-17T14:00:00-04:00",

"expires": "2025-09-17T16:00:00-04:00",

"status": "Actual",

"messageType": "Alert",

"event": "Severe Thunderstorm Warning",

"severity": "Severe",

"certainty": "Observed",

"urgency": "Immediate",

"headline": "Severe Thunderstorm Warning issued September 17 at 2:00PM EDT",

"description": "At 2:00 PM EDT, a severe thunderstorm was located near Alexandria...",

"instruction": "Take cover now! Move to a basement or interior room..."

},

"geometry": null

}

**Field Descriptions**

* **id** → Unique CAP (Common Alerting Protocol) alert identifier.
* **areaDesc** → Areas impacted (counties, cities).
* **sent / effective / expires** → Timestamps for when the alert was issued, takes effect, and expires.
* **status** → Usually “Actual” for real alerts.
* **event** → Alert type (e.g., Flood Warning, Tornado Watch).
* **severity** → Minor, Moderate, Severe, or Extreme.
* **certainty** → Observed, Likely, Possible.
* **urgency** → Immediate, Expected, Future.
* **headline / description** → Human-readable text for public alerts.
* **instruction** → Recommended protective actions.
* **geometry** → Sometimes includes polygon data for affected regions.

**Usage in Your Project**

* Display **active Alexandria alerts** in a dashboard ticker or alert banner.
* Filter by **area=VA** and **zone=VAZ054** to isolate Alexandria-specific warnings.
* Highlight **event, severity, urgency, instruction** for public-facing emergency notifications.
* Data is **real-time, official, and authoritative** from NOAA/NWS

**WMATA Incidents API — EAS-Focused (Alexandria: KSS / BRD / PVT)**

**0) Auth (required for all requests)**

Get a key: <https://developer.wmata.com>

Add header to every call:

api\_key: YOUR\_WMATA\_KEY

Accept: application/json

**1) Core endpoint (transit disruptions)**

**GET**

https://api.wmata.com/Incidents.svc/json/Incidents

**What you get**

A single list Incidents that may include **Metrorail** and **Metrobus** disruptions (unplanned + planned).

Each item:

IncidentType → "Metrorail" or "Metrobus"

Description → Human-readable summary (often includes cause & impact)

LinesAffected → e.g., "BL; YL"

LocationsAffected → station/stop codes (e.g., "BRD; KSS; PVT")

DateUpdated → ISO timestamp of last update

**Alexandria station codes**

KSS = King St–Old Town

BRD = Braddock Road

PVT = Potomac Yard–VT

**Alexandria-relevant rail lines**

Blue (BL) and Yellow (YL)

**2) Postman setup (incidents only)**

**Method:** GET  
**URL:** https://api.wmata.com/Incidents.svc/json/Incidents  
**Headers:**

api\_key: YOUR\_WMATA\_KEY

Accept: application/json

**Postman Tests (filter for Alexandria + BL/YL):**

const data = pm.response.json();

const keepStations = new Set(["KSS","BRD","PVT"]);

const inAlexandria = (s = "") => s.split(";").map(x=>x.trim()).some(c=>keepStations.has(c));

const linesAlex = (s = "") => ["BL","YL"].some(l => s.includes(l));

const relevant = (data.Incidents || []).filter(i =>

linesAlex(i.LinesAffected || "") || inAlexandria(i.LocationsAffected || "")

);

pm.collectionVariables.set("WMATA\_INCIDENTS\_ALEX", JSON.stringify(relevant));

pm.test("Incidents fetched", () => pm.expect(data.Incidents).to.be.an('array'));

**3) cURL (quick check)**

curl -H "api\_key: YOUR\_WMATA\_KEY" \

-H "Accept: application/json" \

"https://api.wmata.com/Incidents.svc/json/Incidents"

**4) Sample incident (trimmed)**

{

"IncidentType": "Metrorail",

"Description": "BL/YL: Single tracking near Braddock Rd due to track work. Expect delays.",

"LinesAffected": "BL; YL",

"LocationsAffected": "BRD; KSS; PVT",

"DateUpdated": "2025-09-17T17:40:00"

}

**5) EAS logic (recommended)**

**Trigger “heads-up” banner** when ANY is true:

IncidentType == "Metrorail" **AND** LinesAffected contains BL or YL

LocationsAffected contains any of KSS, BRD, PVT

**Escalate to “service disruption” alert** if:

Description contains strong impact terms:

["suspended","no service","station closed","evacuate","police activity","power outage"]

Or if you parse time impact (e.g., > 20 min delay) from Description

**De-duplicate** by (IncidentType + LinesAffected + LocationsAffected) and prefer the **latest DateUpdated**.

**6) Front-end snippets (minimal, incidents only)**

**Fetch & filter (browser JS)**

const WMATA\_KEY = "YOUR\_WMATA\_KEY";

const ALEX\_STATIONS = new Set(["KSS","BRD","PVT"]);

function affectsAlexandria(inc) {

const lines = inc.LinesAffected || "";

const locs = (inc.LocationsAffected || "").split(";").map(s=>s.trim());

const alexHit = locs.some(c => ALEX\_STATIONS.has(c));

const lineHit = ["BL","YL"].some(l => lines.includes(l));

return alexHit || lineHit;

}

async function fetchIncidents() {

const r = await fetch("https://api.wmata.com/Incidents.svc/json/Incidents", {

headers: { "api\_key": WMATA\_KEY, "Accept": "application/json" }

});

const j = await r.json();

return (j.Incidents || []).filter(affectsAlexandria);

}

**Banner rendering (example)**

function classifySeverity(desc="") {

const d = desc.toLowerCase();

if (/(suspended|no service|station closed|evacuate|police activity)/.test(d)) return "critical";

if (/(single tracking|delay)/.test(d)) return "major";

return "info";

}

function renderIncidentBanner(inc) {

const sev = classifySeverity(inc.Description);

const el = document.createElement("div");

el.className = `alert-banner ${sev}`; // style .critical/.major/.info in CSS

el.textContent = inc.Description;

document.querySelector("#alerts").appendChild(el);

}

**Refresh cadence**

**Incidents:** every **60–120s** (they don’t change second-by-second)

Clear & re-render to avoid stacking duplicates.

**7) What Incidents won’t give you**

It is **transit-focused**. It may mention an *accident* **only if** it impacts Metro operations.

For general **roadway crashes, closures, hazards**: add **Waze CCP** (accident feed) and/or **Alexandria police/open-data** to your stack.

**8) Optional: elevator/escalator outages (accessibility)**

Some events are critical for vulnerable populations.

**GET**

https://api.wmata.com/Rail.svc/json/jElevatorIncidents

Filter for LocationName ∈ ["KING ST", "BRADDOCK RD", "POTOMAC YARD–VT"] and raise an **accessibility alert** (lower priority than service suspension but still important).

**9) Minimal storage schema (for your backend)**

type WmataIncident = {

id: string; // hash of IncidentType+LinesAffected+LocationsAffected+Description

type: "Metrorail" | "Metrobus";

lines: string[]; // parse LinesAffected

stations: string[]; // parse LocationsAffected

description: string;

updatedAt: string; // DateUpdated

severity: "critical" | "major" | "info"; // derived

};

**10) QA checklist**

✅ Header includes api\_key and Accept: application/json

✅ Filter to BL/YL and KSS/BRD/PVT

✅ De-duplication logic + newest DateUpdated wins

✅ Fallback: if no relevant incidents, show “No Metrorail disruptions affecting Alexandria”

**Waze for Cities (CCP) — Incidents/Accidents Feed (Alexandria)**

**0) Access**

* Apply (free for governments/academia): **Waze for Cities (CCP)**
* Once approved, you’ll receive a **private HTTPS feed URL** (or regional URLs) and an **auth token**.
* Data includes **alerts** (crashes, hazards, road-closures), **jams** (congestion), and **irregularities** (anomalies vs typical traffic).

Until access is granted, you can prototype your parser using saved JSON fixtures and flip the URL later.

**1) Postman — core request (typical)**

**Method:** GET  
**URL (example pattern; yours will differ):**

https://<your-waze-ccp-host>/traffic?format=JSON&types=alerts,jams,irregularities&bbox=-77.20,38.70,-76.95,39.00

**Headers**

Authorization: Bearer YOUR\_WAZE\_TOKEN

Accept: application/json

**Query params**

* types → alerts,jams,irregularities (choose any subset)
* bbox → minLon,minLat,maxLon,maxLat (e.g., Alexandria/DC box above)
* Some feeds support since=SECONDS\_AGO or ts/from—use if provided in your feed docs.

If your feed is pre-scoped by region, you may not need bbox.

**2) Sample JSON (trimmed)**

{

"alerts": [

{

"uuid": "b3f1c...",

"type": "ACCIDENT",

"subtype": "ACCIDENT\_MINOR",

"roadType": "STREET",

"location": { "x": -77.0532, "y": 38.8129 },

"street": "Duke St",

"city": "Alexandria",

"pubMillis": 1694985600000,

"reportRating": 4,

"reliability": 7

},

{

"uuid": "8b2c7...",

"type": "HAZARD",

"subtype": "HAZARD\_ON\_ROAD",

"location": { "x": -77.0449, "y": 38.8048 },

"street": "King St",

"city": "Alexandria",

"pubMillis": 1694987400000

}

],

"jams": [

{

"uuid": "jam-12ab",

"line": [

{ "x": -77.0601, "y": 38.8171 },

{ "x": -77.0542, "y": 38.8143 }

],

"severity": 2,

"speed": 8.5,

"length": 520,

"street": "US-1 (Richmond Hwy)",

"blockingAlertUuid": "b3f1c..."

}

],

"irregularities": [

{

"uuid": "ir-9ff1",

"severity": 3,

"speed": 12,

"regularSpeed": 28,

"confidence": 6,

"nThumbsUp": 3,

"line": [

{ "x": -77.0470, "y": 38.8069 },

{ "x": -77.0418, "y": 38.8060 }

]

}

]

}

**Key fields**

* **alerts** → discrete events: type (ACCIDENT, HAZARD, ROAD\_CLOSED, etc.), location (lon x, lat y), street, pubMillis (epoch ms).
* **jams** → polylines with speed, length (m), severity, optional link to blocking alert via blockingAlertUuid.
* **irregularities** → traffic slower than historical norm: regularSpeed vs speed.

**3) EAS logic (recommended)**

**Trigger accident alert** when:

* alerts[\*].type == "ACCIDENT" (any subtype), **and** location within Alexandria bbox.
* Optional: escalate if:
  + ROAD\_CLOSED appears, or
  + a **jam** references blockingAlertUuid that’s an ACCIDENT and **severity ≥ 2** or **length > 500 m**.

**De-duplication**

* Use uuid as primary key; update existing incidents on new pulls.
* Consider merging **alert + jam** when blockingAlertUuid ties them.

**Prioritization heuristic**

* Level **Critical**: ROAD\_CLOSED, multi-segment jam with blocking accident, or incidents near **KSS/BRD/PVT** ingress routes (station access).
* Level **Major**: ACCIDENT with jam severity ≥ 2 or speed < 10 mph.
* Level **Info**: HAZARD\_ON\_ROAD, jam severity=1.

**4) Postman Tests (quick filter for accidents in Alexandria)**

const data = pm.response.json();

const bbox = { minX: -77.20, minY: 38.70, maxX: -76.95, maxY: 39.00 };

const inBbox = (p) => p.x >= bbox.minX && p.x <= bbox.maxX && p.y >= bbox.minY && p.y <= bbox.maxY;

const accidents = (data.alerts || []).filter(a =>

a.type === "ACCIDENT" && a.location && inBbox(a.location)

);

pm.collectionVariables.set("WAZE\_ACCIDENTS\_ALEX", JSON.stringify(accidents));

pm.test("Accidents array ready", () => pm.expect(accidents).to.be.an('array'));

**5) Front-end snippets**

**Map markers for alerts**

function toLatLng(a){ return [a.location.y, a.location.x]; }

function iconFor(a){

if (a.type === "ACCIDENT") return "accident";

if (a.type === "ROAD\_CLOSED") return "closure";

if (a.type && a.type.includes("HAZARD")) return "hazard";

return "info";

}

function renderAlerts(map, alerts) {

alerts.forEach(a => {

const marker = L.marker(toLatLng(a)).addTo(map);

marker.bindPopup(`${a.type} on ${a.street || 'road'}<br>${new Date(a.pubMillis).toLocaleString()}`);

marker.\_icon.classList.add(iconFor(a)); // style via CSS classes

});

}

**Polylines for jams**

function renderJams(map, jams){

jams.forEach(j => {

const latlngs = (j.line || []).map(p => [p.y, p.x]);

const poly = L.polyline(latlngs).addTo(map);

poly.bindPopup(`Jam on ${j.street || 'road'}<br>speed ${j.speed} mph, len ${j.length} m`);

});

}

**Severity banner (example)**

function classifyAlert(a){

const t = a.type || "";

if (t === "ROAD\_CLOSED") return "critical";

if (t === "ACCIDENT") return "major";

return "info";

}

**6) Refresh & rate guidance**

* **Pull every 30–60s** for alerts/jams.
* Backoff if unchanged (compare previous uuid set).
* Cache responses; don’t fan out with many overlapping bbox calls.

**7) Data hygiene & geofencing**

* Normalize coordinates (Waze uses {x: lon, y: lat}).
* Snap to **Alexandria city limits** (GIS polygon) to suppress border noise.
* Optional: fuse with **WMATA Incidents** to mark transit access impacted (e.g., accident on US-1 near PVT).

**8) Edge cases**

* Duplicate alerts with evolving subtype—update same uuid.
* Stale alerts: if pubMillis older than your alert TTL (e.g., 90 minutes) and no jam linkage, downgrade/remove.
* Temporary **token** invalidation → handle 401/403 and retry with refreshed credentials.

If you want, I can generate a **mock JSON fixture** (Alexandria accidents + jams) for your dev environment, plus a tiny **Node/Python proxy** template you can deploy to hide the CCP token and serve your frontend with clean JSON.

**Alexandria Open Data — Police Incidents**

**0) Endpoint & auth**

* Base portal: https://data.alexandriava.gov/
* **Incidents API (JSON):**  
  https://data.alexandriava.gov/resource/ci25-wg4g.json
* Auth: Public (rate-limited). For higher limits, use an app token in header X-App-Token: YOUR\_TOKEN (optional).

**1) Postman — core request (recent incidents, sorted)**

**GET**

https://data.alexandriava.gov/resource/ci25-wg4g.json?$select=incident\_id,offense,location,block\_address,report\_datetime,latitude,longitude&$order=report\_datetime DESC&$limit=50

**Headers (optional but recommended)**

X-App-Token: YOUR\_TOKEN

Accept: application/json

**2) Filter to Alexandria bbox (tight area) + last 7 days**

Socrata’s **$where** supports spatial + time filters.

**GET**

https://data.alexandriava.gov/resource/ci25-wg4g.json

?$select=incident\_id,offense,location,block\_address,report\_datetime,latitude,longitude

&$where=within\_box(location,-77.12,38.87,-77.02,38.76)

AND report\_datetime >= dateadd('day',-7, now())

&$order=report\_datetime DESC

&$limit=100

*(Coordinates are within\_box(location, minLon, maxLat, maxLon, minLat); Socrata uses that order.)*

**3) Filter by offense types (e.g., violent/property for alerting)**

**GET**

https://data.alexandriava.gov/resource/ci25-wg4g.json

?$select=incident\_id,offense,location,block\_address,report\_datetime,latitude,longitude

&$where=within\_box(location,-77.12,38.87,-77.02,38.76)

AND report\_datetime >= dateadd('day',-3, now())

AND upper(offense) in ('ASSAULT','ROBBERY','BURGLARY','WEAPONS OFFENSE')

&$order=report\_datetime DESC

&$limit=100

**4) Sample JSON (trim)**

[

{

"incident\_id": "23-001234",

"offense": "BURGLARY",

"block\_address": "500 KING ST",

"report\_datetime": "2025-09-16T22:30:00.000",

"latitude": "38.8049",

"longitude": "-77.0469",

"location": { "type": "Point", "coordinates": [-77.0469, 38.8049] }

}

]

**Common fields**

* incident\_id – unique ID
* offense – offense category
* block\_address – block-level (privacy-preserving)
* report\_datetime – when reported
* latitude/longitude and location (Point) – mapping

**5) Postman “Tests” (simple EAS filter + severity tags)**

const rows = pm.response.json();

function classify(offense="") {

const o = offense.toUpperCase();

if (/(WEAPONS|HOMICIDE|SHOTS|ROBBERY|ASSAULT)/.test(o)) return "critical";

if (/(BURGLARY|ARSON)/.test(o)) return "major";

return "info";

}

const enriched = rows.map(r => ({ ...r, severity: classify(r.offense||"") }));

pm.collectionVariables.set("ALX\_POLICE\_INCIDENTS", JSON.stringify(enriched));

pm.test("Got incidents array", () => pm.expect(rows).to.be.an('array'));

**6) Front-end snippet (Leaflet markers)**

async function fetchAlxIncidents() {

const url = "https://data.alexandriava.gov/resource/ci25-wg4g.json?$select=incident\_id,offense,location,block\_address,report\_datetime,latitude,longitude&$order=report\_datetime%20DESC&$limit=100";

const resp = await fetch(url);

const data = await resp.json();

return data;

}

function badge(sev){ return sev === "critical" ? "🔴" : sev === "major" ? "🟠" : "🟢"; }

function classify(offense=""){

const o = offense.toUpperCase();

if (/(WEAPONS|HOMICIDE|SHOTS|ROBBERY|ASSAULT)/.test(o)) return "critical";

if (/(BURGLARY|ARSON)/.test(o)) return "major";

return "info";

}

function plotIncidents(map, rows){

rows.forEach(r => {

const lat = parseFloat(r.latitude), lon = parseFloat(r.longitude);

if (isNaN(lat) || isNaN(lon)) return;

const sev = classify(r.offense||"");

const m = L.marker([lat, lon]).addTo(map);

m.bindPopup(`${badge(sev)} ${r.offense || "Incident"}<br>${r.block\_address || ""}<br>${new Date(r.report\_datetime).toLocaleString()}`);

});

}

**7) EAS logic (recommended)**

* **Trigger info banner** for any incident with critical or major in the last X minutes within your AOI.
* **Suppress duplicates** by incident\_id. Update on re-fetch if report\_datetime changes.
* **Avoid over-alerting:** use a rolling window (e.g., last 60–120 min) and cap pushes.

**8) Notes / gotchas**

* **Latency:** City open data may be near-real-time but not instantaneous; treat as *situational awareness*, not paging.
* **Schema drift:** Columns can change names; keep a light mapping layer.
* **App token:** If you hit rate limits, request a Socrata app token and add X-App-Token.

**USGS NWIS — Real-time gauges for Alexandria**

**Key Alexandria stations (USGS site IDs)**

* **01653000 — Cameron Run at Alexandria, VA** (streamflow & stage). [waterdata.usgs.gov+2waterdata.usgs.gov+2](https://waterdata.usgs.gov/monitoring-location/01653000/?utm_source=chatgpt.com)
* **01652500 — Fourmile Run at Alexandria, VA** (streamflow & stage). [waterdata.usgs.gov+2waterdata.usgs.gov+2](https://waterdata.usgs.gov/monitoring-location/01652500/?utm_source=chatgpt.com)
* **01653445 — Hunting Creek at Hwy 1 at Alexandria, VA** (stage). [waterdata.usgs.gov+1](https://waterdata.usgs.gov/monitoring-location/01653445/?utm_source=chatgpt.com)
* **01652590 — Potomac River at Alexandria, VA** (tidal water level/water quality site record). [waterdata.usgs.gov+1](https://waterdata.usgs.gov/monitoring-location/01652590/?utm_source=chatgpt.com)
* **0165258890 — Potomac River at Cameron St Dock (Alexandria, VA)** (tidal water level). [waterdata.usgs.gov+1](https://waterdata.usgs.gov/monitoring-location/0165258890/?utm_source=chatgpt.com)
* Also nearby tide/water-level points at **101 King St** and **100 Madison St** (USGS IDs 384816077022500, 384844077021800). [waterdata.usgs.gov+1](https://waterdata.usgs.gov/monitoring-location/384816077022500/?utm_source=chatgpt.com)

Common parameters: **00060** (Discharge, cfs), **00065** (Gage height, ft). Most flood logic keys off **00065**.

**1) Postman — Instantaneous Values (IV) JSON**

USGS Water Services IV endpoint:

https://waterservices.usgs.gov/nwis/iv/?format=json&sites=SITE\_ID&parameterCd=00065,00060

**Examples (copy/paste)**

**Cameron Run (01653000) — stage + flow**

GET https://waterservices.usgs.gov/nwis/iv/?format=json&sites=01653000&parameterCd=00065,00060

**Fourmile Run (01652500) — stage + flow**

GET https://waterservices.usgs.gov/nwis/iv/?format=json&sites=01652500&parameterCd=00065,00060

**Hunting Creek (01653445) — stage**

GET https://waterservices.usgs.gov/nwis/iv/?format=json&sites=01653445&parameterCd=00065

**Potomac at Alexandria (01652590) — stage (if available)**

GET https://waterservices.usgs.gov/nwis/iv/?format=json&sites=01652590&parameterCd=00065

**Cameron St Dock (0165258890) — tidal level**

GET https://waterservices.usgs.gov/nwis/iv/?format=json&sites=0165258890&parameterCd=00065

**Headers:** none required.  
**Auth:** none.  
**Notes:** &siteStatus=all can be added to include inactive sites; &period=P1D (last day) or &startDT=...&endDT=... for ranges.

**2) Sample JSON (trimmed)**

{

"value": {

"timeSeries": [

{

"sourceInfo": { "siteCode": [{"value":"01653000"}], "siteName":"Cameron Run at Alexandria, VA" },

"variable": { "variableCode":[{"value":"00065"}], "variableName":"Gage height, ft" },

"values": [{ "value":[ {"value":"3.45","dateTime":"2025-09-17T21:10:00.000-04:00"} ]}]

}

]

}

}

**3) Postman Tests — turn IV data into simple numbers**

const j = pm.response.json();

const ts = j.value?.timeSeries || [];

const flat = ts.map(s => ({

site: s.sourceInfo?.siteCode?.[0]?.value,

name: s.sourceInfo?.siteName,

param: s.variable?.variableCode?.[0]?.value, // 00065 or 00060

label: s.variable?.variableName,

reading: s.values?.[0]?.value?.slice(-1)?.[0]?.value,

time: s.values?.[0]?.value?.slice(-1)?.[0]?.dateTime

}));

pm.collectionVariables.set("USGS\_IV", JSON.stringify(flat));

pm.test("Got timeSeries", () => pm.expect(ts.length).to.be.above(0));

**4) Front-end usage (very small fetch)**

<script>

async function getUSGS(site, params="00065"){

const url = `https://waterservices.usgs.gov/nwis/iv/?format=json&sites=${site}&parameterCd=${params}`;

const j = await (await fetch(url)).json();

const ts = j.value?.timeSeries || [];

return ts.map(s => {

const v = s.values?.[0]?.value?.slice(-1)?.[0] || {};

return {

site: s.sourceInfo?.siteCode?.[0]?.value,

name: s.sourceInfo?.siteName,

code: s.variable?.variableCode?.[0]?.value,

label: s.variable?.variableName,

value: Number(v.value),

time: v.dateTime

};

});

}

// Example:

getUSGS("01653000","00065,00060").then(console.log);

</script>

**5) EAS logic (recommended)**

* **Trigger “Flood Watch” banner** when **Cameron Run (01653000) gage height** exceeds thresholds from your flood plan; similarly for **Fourmile Run (01652500)** and **Hunting Creek (01653445)**.
* **Tidal back-up risk:** watch **Potomac tidal levels** at **0165258890** (Cameron St Dock) and nearby river points; rapid rises can compound local runoff.
* **Debounce alerts:** require N consecutive readings over threshold (e.g., 2–3 samples) before pushing a public alert.

**PulsePoint — Fire/EMS Alerts (Alexandria)**

**0) What it is**

**PulsePoint is a public safety platform that shares active Fire/EMS incidents and, in some regions, CPR-needed alerts.**

* **Accessible via the PulsePoint Respond mobile app and Respond for Web (**[**https://web.pulsepoint.org**](https://web.pulsepoint.org?utm_source=chatgpt.com)**).**
* **A JSON/REST API exists but is only available to official government/agency partners.**

**1) Availability for Alexandria**

* **✅ Nearby Northern Virginia agencies such as Fairfax County, Arlington County, Loudoun County, and Prince William County are available on PulsePoint.**
* **❌ City of Alexandria Fire Department is *not available* as a PulsePoint agency.**
* **Therefore, no Alexandria-specific PulsePoint data can be accessed at this time.**

**2) What data PulsePoint provides (where available)**

* **Incident list (Fire, Medical, Traffic, Hazmat, Rescue).**
* **Incident details: type/subtype, time dispatched, address (approximate), responding units.**
* **Map view of incident locations.**
* **In some regions: CPR-needed alerts and AED locations.**

**3) How to access (for participating agencies)**

* **Public: View incidents in the app or web portal. Agencies are selected by their PulsePoint ID.**
* **Developers/Partners: Agency must request API access from PulsePoint; data is not open for public API use.**

**4) EAS Relevance**

* **Direct Alexandria coverage: Not available.**
* **Regional coverage: You can embed or monitor neighboring agencies (Fairfax, Arlington, etc.) for situational awareness.**
* **Use case: Supplementary public-facing awareness tool — *not* suitable as a core API feed unless Alexandria joins the program.**

**✅ Conclusion: PulsePoint currently does not provide Fire/EMS incident data for Alexandria City. It can still provide regional context via neighboring agencies, but is not a usable API source for Alexandria-specific EAS integration.**

**Broadcastify & IPN — Alexandria City**

**0) What it is**

* **Broadcastify streams live radio traffic (Fire/EMS, Police, etc.) from scanners.**
* **IPN (Incident Page Network) is a subscription service that provides structured incident alerts; some of those alerts are displayed alongside Broadcastify feeds.**

**1) Availability for Alexandria**

* **✅ Alexandria City Fire and EMS feed is live and publicly accessible.**
* **✅ Additional feeds include N3ST Repeater System and Washington DC Interoperability.**
* **✅ IPN incident summaries are shown for Northern Virginia incidents (major accidents, trauma, fires).**
* **❌ No structured JSON or open API for Alexandria-specific radio traffic (audio only). IPN structured alerts require a paid subscription.**

**2) What data is provided**

* **Broadcastify (free):**
  + **Live audio stream of Fire/EMS dispatch in Alexandria.**
  + **Player formats: Web (HTML5), external media players (iTunes, Winamp, etc.).**
  + **Status (Online/Offline), listener count.**
* **IPN (with subscription):**
  + **Structured alerts with incident type, location, time, short description (e.g., “Major Accident — Pedestrian Struck — West Falls Church — transported with injuries”).**
  + **Delivered via pager, SMS, email, or IPN console.**

**3) How to access**

* **Broadcastify (Alexandria):**
  + **Web: Broadcastify Alexandria City Fire and EMS**
  + **Free to stream; archives available with Premium subscription.**
* **IPN:**
  + **Subscription required (incidentpage.net).**
  + **Alerts delivered via member portal or notifications; not public API.**

**4) EAS Relevance**

* **Broadcastify:**
  + **Great for real-time human situational awareness.**
  + **Not machine-readable (audio-only) unless paired with speech-to-text.**
* **IPN:**
  + **Provides structured, text-based incident alerts (ideal for dashboards).**
  + **Paid service, no free API.**

**✅ Conclusion:**

* **Broadcastify gives Alexandria live Fire/EMS radio streams (free, audio-only).**
* **IPN adds structured incident alert text feeds, but requires a subscription and is not open data.**
* **Together, they provide valuable but semi-structured data for EAS — supplementary, not a core open API source.**