Fast
Hazard

FastFlood Global

API Documentation

Date : 01-03-2025
Authors : B. van den Bout, F.J. Koloparambil, K. van Roon, D.T. Meijvogel
Contact : info@fasthazard.com
Version : 1.0

FastFlood API Documentation

Overview

The FastFlood API lets you run hydrodynamic flood simulations and related terrain-processing workflows by submitting a single JSON payload. The API is designed to support:

- rapid hazard-map generation
- forecast-based flood modelling
- dam-break and custom boundary-condition scenarios
- DEM acquisition and preprocessing
- calibration workflows
- exposure/intensity processing
- drainage-focused preprocessing

The API input is based on Pydantic models and accepts both simple scalar values and structured geospatial inputs such as GeoTIFF and GeoJSON.

Quick Start

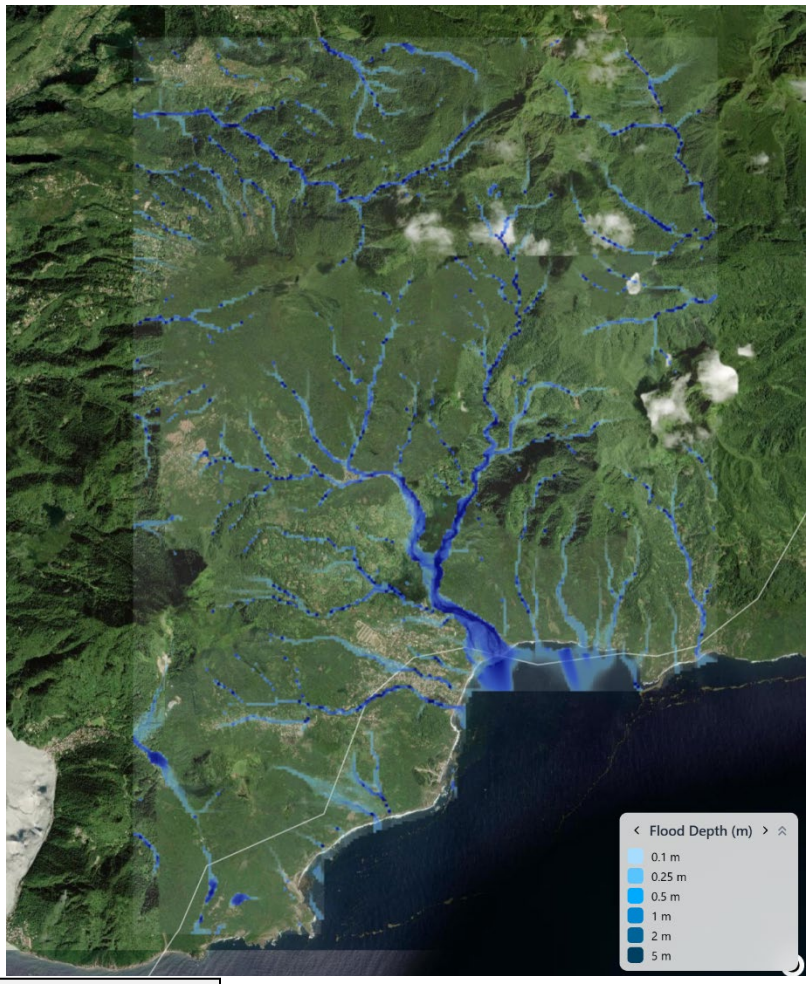
The fastest way to use the FastFlood API is to send a single JSON payload with an automatic setup option such as autohazard or autoforecast.

1. Quick hazard map

Use this for a simple hazard run based on location, scenario, return period, and resolution.


```
{
 "autohazard": {
  "bbox": [
   5.935549136760547,
   52.212453671885775,
   5.986843342154979,
   52.18101278076109
  ],
  "return_period": 100,
  "scenario": "flashfloodfluvialflood",
  "resolution": "medium"
 }
}
```

This will automatically set up a hazard simulation for the selected area.



Notes

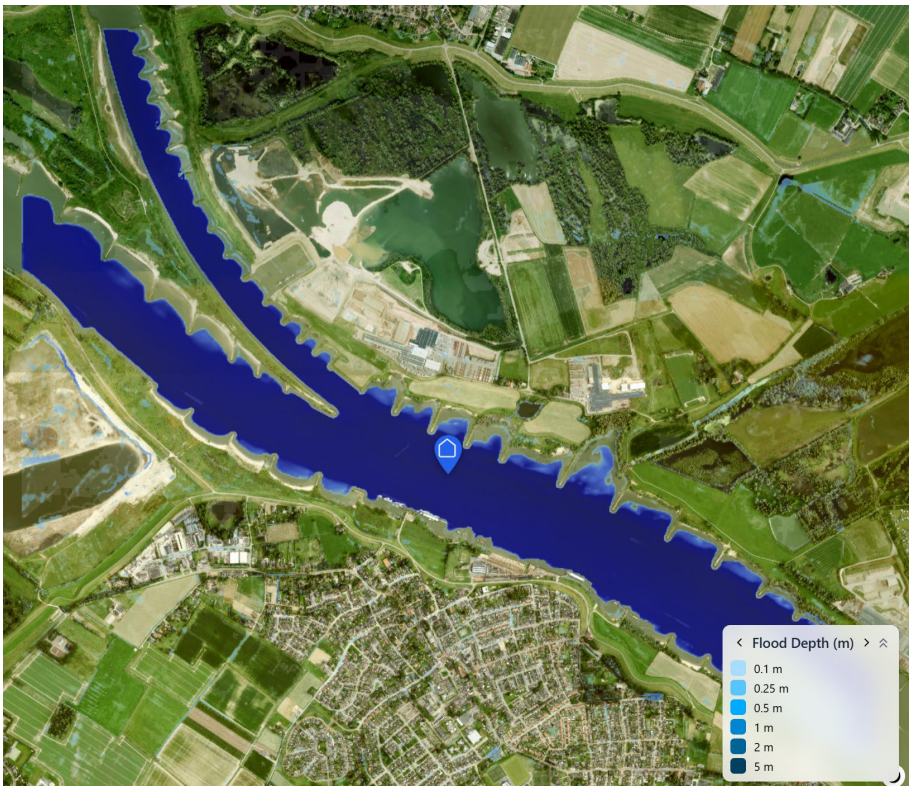
- bbox defines the model area.
- resolution is usually low, medium, or high.
- scenario can combine types like flashflood, fluvialflood, coastal, or dambreak.
- For most first-time users, autohazard is the easiest starting point.

2. Quick forecast run

Use this when you want to run the latest available forecast.

```
{
 "autoforecast": {
  "bbox": [
   6.016699870346243,
   51.88674039341924,
   6.067621750245864,
   51.85529950229455
  ],
  "date": "latest",
  "resolution": "high",
  "scenario": "flashfloodfluvialflood",
  "start_time": 0,
  "end_time": 150
 }
}
```

This will automatically configure a forecast-based simulation using the latest available forecast data.

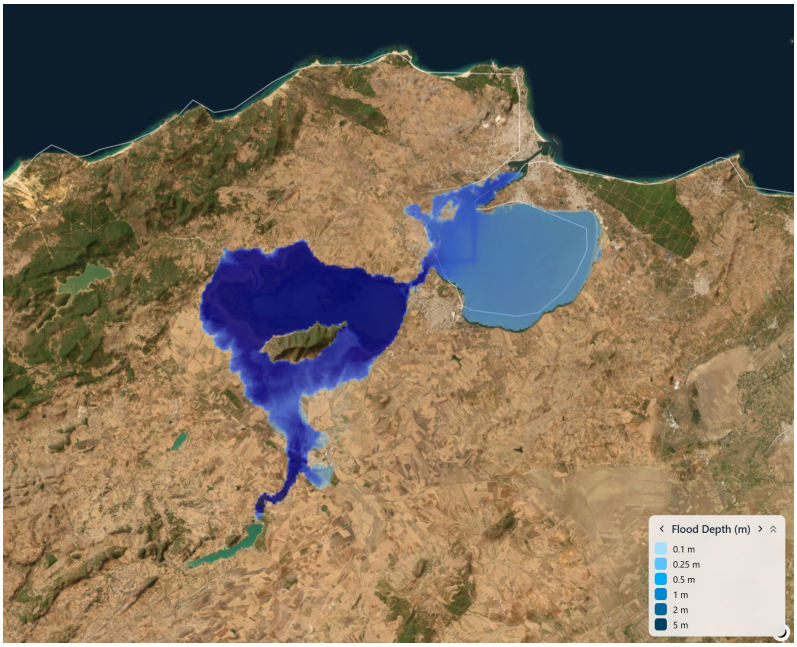

Flood Depth (m)
0.1 m
0.25 m
0.5 m
1 m
2 m
5 m

3. Simple custom event

Use this when you want a manual inflow event with automatically downloaded terrain.

```
{
 "autodem": {
  "bbox": [
   9.282646478085933,
   37.2566357736135,
   9.937936380035964,
   36.733263228301404
  ],
  "resolution": "medium"
 },
 "qin": {
  "latitude": 36.99494950095745,
  "longitude": 9.610291429060949,
  "flow_rate": 3826.1831744142314
 },
 "dur": 1,
 "rain": 0
}
```

This creates a short event with a manual discharge inflow and no rainfall.



Response format

All endpoints return a standard response envelope:

```
{
  "status_code": 200,
  "message": "Model executed successfully",
  "data": {
    "files": [
      {
        "name": "whout.tif",
        "type": "ModelOutputFile",
        "href": "https://..."
      }
    ]
  },
  "errors": null
}
```

It will contain several files linked, that will persist for 24 hours on azure blob storage. Cloud optimized geotiffs can be visualized directly from these download locations using many libraries (e.g. MapLibreGL) built-in COG tif support using HTTP-Get Ranges requests.

Data name	Description	Type
whout.tif	Peak flood depth at every location during the event in meter depth	Cloud optimized Geotiff
qout.tif	Peak flow rates at every location during the event in m3/s	Cloud optimized Geotiff
dem.tif	Elevation raster in meters above sea level, merging seamlessly national LIDAR and global elevation datasets	Cloud optimized Geotiff
vel.tif	Peak flow velocity of the water in m3/s	Cloud optimized Geotiff
velx.tif	Peak flow velocity of the water in m3/s x-component for directional visualizing	Cloud optimized Geotiff
vely.tif	Peak flow velocity of the water in m3/s y-component for directional visualizing	Cloud optimized Geotiff
channel.geojson	Channel polylines, with attribute for drainage area, as well as flow rates	GeoJSON polylines
bcond.geojson	Boundary condition locations, points with discharge attribute.	GeoJSON points
swd.geojson	Storm water drainage system, estimated in detail. Often empty in default settings as gridded estimates are used	GeoJSON Polylines
stdout.txt	Text output from the model, hydro info.	Text file
stderr.txt	API Error output, typically empty.	Text file

Standard response schema

```python
class StandardResponse(BaseModel, Generic[T]):
    status_code: int
    message: str
    data: Optional[T] = None
    errors: Optional[Any] = None
```

Response fields

Field	Type	Description
`status_code`	integer	HTTP-style status code for the request result.
`message`	string	Human-readable summary of the result.
`data`	object or `null`	Response payload. For successful model runs this typically contains output files.
`errors`	any or `null`	Validation or execution errors. Usually `null` on success.

Core input model

The main simulation payload is represented by `ModelParameters`.

The API follows a flexible pattern:

- You may provide data manually, such as your own DEM, rainfall, boundary conditions, or land-surface modifiers.
- You may use automatic setup helpers such as `autohazard`, `autoforecast`, `autodem`, or `d_dem`.
- Automatic options can be overridden by explicitly providing manual inputs.

Typical design philosophy

A request usually starts with one of these approaches:

1. **Quick hazard map** using `autohazard`
2. **Forecast-based simulation** using `autoforecast` or `forecast`
3. **Manual event setup** using custom DEM, rainfall, and boundary conditions
4. **Terrain preparation** using `autodem` or `d_dem`

Coordinate conventions

Bounding boxes (BBox)

Several models use a bbox field.

In user-facing JSON, this is provided as an array of four coordinates:

```
[min_x, max_y, max_x, min_y]
```

Based on the examples, this corresponds to:

- left / west boundary
- top / north boundary
- right / east boundary
- bottom / south boundary

In geographic inputs this is typically:

```
[min_longitude, max_latitude, max_longitude, min_latitude]
```

Example:

```
"bbox": [5.935549136760547, 52.212453671885775, 5.986843342154979, 52.1810127
8076109]
```

Point coordinates

For explicit point-based options such as `qin`, coordinate order is:

```
longitude, latitude
```

For GeoJSON, standard GeoJSON coordinate order applies:

```
[x, y] = [longitude, latitude]
```

Supported geospatial input styles

Many fields support either:

- an inline object
- a URL to a remote file

GeoTIFF URL

Example:

```
{
  "url": "https://example.com/file.tiff"
}
```

Used for fields such as:

- `dem`
- `rain`
- `inf`
- `man`
- `sm`
- `mult_chan`
- `i1`, `i2`, `i3`

Inline GeoJSON

Example line input:

```
{
  "type": "geojson",
  "geojson": {
    "type": "FeatureCollection",
    "features": []
  }
}
```

GeoJSON URL

```
{
  "type": "url",
  "url": "https://example.com/file.geojson"
}
```

This pattern is used for:

- `barriers`
- `adaptations`
- `bcond`
- `swd`
- `culverts`
- `obs_flood_shp`
- `obs_q`
- `obs_wh`

Simulation request model

`ModelParameters`

This is the main payload for flood simulations.

Domain and terrain

`dem`

Provide a DEM as a GeoTIFF URL. This defines model extent and resolution.

Type:

- `GeoTiffUrl | null`

Example:

```
"dem": {
  "url": "https://example.com/dem.tif"
}
```

`dfdtm`

Optional DTM filter applied to background elevation datasets.

Type:

- `DTMFilter | null`

Example:

```
"dfdtm": {
  "intensity": 1.0,
  "iterations": 50
}
```

`d_dem`

Automatically download a DEM for a specified area and resolution.

Type:

- `DEMOption | null`

Example:

```
"d_dem": {
  "elevation_model_name": "cop30",
  "resolution": "20m",
  "bbox": [-6827737.551573, 1722546.650921, -6822960.237306, 1714291.451866]
}
```

autodem

Automatically obtain and preprocess DEM data for an area, including higher-resolution national datasets where available.

Type:

- `AutoDEMOption | null`

Example:

```
"autodem": {
  "bbox": [9.282646478085933, 37.2566357736135, 9.937936380035964, 36.733263228301404],
  "resolution": "medium"
}
```

Automatic scenario setup

autohazard

Creates a hazard map setup automatically based on:

- extent (bbox)
- scenario string
- return period
- resolution

This option can populate many required model settings automatically.

Type:

- `HazardOption | null`

Fields:

Field	Type	Description
`scenario`	string	Concatenated scenario keywords. Can include combinations of `flashflood`, `fluvialflood`, `coastal`, `dambreak`, climate scenario strings such as `ssp124`, `ssp245`, `ssp460`, `ssp585`, and time-period values between 2020 and 2100.

Field	Type	Description
`return_period`	integer	Return period in years. Must be between 2 and 1000.
`bbox`	array	Simulation bounding box.
`resolution`	string	`low`, `medium`, or `high`. Invalid values default to `low`.

Example:

```
{
  "autohazard": {
    "bbox": [
      5.935549136760547,
      52.212453671885775,
      5.986843342154979,
      52.18101278076109
    ],
    "return_period": 100,
    "scenario": "flashfloodfluvialflood",
    "resolution": "medium"
  }
}
```

Use this when you want a fast hazard product with minimal manual setup.

`autoforecast`

Creates a forecast-based simulation using a selected forecast date and lead-time window.

Type:

- `ForecastOption | null`

Fields:

Field	Type	Description
`scenario`	string	Same scenario logic as `autohazard`.
`date`	string	Forecast date in `DD-MM-YYYY` format, or `latest`.
`start_time`	integer	Start of forecast window in hours after forecast initialization.
`end_time`	integer	End of forecast window in hours after forecast initialization.

Field	Type	Description
`bbox`	array	Simulation bounding box.
`resolution`	string	`low`, `medium`, or `high`.

Example:

```
{
  "autoforecast": {
    "bbox": [
      6.016699870346243,
      51.88674039341924,
      6.067621750245864,
      51.85529950229455
    ],
    "date": "latest",
    "resolution": "high",
    "scenario": "flashfloodfluvialflood",
    "start_time": 0,
    "end_time": 150
  }
}
```

Use this for rapid flood forecasting based on operational forecast data.

Hydrologic and hydraulic forcing

`rain`

Rainfall input. Can be:

- a numeric event-average intensity in `mm/hour`
- a GeoTIFF URL

Type:

- `float | GeoTiffUrl | null`

Examples:

```
"rain": 12.5
```

```
"rain": {
  "url": "https://example.com/rainfall.tif"
}
```

`forecast`

Activates rainfall input from forecast data.

Type:

- `RainForecastOption | null`

Fields:

Field	Type	Description
`forecast_date`	string	Date or `latest`.
`start_time`	integer	Start hour relative to forecast origin.
`end_time`	integer	End hour relative to forecast origin.

Example:

```
"forecast": {
  "forecast_date": "latest",
  "start_time": 0,
  "end_time": 24
}
```

designstorm

Automatically sets rainfall intensity from a design-storm dataset.

Type:

- `DesignStorm | null`

Fields:

Field	Type	Description
`return_period`	integer	Valid values: `2, 5, 10, 20, 40, 50, 100, 200, 500, 1000`
`duration`	integer	Valid values: `3, 6, 12, 24, 48, 72, 120, 240` hours

Example:

```
"designstorm": {
  "return_period": 100,
  "duration": 24
}
```

dur

Event duration in hours.

Type:

- `float | null`

Example:

```
"dur": 6
```

ocean

Ocean boundary condition in meters.

Type:

- `float | null`

Example:

```
"ocean": 1.2
```

qin

Single discharge boundary condition using explicit coordinates and flow rate.

Type:

- `DischargeBoundaryConditionOption | null`

Fields:

Field	Type	Description
`latitude`	float	Latitude in WGS84 / EPSG:4326
`longitude`	float	Longitude in WGS84 / EPSG:4326
`flow_rate`	float	Discharge in m^3/s

Example:

```
"qin": {
  "latitude": 52.0,
  "longitude": 5.9,
  "flow_rate": 250.0
}
```

bcond

Multiple discharge boundary conditions through GeoJSON points.

Type:

- `GeoJsonPointsUrl | GeoJsonPoints | null`

Expected point attributes:

Attribute	Type	Description
`discharge`	number	Discharge in m^3/s
`area`	number	Drainage area in km^2
`block_region`	number	Between 0 and 1. Indicates whether incoming surrounding boundary conditions should be auto-blocked. Defaults to 1 if missing.

Inline example:

```
"bcond": {
  "type": "geojson",
  "geojson": {
    "type": "FeatureCollection",
    "features": [
      {
        "type": "Feature",
        "properties": {
          "discharge": 3826.1831744142314
        },
        "geometry": {
          "type": "Point",
          "coordinates": [9.610291429060949, 36.99494950095745]
        }
      }
    ]
  }
}
```

Surface properties and modifiers

`inf`

Infiltration rate in `m/hour`.

Can be either:

- numeric value
- GeoTIFF URL

Type:

- `float | GeoTiffUrl | null`

Examples:

```
"inf": 0.005
```

```
"inf": {
  "url": "https://example.com/infiltration.tif"
}
```

man

Manning roughness coefficient as scalar or GeoTIFF.

Type:

- `float | GeoTiffUrl | null`

sm

Soil moisture as scalar or GeoTIFF.

Type:

- `float | GeoTiffUrl | null`

barriers

Barrier geometries as line GeoJSON.

Type:

- `GeoJsonLineUrl | GeoJsonLine | null`

Expected geometry:

- `LineString`

Expected attribute:

Attribute	Description
`height`	Barrier height in meters. Defaults to `10` if missing.

Example:

```
"barriers": {
  "type": "url",
  "url": "https://example.com/barriers.geojson"
}
```

adaptations

Adaptation or nature-based-solution features using polygons or polylines.

Type:

- `GeoJsonPolygonUrl | GeoJsonPolygon | null`

Supported attributes include:

Attribute	Description
`elevation`	Elevation modification in meters
`manning`	Surface roughness coefficient
`inf`	Infiltration in mm/hour
`precipitation`	Rainfall in mm/hour

Negative values indicate that the parameter should be ignored.

Example:


```
"adaptations": {
  "type": "geojson",
  "geojson": {
    "type": "FeatureCollection",
    "features": [
      {
        "type": "Feature",
        "properties": {
          "manning": "0.12",
          "id": "conservation_forest",
          "mode": "polygon"
        },
        "geometry": {
          "type": "Polygon",
          "coordinates": [[[5.95087291,52.20426972],[5.948076416,52.202513014
],[5.948859435,52.200130637],[5.954088876,52.192477034],[5.970056858,52.19705
3928],[5.970923769,52.19983069],[5.971007664,52.203841262],[5.967260361,52.20
792002],[5.962142778,52.209170992],[5.95087291,52.20426972]]]
        }
      }
    ]
  }
}
```


This is a good option for representing restoration measures, forest patches, retention zones, or engineered interventions.

Sewer and culvert network inputs

`d_swd`

Automatically estimate storm-water drainage in the region.

Type:

- `bool | null`

swd

Provide storm-water drainage system input as line GeoJSON.

Type:

- `GeoJsonLineUrl | GeoJsonLine | null`

culverts

Provide culverts as line GeoJSON.

Type:

- `GeoJsonLineUrl | GeoJsonLine | null`

swd_eff

Sewer drain efficiency.

Type:

- `float | null`

sdw_set

Sewer drain setting.

Type:

- `float | null`

channelmv

Drainage-area threshold in km^2 above which a channel behaves as an infinite drain.

Type:

- `float | null`

Automation toggles

d_lu

Automatically download land-use information.

Type:

- `bool | null`

d_inf

Automatically download infiltration information.

Type:

- `bool | null`

pseudomercator

Project output grids to Web/Pseudo Mercator for easier visualization.

Type:

- `bool | null`

autoprocessgrids

Use larger-domain grids automatically to feed model processes into the chosen domain.

Type:

- `bool | null`

noautobcond

Prevent automatic boundary-condition placement during auto-hazard or auto-forecast runs.

Type:

- `bool | null`

noarf

Disable automatic area reduction factors for design events.

Type:

- `bool | null`

This is useful for large-scale simultaneous modelling of flash floods where rainfall reduction by area is not desired.

Multipliers

These fields adjust model inputs or process strengths.

Field	Type	Description
`mult_man`	float	Manning multiplier
`mult_inf`	float	Infiltration multiplier
`mult_rain`	float	Rainfall multiplier
`mult_chan`	float or GeoTIFF	Channel-dimension multiplier
`mult_qbcond`	float	Boundary-condition discharge multiplier
`mult_runoff`	float	Runoff multiplier

Field	Type	Description
`mult_fpl`	float	Flood-protection-level multiplier

Example:

```
{
  "mult_man": 1.1,
  "mult_inf": 0.85,
  "mult_rain": 1.2
}
```

Calibration

`cal`

Enable automatic calibration.

Type:

- `bool | null`

`cal_man`, `cal_inf`, `cal_conc`, `cal_chan`

Calibration ranges for selected parameters.

Type:

- `CalibrationOption | null`

Each accepts:

Field	Type	Description
`count`	integer	Number of parameter variations
`lower_limit`	float	Lower bound
`upper_limit`	float	Upper bound

Example:

```
"cal_man": {
  "count": 6,
  "lower_limit": 0.5,
  "upper_limit": 1.5
}
```

Observational inputs

These support validation and calibration workflows.

Field	Type	Description
obs_flood	float or GeoTIFF	Observed flood metric
obs_flood_shp	polygon GeoJSON or URL	Observed flood extent shapes
obs_q	point GeoJSON or URL	Observed discharge points
obs_wh	point GeoJSON or URL	Observed water-height points
d_obs_flood	float	Weight/setting for flood observations
d_obs_q	float	Weight/setting for discharge observations

Helper models

DTMFilter

Used to smooth or filter terrain input.

```
{
  "intensity": 1.0,
  "iterations": 50
}
```

Field	Type	Description
intensity	float	Filter strength
iterations	integer	Number of solver iterations

DEMOption

```
{
  "elevation_model_name": "cop30",
  "resolution": "20m",
  "bbox": [-6827737.551573, 1722546.650921, -6822960.237306, 1714291.451866]
}
```

AutoDEMOption

```
{
  "bbox": [-6827737.551573, 1722546.650921, -6822960.237306, 1714291.451866],
  "resolution": "high"
}
```

RainForecastOption

```
{
  "forecast_date": "latest",
  "start_time": 0,
```

```
  "end_time": 24
}
```

`DesignStorm`

```
{
  "return_period": 10,
  "duration": 3
}
```

`DischargeBoundaryConditionOption`

```
{
  "latitude": 52.0,
  "longitude": 5.9,
  "flow_rate": 100.0
}
```

`CalibrationOption`

```
{
  "count": 6,
  "lower_limit": 0.5,
  "upper_limit": 1.5
}
```

Other request models

`ModelExposureParameters`

Used for exposure or impact workflows driven by intensity rasters.

Fields

Field	Type	Required	Description
`i1`	`GeoTiffUrl`	Yes	Intensity parameter 1. Defines model domain.
`i2`	`GeoTiffUrl`	No	Intensity parameter 2. Should match domain.
`i3`	`GeoTiffUrl`	No	Intensity parameter 3. Should match domain.
`i1_impactthreshold`	float	No	Impact threshold for intensity parameter 1.

Example

```
{
  "i1": { "url": "https://example.com/depth.tif" },
  "i2": { "url": "https://example.com/velocity.tif" },
  "i1_impactthreshold": 0.15
}
```

`ModelDrainageParameters`

Focused terrain/drainage preprocessing model.

Fields

Field	Type	Description
`dem`	GeoTIFF URL	Provide DEM directly
`dfdtm`	`DTMFilter`	Optional filter
`d_dem`	`DEMOption`	Download DEM automatically
`autodem`	`AutoDEMOption`	High-level automatic DEM workflow

Example

```
{
  "autodem": {
    "bbox": [5.93, 52.21, 5.99, 52.18],
    "resolution": "medium"
  },
  "dfdtm": {
    "intensity": 0.8,
    "iterations": 40
  }
}
```

`ModelDEMParameters`

A DEM-oriented request model, structurally similar to `ModelDrainageParameters`, used when the focus is on terrain preparation.

Example

```
{
  "d_dem": {
    "elevation_model_name": "cop30",
    "resolution": "20m",
    "bbox": [-6827737.551573, 1722546.650921, -6822960.237306, 1714291.451866
]
```

```
  }
}
```

Practical examples

1. Quick hazard map

This is the simplest way to generate a hazard map.

```
{
  "autohazard": {
    "bbox": [
      5.935549136760547,
      52.212453671885775,
      5.986843342154979,
      52.18101278076109
    ],
    "return_period": 100,
    "scenario": "flashfloodfluvialflood",
    "resolution": "medium"
  }
}
```

What it does

- defines the simulation domain from the bounding box
- automatically configures the selected hazard scenario
- uses a 100-year return period
- produces a medium-resolution output set

Best use case

Use this for first-pass screening and rapid hazard assessments.

---

2. Forecast-based flood simulation

```
{
  "autoforecast": {
    "bbox": [
      6.016699870346243,
      51.88674039341924,
      6.067621750245864,
      51.85529950229455
    ],
    "date": "latest",
```

```
    "resolution": "high",
    "scenario": "flashfloodfluvialflood",
    "start_time": 0,
    "end_time": 150
  }
}
```

What it does

- selects the latest forecast
- uses forecast rainfall between hour 0 and hour 150
- runs at high resolution
- applies the chosen combined scenario

Best use case

Use this for operational forecasting and early-warning workflows.

---

3. Dam-break style event with manual inflow boundary

```
{
  "autodem": {
    "bbox": [
      9.282646478085933,
      37.2566357736135,
      9.937936380035964,
      36.733263228301404
    ],
    "resolution": "medium"
  },
  "bcond": {
    "type": "geojson",
    "geojson": {
      "type": "FeatureCollection",
      "features": [
        {
          "type": "Feature",
          "properties": {
            "discharge": 3826.1831744142314
          },
          "geometry": {
            "type": "Point",
            "coordinates": [
              9.610291429060949,
              36.99494950095745
            ]
          }
        }
```

```
      ]
    }
  },
  "duration": 1,
  "rain": 0
}
```

Notes

The source model actually defines event duration as `dur`. If your API layer expects `dur`, prefer this canonical form:

```
{
  "autodem": {
    "bbox": [9.282646478085933, 37.2566357736135, 9.937936380035964, 36.73326
3228301404],
    "resolution": "medium"
  },
  "bcond": {
    "type": "geojson",
    "geojson": {
      "type": "FeatureCollection",
      "features": [
        {
          "type": "Feature",
          "properties": {
            "discharge": 3826.1831744142314
          },
          "geometry": {
            "type": "Point",
            "coordinates": [9.610291429060949, 36.99494950095745]
          }
        }
      ]
    }
  },
  "dur": 1,
  "rain": 0
}
```

What it does

- generates terrain automatically
- injects an inflow hydrograph-like boundary using a point discharge
- sets rainfall to zero
- simulates a short event duration suitable for a release or breach scenario

---

4. Hazard scenario with nature-based solutions


```
{
  "autohazard": {
    "bbox": [
      5.942542346939466,
      52.21722428551637,
      5.99384205847215,
      52.185783394391684
    ],
    "return_period": 100,
    "scenario": "flashfloodfluvialflood",
    "resolution": "medium"
  },
  "adaptations": {
    "type": "geojson",
    "geojson": {
      "type": "FeatureCollection",
      "features": [
        {
          "id": "595aeec2-1b59-4e15-a347-5274285fc3c1",
          "type": "Feature",
          "properties": {
            "ksat": "119",
            "man": "",
            "elev": "",
            "rain": "",
            "id": "conservation_forest",
            "manning": "0.12",
            "mode": "polygon"
          },
          "geometry": {
            "type": "Polygon",
            "coordinates": [
              [
                [5.95087291, 52.20426972],
                [5.948076416, 52.202513014],
                [5.948859435, 52.200130637],
                [5.954088876, 52.192477034],
                [5.970056858, 52.197053928],
                [5.970923769, 52.19983069],
                [5.971007664, 52.203841262],
                [5.967260361, 52.20792002],
                [5.962142778, 52.209170992],
                [5.95087291, 52.20426972]
              ]
            ]
          }
        }
      ]
```

```
    }
  }
}
```

What it does

- creates a standard hazard run
- overlays adaptation geometry
- modifies at least one surface property inside the polygon, such as Manning roughness
- can be used to assess nature-based or land-management interventions

---

5. Manual rainfall event with custom DEM

```
{
  "dem": {
    "url": "https://example.com/dem.tif"
  },
  "rain": 25,
  "dur": 3,
  "man": 0.05,
  "inf": 0.002,
  "pseudomercator": true
}
```

Best use case

Use this when all key forcing layers are already available and you want full control over the setup.

---

6. Design storm with automatic DEM and output projection

```
{
  "autodem": {
    "bbox": [5.94, 52.21, 5.99, 52.18],
    "resolution": "high"
  },
  "designstorm": {
    "return_period": 50,
    "duration": 24
  },
  "mult_rain": 1.1,
  "pseudomercator": true
}
```

---

7. Calibration setup

```
{
  "autohazard": {
    "bbox": [5.94, 52.21, 5.99, 52.18],
    "return_period": 100,
    "scenario": "flashfloodfluvialflood",
    "resolution": "medium"
  },
  "cal": true,
  "cal_man": {
    "count": 6,
    "lower_limit": 0.7,
    "upper_limit": 1.4
  },
  "cal_inf": {
    "count": 5,
    "lower_limit": 0.5,
    "upper_limit": 1.8
  }
}
```

Output files

A successful simulation typically returns a list of files.

Example:

```
{
  "status_code": 200,
  "message": "Model executed successfully",
  "data": {
    "files": [
      {
        "name": "whout.tif",
        "type": "ModelOutputFile",
        "href": "https://..."
      },
      {
        "name": "qout.tif",
        "type": "ModelOutputFile",
        "href": "https://..."
      },
      {
        "name": "dem.tif",
        "type": "ModelOutputFile",
        "href": "https://..."
```

```
      },
      {
        "name": "vel.tif",
        "type": "ModelOutputFile",
        "href": "https://..."
      },
      {
        "name": "velx.tif",
        "type": "ModelOutputFile",
        "href": "https://..."
      },
      {
        "name": "vely.tif",
        "type": "ModelOutputFile",
        "href": "https://..."
      },
      {
        "name": "channel.geojson",
        "type": "ModelOutputFile",
        "href": "https://..."
      },
      {
        "name": "bcond.geojson",
        "type": "ModelOutputFile",
        "href": "https://..."
      },
      {
        "name": "swd.geojson",
        "type": "ModelOutputFile",
        "href": null
      },
      {
        "name": "runoff.tif",
        "type": "ModelOutputFile",
        "href": "https://..."
      },
      {
        "name": "stdout.txt",
        "type": "MetaData",
        "href": "https://..."
      },
      {
        "name": "stderr.txt",
        "type": "MetaData",
        "href": "https://..."
      }
    ]
  },
  "errors": null
}
```

Common outputs

File	Meaning
`whout.tif`	Flood depth raster in meters
`qout.tif`	Peak discharge raster in m^3/s
`dem.tif`	Terrain elevation raster in meters
`vel.tif`	Peak flow velocity magnitude in m/s
`velx.tif`	Peak velocity x-component
`vely.tif`	Peak velocity y-component
`channel.geojson`	River or channel network geometry
`bcond.geojson`	Water-source or boundary-condition geometry
`swd.geojson`	Storm-water drainage geometry
`runoff.tif`	Runoff raster
`stdout.txt`	Model run log
`stderr.txt`	API/runtime diagnostics

About `href`

- If `href` is a URL, the file is available for download.
- If `href` is `null`, that output exists logically but no downloadable file was generated or published.

Validation notes and implementation details

Boolean handling

Boolean fields are converted internally to lowercase strings:

- `true`
- `false`

Numeric coercion

Some numeric fields are passed to the execution backend as strings, especially for mixed-type fields such as:

- `inf`
- `rain`
- `man`
- `sm`
- `mult_chan`

Command-line serialization

Several helper models expose a `toCommandLineOption()` method. This means the API layer converts structured JSON into positional command-line arguments behind the scenes.

This is internal behavior, but it explains why fields like `bbox`, `designstorm`, `forecast`, and `qin` are grouped into dedicated option objects.

Recommended usage patterns

Fastest way to get started

Start with `autohazard`.

It is the most compact entry point and needs only:

- `bbox`
- `scenario`
- `return_period`
- `resolution`

Best option for operations

Use `autoforecast` when you need near-real-time or forecast-driven mapping.

Best option for research-grade custom runs

Provide:

- `dem`
- `rain` or `forecast`
- `inf`
- `man`
- `bcond`
- `adaptations`
- calibration and multiplier options as needed

Best option for intervention studies

Use `autohazard` plus `adaptations`.

Common pitfalls

1. Using `duration` instead of `dur`

The model definition uses `dur`. If your external examples or wrapper use `duration`, confirm whether your API translates it. The canonical field in the provided Pydantic model is `dur`.

2. Invalid design storm values

Only the listed return periods and durations are valid for `designstorm`.

3. Wrong GeoJSON geometry type

Make sure each field receives the expected geometry family:

- `barriers`, `swd`, `culverts`: lines
- `bcond`, `obs_q`, `obs_wh`: points
- `adaptations`, `obs_flood_shp`: polygons or compatible shapes

4. Missing attributes in GeoJSON

Some layers depend on attributes such as:

- `height`
- `discharge`
- `manning`
- `elevation`
- `block_region`

Missing attributes may trigger fallback defaults or reduce model realism.

5. Bounding-box order confusion

Be consistent with the documented bbox ordering used by your API wrapper.

Minimal request cookbook

Hazard

```
{
  "autohazard": {
    "bbox": [5.94, 52.21, 5.99, 52.18],
    "return_period": 100,
    "scenario": "flashflood",
    "resolution": "medium"
```

```
  }
}
```

Forecast

```
{
  "autoforecast": {
    "bbox": [5.94, 52.21, 5.99, 52.18],
    "date": "latest",
    "scenario": "flashfloodfluvialflood",
    "start_time": 0,
    "end_time": 72,
    "resolution": "high"
  }
}
```

Manual inflow event

```
{
  "autodem": {
    "bbox": [9.28, 37.25, 9.94, 36.73],
    "resolution": "medium"
  },
  "qin": {
    "latitude": 36.9949,
    "longitude": 9.6103,
    "flow_rate": 3826.18
  },
  "dur": 1,
  "rain": 0
}
```

DEM download only

```
{
  "autodem": {
    "bbox": [5.94, 52.21, 5.99, 52.18],
    "resolution": "high"
  }
}
```

Summary

The FastFlood API provides a compact but highly flexible simulation interface. The recommended way to think about it is:

- use **automatic options** for speed
- use **manual data inputs** for control

- use **GeoJSON and GeoTIFF** for spatial customization
- use **adaptations, calibration, and multipliers** for scenario analysis
- inspect returned file URLs to retrieve raster, vector, and log outputs

For most users, the best first call is a small `autohazard` request. From there, forecasting, intervention modelling, calibration, and advanced hydraulic forcing can be added incrementally.