DestinPQ Image Generation API — Quick Start & Reference

This guide shows how to create an image generation task and poll for the result. It's intentionally concise and copy-paste friendly.

Base URL

```
https://video-api.destinpq.com/api/v1
```

Authentication

> The API **expects authentication** in general. However, the deployed environment is configured so **Pratik can call it without auth**.

If/when auth is enabled for others, include your chosen auth header (e.g., `Authorization: Bearer <token>`). Details depend on your environment.

Absolute, Non-Negotiable Fields (must be sent exactly as below)

These fields **must always be present with these exact values** for image generation requests — **no changes at all**:

```json
{
  "task_type": "image",
  "provider": "replicate",
  "service_id": 4,
  "raw": true
}
```

Endpoint 1 — Create Image Task

POST `/creations`

Request Body (JSON)

```json
{
  "task_type": "image",
  "provider": "replicate",
  "service_id": 4,
  "raw": true,
  "input_data": {
    "prompt": "create an mockup design for a health app",
    "aspect_ratio": "9:16",
    "output_format": "png",
    "safety_filter_level": "block_only_high"
  }
}
```

`input_data` parameters

- `prompt` *(string, required)* — Text prompt for generation.
- `aspect_ratio` *(string, required)* — One of: `"1:1"`, `"9:16"`, `"16:9"`, `"3:4"`, `"4:3"`.
- `output_format` *(string, required)* — One of: `"png"`, `"jpg"`.
- `safety_filter_level` *(string, required)* — One of:
- `"block_only_high"`
- `"blow_low_and_above"` *(as provided)*
- `"block_medium_and_above"`

  Note: Use the spellings exactly as above. If you introduce new values, they will be rejected.

Example — cURL

```
curl -X POST \
  https://video-api.destinpq.com/api/v1/creations \
  -H 'Content-Type: application/json' \
  -d '{
    "task_type": "image",
    "provider": "replicate",
    "service_id": 4,
    "raw": true,
    "input_data": {
      "prompt": "create an mockup design for a health app",
      "aspect_ratio": "9:16",
      "output_format": "png",
      "safety_filter_level": "block_only_high"
    }
  }'
```

Example Successful Response (queued)

You'll receive an object with a unique `id` and `status: "pending"` (or similar queued state). Save the `id` — you'll use it to poll.

Endpoint 2 — Poll Task Status

GET `/creations/{task_id}?raw=true`

Call this **every ~2 seconds** until the task's `status` becomes `"completed"` (or a terminal error state). Example terminal states include `completed` or an error with `error_message`.

Example — cURL

```
curl "https://video-api.destinpq.com/api/v1/creations/7acdd14a-0d49-4944-
ba72-93073ee8543d?raw=true"
```

Completed Response (shape)

- `status`: `"completed"`
- `output_assets`: array with at least one asset
- `output_assets[0].url`: **Direct URL** to the generated image (download this)
- `asset_type`: `"image"`
- `mime_type`: e.g., `image/OutputFormat.PNG`
- `metadata.replicate_prediction`: upstream details (IDs, timing, logs)
- `local_image_url` / `local_thumbnail_url`: internal storage paths (optional for your workflow)

Step 3 — Download the Image

Once the polling response shows `status: "completed"`, download the first asset:

```
GET  output_assets[0].url
```

Save it using the file extension consistent with `output_format` (`.png` or `.jpg`).

End-to-End Examples

Node.js (fetch) — Create, Poll, Download

```
import fs from 'node:fs/promises';
import path from 'node:path';

const BASE_URL = 'https://video-api.destinpq.com/api/v1';

async function sleep(ms) { return new Promise(r => setTimeout(r, ms)); }

async function createImageTask() {
  const body = {
    task_type: 'image',
    provider: 'replicate',
    service_id: 4,
    raw: true,
    input_data: {
      prompt: 'create an mockup design for a health app',
      aspect_ratio: '9:16',
      output_format: 'png',
      safety_filter_level: 'block_only_high'
    }
  };

  const res = await fetch(`${BASE_URL}/creations`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' }
    // If auth becomes required: add Authorization header here
    , body: JSON.stringify(body)
  });
  if (!res.ok) throw new Error(`Create failed: ${res.status}`);
  return res.json();
}

async function pollUntilDone(id) {
  while (true) {
    const res = await fetch(`${BASE_URL}/creations/${id}?raw=true`);
    if (!res.ok) throw new Error(`Poll failed: ${res.status}`);
    const data = await res.json();

    if (data.status === 'completed') return data;
    if (data.error_message) throw new Error(`Task error: $
{data.error_message}`);

    await sleep(2000); // poll every ~2s
  }
}

async function download(url, outFile) {
  const res = await fetch(url);
  if (!res.ok) throw new Error(`Download failed: ${res.status}`);
  const buf = await res.arrayBuffer();
  await fs.writeFile(outFile, Buffer.from(buf));
  return outFile;
}

(async () => {
  const created = await createImageTask();
  const taskId = created.id;
  const done = await pollUntilDone(taskId);

  const asset = done.output_assets?.[0];
  if (!asset?.url) throw new Error('No output asset found');
  const ext = (done.input_data?.output_format || 'png').toLowerCase();
  const outPath = path.resolve(`image_${taskId}.${ext}`);
  await download(asset.url, outPath);
  console.log('Saved to', outPath);
})();
```

Python — Create, Poll, Download

```python
import time, requests, pathlib

BASE_URL = 'https://video-api.destinpq.com/api/v1'

payload = {
  'task_type': 'image',
  'provider': 'replicate',
  'service_id': 4,
  'raw': True,
  'input_data': {
    'prompt': 'create an mockup design for a health app',
    'aspect_ratio': '9:16',
    'output_format': 'png',
    'safety_filter_level': 'block_only_high'
  }
}

# Create
r = requests.post(f'{BASE_URL}/creations', json=payload)
r.raise_for_status()
created = r.json()

# Poll
task_id = created['id']
while True:
    r = requests.get(f'{BASE_URL}/creations/{task_id}', params={'raw':
'true'})
    r.raise_for_status()
    data = r.json()
    if data['status'] == 'completed':
        break
    if data.get('error_message'):
        raise RuntimeError(f"Task error: {data['error_message']}")
    time.sleep(2)

# Download
asset = data['output_assets'][0]
url = asset['url']
ext = data['input_data'].get('output_format', 'png').lower()
path = pathlib.Path(f'image_{task_id}.{ext}')
img = requests.get(url)
```

```
img.raise_for_status()
path.write_bytes(img.content)
print('Saved to', path)
```

Field Reference (selected)

Top-level

Field	Type	Description
`task_type`	string	**Must be** `"image"`.
`provider`	string	**Must be** `"replicate"`.
`service_id`	number	**Must be** `4`.
`raw`	boolean	**Must be** `true`.
`status`	string	Lifecycle of the task: e.g., `pending` → `completed` or error.
`error_message`	string|null	Present if the task failed.
`output_assets`	array|null	Populated when `status` is `completed`.

`input_data`

Field	Type	Allowed Values
`prompt`	string	Any non-empty text
`aspect_ratio`	string	`1:1`, `9:16`, `16:9`, `3:4`, `4:3`
`output_format`	string	`png`, `jpg`
`safety_filter_level`	string	`block_only_high`, `blow_low_and_above`, `block_medium_and_above`

`output_assets[]`

Field	Type	Notes
`url`	string (URL)	Download this to get the image.
`asset_type`	string	Typically `image`.
`mime_type`	string	MIME hint, may vary by provider.
`metadata`	object	Includes `replicate_prediction` details.

Implementation Notes & Gotchas

- **Always send the four immutable fields** exactly as specified; missing or changing them will fail the request.
- **Polling cadence:** ~2 seconds is recommended to balance latency and load.
- **Completion vs error:** Stop polling on `status: "completed"` or when `error_message` is present.
- **File extension:** Save with `.png`/`.jpg` to match `output_format`.
- **Traceability:** Keep `id` for auditing and correlating to upstream provider IDs in `metadata`.

Minimal Test Checklist

- [ ] POST returns `200 OK` with `status: "pending"` and an `id`.
- [ ] GET with `?raw=true` transitions to `status: "completed"`.
- [ ] `output_assets[0].url` is reachable and downloads the expected format.
- [ ] No deviations from the four immutable fields.

That's it. Use the examples above as your drop-in starter for integrating the image generation flow.