Diamond Sport

Sport API Integration

REST API

Live events, odds, **match odd** and **fancy** results, and cricket scores. Replace `{sport}` with a sport slug (e.g. `cricket`) and `{gmid}` with the game ID from the events feed. Full reference: **diamondtech.shop/portal/docs**

SPORT GATEWAY BASE URL

```
https://diamondtech.shop/api/v1
```

Paths below are **relative to this base** — e.g. `/health` → `https://diamondtech.shop/api/v1/health`. Do not prefix paths with `/v1` again.

Method	Path	Feature	Auth	Description
GET	`/health`	—	—	Gateway health check
GET	`/client/api-key?client=name`	—	—	Retrieve API key (whitelisted IP)
GET	`/events/latest`	events	API key	Latest events snapshot (all sports)
GET	`/events/:sport`	events	API key	Events for one sport
GET	`/odds/:sport/:gmid`	odds	API key	Live match odds / markets
GET	`/result/match-odd/:sport/:gmid`	result	API key	**Match odd result** — one event
POST	`/result/match-odd/:sport/batch`	result	API key	**Match odd results** — up to 25 events together
GET	`/result/fancy/:sport/:eventId/:marketKey`	result	API key	**Fancy result** — one settled market row
GET	`/score/cricket/:id`	score	API key	Live cricket scoreboard

AUTH `GET /client/api-key?client=YOUR_CLIENT`

Call from your whitelisted server IP to retrieve your API key (max 10 requests/hour per IP). Then send on every authenticated request:

```
Header: X-Api-Key: sk_live_***
Or: Authorization: Bearer sk_live_***
Optional: Accept-Encoding: gzip
```

Success response:

```
{
  "ok": true,
  "client": "your-client-name",
  "client_id": "...",
  "api_key": "sk_live_***"
}
```

Step	When	Method & Path
0	Health check	`GET /health`
1	Event list	`GET /events/cricket` or `GET /events/latest`
2	Live markets	`GET /odds/cricket/825211606`
3	Match odd result	`GET /result/match-odd/soccer/758179736`
4	Match odd results (up to 25 events)	`POST /result/match-odd/cricket/batch`
5	Fancy result	`GET /result/fancy/cricket/{eventId}/{marketId}_{sid}`
6	Cricket scoreboard	`GET /score/cricket/825211606`

1. **events** — Match list
2. **odds** — Live markets
3. **result** — Match odd result
4. **score** — Cricket live

Requests require a non-empty `allowed_ips` whitelist per client. Poll **odds** every **100–200 ms** (always fresh, no response cache). Poll **events** and **results** every **500–700 ms**. Reuse HTTP keep-alive and send `Accept-Encoding: gzip` on odds.

Health

No auth

HEALTH `GET /health`


```
{
  "ok": true,
  "service": "prod_publisher",
  "ts": "2026-07-03T12:00:00.000Z"
}
```


Events

Feature: events

LATEST `GET /events/latest`

Cross-sport latest events snapshot. Same inner shape as per-sport events (provider payload under `data`).


```
{
  "ok": true,
  "data": {
    "data": {
      "t1": [ /* in-play / live rows */ ],
      "t2": [ /* upcoming rows */ ]
    }
  }
}
```


CRICKET `GET /events/cricket`

`data` is the raw provider highlight payload. Match rows live in `data.data.t1` (live) and `data.data.t2` (upcoming). Field names vary — always read `gmid` from these arrays.


```
{
  "ok": true,
  "sport": "cricket",
  "data": {
    "data": {
      "t1": [
        {
          "gmid": 825211606,
          "ename": "India v Australia",
          "stime": "2026-07-03T14:30:00.000Z",
          "etid": 4,
          "iplay": true,
          "tv": true
        }
      ],
      "t2": []
    }
  }
}
```

SOCCER GET /events/soccer


```
{
  "ok": true,
  "sport": "soccer",
  "data": {
    "data": {
      "t1": [
        {
          "gmid": 758179736,
          "ename": "Arsenal v Chelsea",
          "stime": "2026-07-03T19:30:00.000Z",
          "etid": 1,
          "iplay": true,
          "tv": true
        }
      ],
      "t2": []
    }
  }
}
```


TENNIS GET /events/tennis


```
{
  "ok": true,
  "sport": "tennis",
  "data": {
    "data": {
      "t1": [
        {
          "gmid": 602394521,
          "ename": "Djokovic v Alcaraz",
          "stime": "2026-07-03T16:00:00.000Z",
          "etid": 2,
          "iplay": false,
          "tv": false
        }
      ],
      "t2": []
    }
  }
}
```


Use `gmid` from the events response for odds and result calls. Other sport slugs:

Sport	Slug	etid	Sport	Slug	etid
Cricket	`cricket`	4	Soccer	`soccer`	1
Tennis	`tennis`	2	MMA	`mixed_martial_art`	3
Table Tennis	`table_tennis`	8	Boxing	`boxing`	6
Horse Racing	`horce_racing`	10	eGames	`egames`	11
Basketball	`basketball`	15	Volleyball	`vollyball`	18
Ice Hockey	`ice_hockey`	19	Badminton	`badmintion`	22
Darts	`dart`	57	American Football	`american_football`	58
Sooner	`sooner`	59	Greyhound	`greyhound_racing`	65
Kabaddi	`kabbdi`	66	eSoccer	`esoccer`	68
Wrestling	`wrestling`	69	Motor Sport	`motor_sport`	52

Use `gmid` from the events response for odds and result calls.

Odds

Feature: odds

Response envelope: `ok`, `sport`, `gmid`, and `data`. The `data` field is the **full Diamond gamedata JSON** (often tens of KB uncompressed — use gzip). It contains match-odd, fancy, and other markets; parse the provider structure directly.

EXAMPLE `GET /odds/cricket/825211606`

Poll every **100–200 ms** while the match is open. Response header `X-Cache: BYPASS` (odds are never HTTP-cached).


```
{
  "ok": true,
  "sport": "cricket",
  "gmid": "825211606",
  "data": {
    "data": [
      {
        "mid": 4852936121997,
        "mname": "MATCH_ODDS",
        "gtype": "match",
        "section": [
          { "sid": 53290, "nat": "India", "odds": [ { "b": 1.85, "l": 1.86 } ] },
          { "sid": 53291, "nat": "Australia", "odds": [ { "b": 2.1, "l": 2.12 } ] }
        ]
      },
      { "mid": 179579571683, "mname": "Normal", "gtype": "fancy", "...": "..." }
    ]
  }
}
```


Same path pattern for all sports: `GET /odds/soccer/{gmid}`, `GET /odds/tennis/{gmid}`, etc.

POLLING `Fast odds requests — keep-alive + gzip`

For live odds, reuse one HTTP connection and enable compression. Always send both headers on every poll:

```
X-Api-Key: sk_live_***
Accept-Encoding: gzip
Connection: keep-alive
```

Why: `Accept-Encoding: gzip` shrinks the JSON payload on the wire (often ~40 KB → ~3 KB). `Connection: keep-alive` reuses the same TCP/TLS session so you skip handshake overhead on repeated polls.

In production, poll the same odds URL every **100–200 ms**. Use a single HTTP client / connection pool — do not open a new TCP connection per request.

Benchmark latency (10 requests) from your whitelisted server:

```
export KEY="sk_live_***"

for i in $(seq 1 10); do
  curl -sS -o /dev/null --compressed --http2 \
    -w "req $i: HTTP %{http_code} wire=%{size_download}B TTFB=%{time_starttransfer}s total=%{time_total}s\n" \
    -H "x-api-key: $KEY" \
    -H "Accept-Encoding: gzip" \
    -H "Connection: keep-alive" \
    "https://diamondtech.shop/api/v1/odds/cricket/920949375"
  sleep 0.2
done
```

Replace `920949375` with your match `gmid`. Check `TTFB` (time to first byte) and `total` (full round-trip). Response headers: `X-Response-Time` (server ms, typically 1–12) · `X-Cache: BYPASS` on odds · `Server-Timing` (auth breakdown). Expect **350–700 ms** total round-trip from India (network + TLS); server processing is only a few ms.

Match odd results

Feature: result

Match odd endpoints return who won the main match market. Fancy returns one settled fancy row for a specific `eventId` + `marketKey`. All require the `result` feature.

CRICKET `GET /result/match-odd/cricket/825211606`

Match odd result for one event. Redis key: `result:match_odd:4:{gmid}`


```
{
  "ok": true,
  "sport": "cricket",
  "sport_id": "4",
  "gmid": "825211606",
  "data": {
    "type": "match_odds_result",
    "result": "India won",
    "wonDimondSid": 53290,
    "eventId": 825211606,
    "marketId": 4852936121997,
    "sportId": 4,
    "updatedAt": { "date": "2026-07-03", "time": "21:15:22" }
  }
}
```


SOCCER `GET /result/match-odd/soccer/758179736`


```
{
  "ok": true,
  "sport": "soccer",
  "sport_id": "1",
  "gmid": "758179736",
  "data": {
    "type": "match_odds_result",
    "result": "Arsenal won",
    "wonDimondSid": 44102,
    "eventId": 758179736,
    "marketId": 5129384712001,
    "sportId": 1,
    "updatedAt": { "date": "2026-07-03", "time": "22:40:10" }
  }
}
```


TENNIS `GET /result/match-odd/tennis/602394521`


```
{
  "ok": true,
  "sport": "tennis",
  "sport_id": "2",
  "gmid": "602394521",
  "data": {
    "type": "match_odds_result",
    "result": "Djokovic won",
    "wonDimondSid": 77831,
    "eventId": 602394521,
    "marketId": 4982710335002,
    "sportId": 2,
    "updatedAt": { "date": "2026-07-03", "time": "19:05:44" }
  }
}
```


Field	Description
`wonDimondSid`	Winning selection ID
`marketId`	Market ID — store with `gmid` when placing bets

BATCH `POST /result/match-odd/cricket/batch`

Match odd results for **up to 25 events together** in one request (one Redis MGET round-trip). The sport in the URL applies to **all** items. Send `gmid` + `marketId` per event — store both when the bet is placed. Batch responses are not HTTP-cached.

```
Request body:
{
  "items": [
    { "gmid": "825211606", "marketId": 4852936121997 },
    { "gmid": "825211607", "marketId": 4852936121998 },
    { "gmid": "825211608", "marketId": 9999999999999 }
  ]
}

Response:
{
  "ok": true,
  "sport": "cricket",
  "sport_id": "4",
  "results": [
    {
      "gmid": "825211606",
      "marketId": 4852936121997,
      "found": true,
      "data": {
        "type": "match_odds_result",
        "result": "India won",
        "wonDimondSid": 53290,
        "eventId": 825211606,
        "marketId": 4852936121997,
        "sportId": 4,
        "updatedAt": { "date": "2026-07-03", "time": "21:15:22" }
      }
    },
    {
      "gmid": "825211607",
      "marketId": 4852936121998,
      "found": false
    },
    {
      "gmid": "825211608",
      "marketId": 9999999999999,
      "found": false,
      "error": "market_mismatch"
    }
  ]
}
```

`found: false` with no `error` — no result in Redis yet. `market_mismatch` — result exists but stored `marketId` does not match the one you sent.

Fancy results

Feature: result

FANCY `GET /result/fancy/cricket/462877809/179579571683_494`

Settled fancy result for one event and market. Redis key: `result:fancy:{etid}:{eventId}`.

Sport	`marketKey` format	Example path segment
Cricket	`{marketId}_{sid}`	`179579571683_494`
Soccer, tennis, etc.	Numeric `marketId` only	`5129384712001`

Soccer / tennis example:

```
GET /result/fancy/soccer/758179736/5129384712001

{
  "ok": true,
  "sport": "soccer",
  "sport_id": "1",
  "eventId": "758179736",
  "marketKey": "5129384712001",
  "data": [
    {
      "type": "match_odds",
      "result": "Arsenal",
      "marketid": 5129384712001,
      "eventId": 758179736,
      "sportId": 1,
      "updatedAt": { "date": "2026-07-03", "time": "22:40:10" }
    }
  ]
}
```

Cricket example (`marketId_sid`):

```
{
  "ok": true,
  "sport": "cricket",
  "sport_id": "4",
  "eventId": "462877809",
  "marketKey": "179579571683_494",
  "data": {
    "type": "normal",
    "result": "32",
    "marketid": 179579571683,
    "eventId": 462877809,
    "sid": 494,
    "nation": "11 over run BAN 2",
    "sportId": 4,
    "updatedAt": { "date": "2026-06-30", "time": "07:31:46" }
  }
}
```

Cricket score

Feature: score

SCORE `GET /score/cricket/825211606`

Live cricket scoreboard for match `{id}` (same as event `gmid`). Requires `score` feature and cricket on the client subscription.

```
{
  "ok": true,
  "id": "825211606",
  "kind": "regular",
  "data": { "...": "live score payload" }
}
```

kind	Source
`regular`	Standard cricket score feed
`crickettv`	Cricket TV score feed
`virtual`	Virtual cricket score feed

Errors & headers

Reference

Status	Error	Cause
401	`missing_api_key`	No `X-Api-Key` or Bearer token
401	`invalid_api_key`	Key not found
403	`ip_not_allowed`	Request IP not in `allowed_ips`
403	`ip_whitelist_required`	Empty `allowed_ips` on client
403	`feature_not_allowed`	Client lacks required feature
403	`sport_not_subscribed`	Client not subscribed to that sport
403	`subscription_inactive`	Client expired or suspended
404	`not_found`	No data for gmid / id / event
404	`market_not_found`	Fancy — no row for that `marketKey`
400	`invalid_sport`	Unknown sport slug
400	`invalid_eventId` / `invalid_marketKey`	Bad fancy path params
400	`invalid_items`	Malformed batch body
400	`batch_limit_exceeded`	Batch has more than 25 items
503	`encryption_not_configured`	API key reveal — server misconfiguration
409	`ambiguous_client`	API key reveal — multiple clients on same IP
429	`rate_limit_exceeded`	Too many API or key-reveal requests

Response headers: `X-Response-Time` (server ms) · `X-Cache` (**odds:** BYPASS · **events / results / score:** HIT / MISS) · `Server-Timing` (auth breakdown). Server processing is typically ~1–12 ms; full client round-trip from India is usually ~350–700 ms (network + TLS), lower with keep-alive and gzip on repeated polls.