API 通知系统技术设计文档

API 通知系统技术设计

本文档旨在为公司内部“API 通知系统”提供一份清晰、完整且可落地的技术设计方案。读者为各业务系统开发者与平台工程团队。设计核心遵循“**简单可靠**”原则，避免过度工程化，优先保障系统的稳定与交付能力。

---

1. 设计目标与范围

1.1. 核心目标

构建一个高可靠、异步化的 API 通知服务。该服务对内提供标准化的 HTTP API，接收业务方（如交易、营销、CRM 等）的通知请求；对外则负责将这些请求**可靠地**、**最终地**派发至指定的外部合作方 API。

核心价值在于 **“发出去”**：业务方只需调用本系统一次，即可将通知的派发与重试责任完全托管，自身不再关心后续的返回结果与送达状态。本系统承诺“尽力而为”地完成通知，但不保证外部系统的最终处理结果。

1.2. 功能范围

- **异步任务受理**：提供 HTTP API，用于接收并持久化通知任务。
- **可靠派发与重试**：实现带指数退避与抖动的后台重试机制，应对网络抖动与外部服务临时不可用。
- **可配置的成功判定**：提供灵活的成功判定策略，可按合作方维度进行配置。
- **幂等性保障**：防止因客户端重试或系统内部状态问题导致的重复派发。
- **安全合规**：内置域名白名单、敏感信息加密、出站代理等安全措施。
- **可观测性**：提供完善的日志、指标与告警，便于排障与容量管理。

1.3. 默认送达判定与可配置策略

为兼顾通用性与灵活性，系统内置默认的成功与重试判定逻辑，同时支持按合作方（`partner_id`）进行策略覆盖。

- **默认成功判定**：
  - HTTP 状态码为 `2xx` 或 `3xx` 的响应，**默认视为送达成功**。
- **默认重试条件**：
  - **网络层错误**：连接超时、DNS 解析失败等。

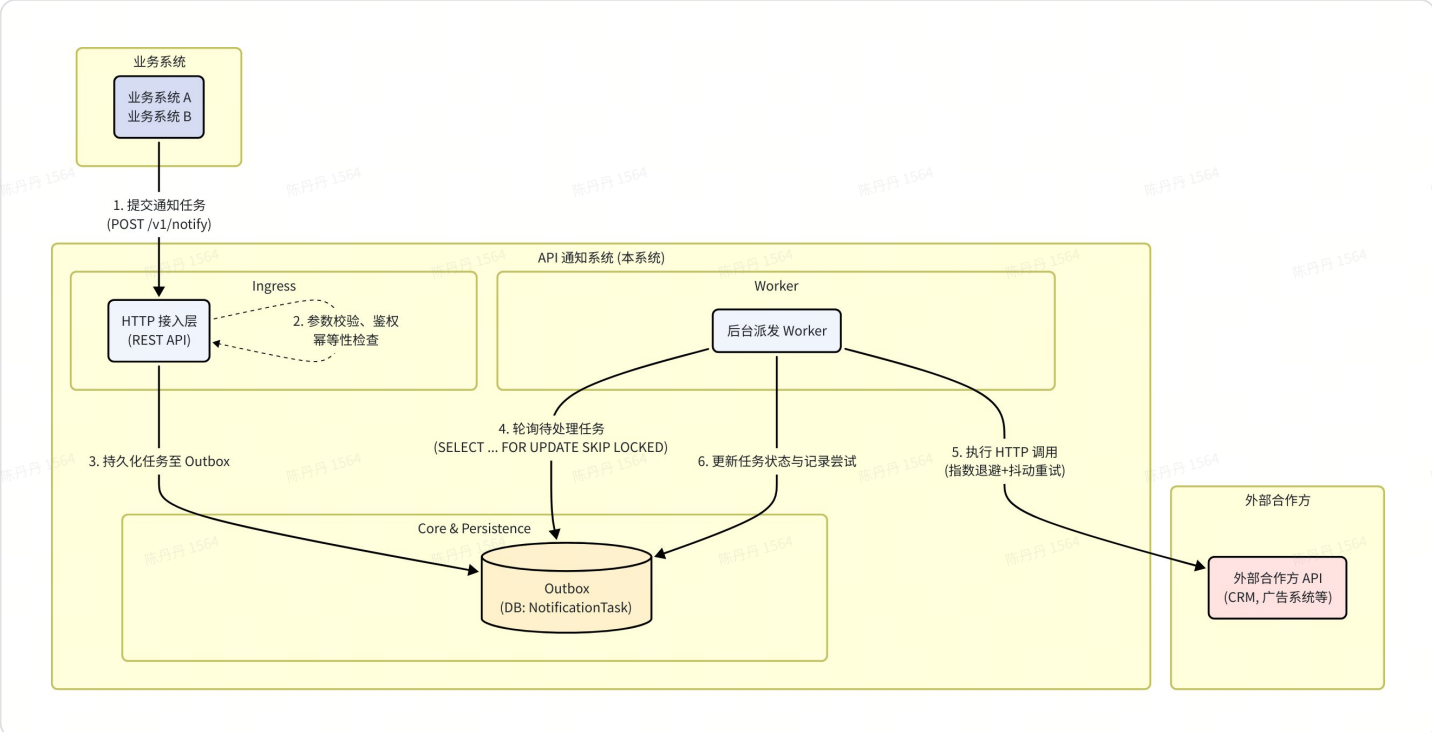
- **HTTP `5xx` 状态码**：表示服务端临时故障，适合重试。
- **HTTP `429 Too Many Requests`**：明确的速率限制信号，应遵循其指示（如 `Retry-After` 头）进行退避重试。

- **默认不重试条件**：
  - **HTTP `4xx` 状态码（除 `429` 外）**：如 `400 Bad Request`、`401 Unauthorized`、`403 Forbidden`。这些通常表示请求本身存在**确定性错误**（如参数错误、权限不足），重试大概率会得到相同结果，因此**默认直接标记为失败**，不再重试。
- **可配置覆盖**：
  - 业务方可在接入时，为特定的 `partner_id` 定义覆盖默认策略。例如，某个合作方的 API 在处理成功时会返回 `404`，则可配置将 `404` 视为此合作方的成功状态。

---

2. 总体架构

本系统采用经典的 **Outbox Pattern**（发件箱模式）实现异步解耦与可靠交付，其核心优势在于**简单可靠**，不引入外部消息队列（如 Kafka、NSQ）等重量级组件，仅依赖**关系型数据库**（如 MySQL/Postgres）即可完成核心功能。



2.1. 核心数据流

1. **提交任务**：业务系统通过内部服务间调用，向 API 通知系统的 **HTTP 接入层**（`/v1/notify`）发起请求，提交通知任务。

2. **持久化**：接入层完成参数校验、鉴权与幂等性检查后，将通知任务的核心信息（目标 URL、请求体、Header 等）封装成一条记录，**原子地写入数据库的 `NotificationTask` 表**，初始状态为 `pending`。成功写入后，立即向业务方返回 `task_id`，表示任务已受理。
3. **轮询与锁定**：**后台派发 Worker**（一个或多个独立的进程/协程）通过定时轮询的方式，从 `NotificationTask` 表中捞取 `pending` 状态且到达执行时间的任务。为避免多实例并发执行同一任务，采用 `SELECT ... FOR UPDATE SKIP LOCKED` 机制，原子地锁定待处理的任务并将其状态更新为 `running`。
4. **执行派发**：Worker 解析任务内容，通过 HTTP 客户端向外部合作方 API 发起调用。
5. **结果处理**：
   - **成功**：根据预设的成功判定策略（如 HTTP 2xx），将任务状态更新为 `succeeded`。
   - **失败（可重试）**：若判定为可重试错误（如 5xx、网络超时），则根据指数退避算法计算下一次执行时间，更新任务的重试次数，并将状态**改回 `pending`**。
   - **失败（不可重试）**：若判定为不可重试错误（如 4xx），或重试次数已达上限，则将任务状态更新为 `failed` 或 `dead`，不再尝试。
6. **记录尝试**：每次派发（无论是成功还是失败）的详细信息，如 HTTP 状态码、响应头摘要、延迟等，都将记录在 `NotificationAttempt` 表中，用于后续的审计与排障。

2.2. 设计取舍

- **为什么不用消息队列（MQ）？**
  - **简单性优先**：引入 MQ 会增加系统复杂度和运维成本（部署、监控、高可用）。对于中低并发的通知场景，数据库的 Outbox 模式完全够用，且更易于维护和理解。
  - **事务一致性**：如果业务操作与发送通知需要在同一个事务中完成，Outbox 模式能更好地保证一致性。虽然本系统是独立服务，但其设计思想源于此，保证了任务的“至少一次”受理。
  - **有序性与依赖管理**：对于需要严格顺序或复杂依赖管理的场景，MQ 更具优势。但本系统的通知任务通常是独立的，不要求严格的先后顺序。

---

3. 关键组件设计

3.1. 接入层 (REST API)

接入层是系统的入口，负责接收、校验和持久化任务。

- **接口定义**：提供 `POST /v1/notify` 接口用于任务提交，`GET /v1/notify/{id}` 用于状态查询。
- **参数校验**：对输入参数进行严格校验，包括 `target_url` 的格式、`method` 的合法性（如仅允许 `POST`, `PUT`）、`headers` 与 `body` 的大小限制等。

- **域名白名单 (SSRF 防护)**：为防止服务端请求伪造 (SSRF) 攻击，系统维护一份**目标域名白名单**。所有 `target_url` 中的域名必须在该白名单内，否则直接拒绝请求。白名单支持前缀匹配，如 `*.example.com`。
- **服务间鉴权**：调用方需携带有效的服务凭证（如内部约定的 `Auth-Token`），接入层会进行校验，确保只有授权的业务系统才能使用本服务。
- **幂等性支持**：调用方可通过请求头或请求体传入 `Idempotency-Key`。详见“幂等与去重”一节。

3.2. 持久化模型

数据模型是系统的核心，定义了任务的生命周期与状态流转。

3.2.1. `notification_tasks` 表

存储通知任务的主体信息。

字段名	类型	描述	索引建议
`id`	`BIGINT`	主键，任务唯一标识	主键
`task_id`	`VARCHAR(64)`	对外暴露的任务 ID，可为 UUID	唯一索引
`partner_id`	`VARCHAR(64)`	合作方/租户标识，用于策略与限速	联合索引 (`partner_id`, `status`, `next_attempt_at`)
`idempotency_key`	`VARCHAR(128)`	幂等键，与 `partner_id` 组合使用	唯一索引 (`partner_id`, `idempotency_key`)
`status`	`VARCHAR(16)`	任务状态 (`pending`, `running`, `succeeded`, `failed`, `dead`)	联合索引 (`status`, `next_attempt_at`)
`target_url`	`VARCHAR(2048)`	目标 URL	
`http_method`	`VARCHAR(16)`	HTTP 方法	
`http_headers`	`TEXT` / `JSONB`	HTTP 请求头（加密存储敏感值）	
`http_body`	`TEXT` / `BLOB`	HTTP 请求体	

`attempt_count`	`INT`	已尝试次数	
`max_attempts`	`INT`	最大允许尝试次数，默认值如 10	
`next_attempt_at`	`DATETIME`	下次尝试的调度时间	联合索引 (`status` , `next_attempt_at`)
`last_attempt_at`	`DATETIME`	上次尝试的执行时间	
`created_at`	`DATETIME`	创建时间	
`updated_at`	`DATETIME`	更新时间	

3.2.2. `notification_attempts` 表

记录每一次派发尝试的详细结果。

字段名	类型	描述	索引建议
`id`	`BIGINT`	主键	主键
`task_id`	`BIGINT`	关联 `notification_tasks` 的主键	索引
`attempt_number`	`INT`	尝试序号，从 1 开始	
`status_code`	`INT`	HTTP 响应状态码	
`response_headers`	`TEXT` / `JSONB`	响应头摘要	
`response_body_summary`	`TEXT`	响应体摘要（默认只记录前 1KB 或哈希值）	
`latency_ms`	`INT`	派发延迟（毫秒）	
`error_message`	`TEXT`	错误信息（如网络错误）	
`created_at`	`DATETIME`	创建时间	

3.2.3. 状态机

任务状态流转是核心逻辑，确保任务在不同阶段的正确处理。

- **`pending`**：初始状态或等待下一次重试。Worker 的主要扫描目标。
- **`running`**：已被 Worker 锁定并正在处理。防止其他 Worker 重复执行。

- `succeeded`：成功送达，终态。
- `failed`：因不可重试错误（如 4xx）而失败，终态。
- `dead`：达到最大重试次数后仍未成功，终态。需要人工介入或归档。

---

4. 派发与重试 (Worker)

后台 Worker 是保证通知送达的核心执行者。

4.1. 核心逻辑

Worker 进程（或一组进程/协程）以固定频率（如每秒）执行以下循环：

1. **查询并锁定任务**：

```
SELECT * FROM notification_tasks
WHERE status = 'pending' AND next_attempt_at <= NOW()
ORDER BY priority DESC, next_attempt_at ASC
LIMIT 100
FOR UPDATE SKIP LOCKED;

```

   - `SKIP LOCKED`：关键特性，允许不同 Worker 实例并发执行此查询而不会相互阻塞，各自获取不同的任务行，实现了简单的分布式任务分发。
   - `LIMIT`：批量获取任务以提高吞吐量。
2. **更新状态**：将获取到的任务状态立即更新为 `running`，防止其他轮询（即使非常罕见）拿到相同的任务。
3. **执行派发**：对每个任务，执行 HTTP 调用。
4. **更新结果**：根据调用结果更新任务状态（`succeeded`、`pending` 或 `dead`）。

4.2. 指数退避与抖动 (Exponential Backoff with Jitter)

为避免在外部服务故障时发起“风暴式”重试，必须采用退避策略。

- **指数退避**：每次重试的间隔时间呈指数级增长。公式：`interval = base_interval * (2 ^ (attempt_count - 1))`。例如，基础间隔为 1s，则重试间隔依次为 1s, 2s, 4s, 8s...
- **随机抖动 (Jitter)**：在计算出的间隔上增加一个随机值，避免所有重试任务在完全相同的时间点“同时唤醒”，从而削平流量尖峰。
  - **Full Jitter**: `next_interval = random_between(0, interval)`

- **Equal Jitter**: `next_interval = interval / 2 + random_between(0, interval / 2)` (更推荐)

4.3. 超时与并发控制

- **HTTP 客户端超时**：必须为出站的 HTTP 请求设置合理的**连接超时**和**请求超时**（如 5s），防止因外部服务响应缓慢而耗尽 Worker 资源。
- **并发控制**：通过控制 Worker 的并发数（如协程池大小）来限制同一时刻向外部发出的总请求量。

4.4. 优先级与速率限制（可选）

- **优先级**：可在 `notification_tasks` 表中增加 `priority` 字段，高优先级的任务被优先轮询和执行。
- **速率限制**：
  - **全局限速**：使用 `Token Bucket` (令牌桶) 等算法，限制整个系统的总出站 QPS。
  - **按合作方限速**：为每个 `partner_id` 维护一个独立的令牌桶，实现更精细的流量控制，避免因某一合作方 API 的限制而影响其他通知。

---

5. 对内标准 API 设计

5.1. `POST /v1/notify` - 提交通知任务

用于创建并提交一个新的通知任务。

Request Body:

代码块

```
{
  "partner_id": "some_crm_vendor",
  "idempotency_key": "unique-order-id-12345",
  "target_url": "https://api.partner.com/v1/webhooks/orders",
  "method": "POST",
  "headers": {
    "Content-Type": "application/json",
    "X-Partner-Token": "secret-token-for-crm"
  },
  "body": "{\"order_id\": \"SO12345\", \"amount\": 99.99, \"status\": \"paid\"}",
  "priority": 10,
  "success_condition": {
    "status_codes": [200, 201, 204]
  }
}
```

```
16
```

- `partner_id` (string, required): 合作方标识，用于路由、策略和限速。
- `idempotency_key` (string, required): 幂等键，在 `partner_id` 范围内唯一。
- `target_url` (string, required): 目标 URL。
- `method` (string, required): HTTP 方法，如 `POST`。
- `headers` (object): HTTP 请求头。
- `body` (string): HTTP 请求体，应为字符串（如 JSON stringified）。
- `priority` (int, optional): 任务优先级，数值越大优先级越高。
- `success_condition` (object, optional): 自定义成功判定条件，覆盖默认策略。

Response (Success):

代码块

```
{
  "task_id": "ts_2a9y3x8z7v6q",
  "status": "accepted"
}

```

5.2. `GET /v1/notify/{id}` - 查询任务状态

查询指定任务的当前状态与最近的尝试记录。

Response:

代码块

```
{
  "task_id": "ts_2a9y3x8z7v6q",
  "status": "succeeded",
  "created_at": "2023-10-27T10:00:00Z",
  "last_attempt_at": "2023-10-27T10:00:05Z",
  "attempt_count": 1,
  "attempts": [
    {
      "attempt_number": 1,
      "status_code": 200,
      "latency_ms": 150,
      "created_at": "2023-10-27T10:00:05Z"
    }
  ]
```

```
}

```

5.3. `POST /v1/notify/{id}/cancel` - 取消任务 (可选)

允许业务方取消尚未执行或正在等待重试的任务。

- **实现**：将 `status` 为 `pending` 的任务更新为 `cancelled` (一个终态)。对 `running` 状态的任务，此操作可能失败或需要更复杂的处理，初期可不支持。

---

6. Go 参考实现示例

以下为核心逻辑的 Go 代码片段，突出设计思想，非完整实现。

6.1. 数据结构与 SQL 定义

代码块

```go
// NotificationTask 对应 notification_tasks 表
type NotificationTask struct {
    ID             int64     `db:"id"`
    TaskID         string    `db:"task_id"`
    PartnerID      string    `db:"partner_id"`
    IdempotencyKey string    `db:"idempotency_key"`
    Status         string    `db:"status"`
    TargetURL      string    `db:"target_url"`
    HTTPMethod     string    `db:"http_method"`
    HTTPHeaders    string    `db:"http_headers"` // 序列化为 JSON 字符串
    HTTPBody       []byte    `db:"http_body"`
    AttemptCount   int       `db:"attempt_count"`
    MaxAttempts    int       `db:"max_attempts"`
    NextAttemptAt  time.Time `db:"next_attempt_at"`
    LastAttemptAt  sql.NullTime `db:"last_attempt_at"`
}

// PostgreSQL 表定义
const createTaskTableSQL = `
CREATE TABLE notification_tasks (
    id BIGSERIAL PRIMARY KEY,
    task_id VARCHAR(64) NOT NULL UNIQUE,
    partner_id VARCHAR(64) NOT NULL,
    idempotency_key VARCHAR(128) NOT NULL,
    status VARCHAR(16) NOT NULL DEFAULT 'pending',
    target_url VARCHAR(2048) NOT NULL,
    http_method VARCHAR(16) NOT NULL,
```

```
    http_headers JSONB,
    http_body BYTEA,
    attempt_count INT NOT NULL DEFAULT 0,
    max_attempts INT NOT NULL DEFAULT 10,
    next_attempt_at TIMESTAMPTZ NOT NULL,
    last_attempt_at TIMESTAMPTZ,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    UNIQUE (partner_id, idempotency_key)
);

CREATE INDEX idx_tasks_status_next_attempt ON notification_tasks (status, next_attempt_at);
`
```

6.2. Worker 轮询与派发核心代码

代码块

```go
func (w *Worker) run() {
    ticker := time.NewTicker(w.pollInterval)
    defer ticker.Stop()

    for range ticker.C {
        tasks, err := w.fetchAndLockTasks()
        if err != nil {
            // log error
            continue
        }

        for _, task := range tasks {
            go w.processTask(task)
        }
    }
}

func (w *Worker) fetchAndLockTasks() ([]NotificationTask, error) {
    var tasks []NotificationTask
    query := `
        SELECT * FROM notification_tasks
        WHERE status = 'pending' AND next_attempt_at <= NOW()
        ORDER BY next_attempt_at ASC
        LIMIT $1
        FOR UPDATE SKIP LOCKED`

```

```
    tx, err := w.db.Begin()
    if err != nil {
        return nil, err
    }
    defer tx.Rollback()

    rows, err := tx.Query(query, w.batchSize)
    // ... 扫描 rows 到 tasks ...

    // 将任务状态更新为 running
    // ...

    return tasks, tx.Commit()
}

func (w *Worker) processTask(task NotificationTask) {
    // 1. 构建 HTTP 请求
    req, _ := http.NewRequest(task.HTTPMethod, task.TargetURL,
bytes.NewReader(task.HTTPBody))
    // ... 设置 headers ...

    // 2. 发送请求
    client := &http.Client{Timeout: 10 * time.Second}
    resp, err := client.Do(req)

    // 3. 处理结果
    if err != nil || (resp.StatusCode >= 500 && resp.StatusCode != 429) {
        // 计算下一次重试时间
        nextAttempt := w.calculateNextAttempt(task)
        // 更新数据库，状态改回 pending
    } else if resp.StatusCode >= 200 && resp.StatusCode < 400 {
        // 更新数据库，状态为 succeeded
    } else {
        // 更新数据库，状态为 failed/dead
    }

    // 4. 记录 attempt
}
```

6.3. 幂等键处理

在 `POST /v1/notify` 接口中，核心逻辑如下：

1. 从请求中获取 `partner_id` 和 `idempotency_key`。

2. **查询数据库**：`SELECT task_id, status FROM notification_tasks WHERE partner_id = ? AND idempotency_key = ?`。
3. **如果记录已存在**：
    - 直接返回已存在的 `task_id` 和 `200 OK` 或 `202 Accepted`，表示请求已被处理。
    - **不要**创建新任务。
4. **如果记录不存在**：
    - 这是新请求，继续执行创建任务的流程。
    - 为避免并发冲突，应在数据库层面为 `(partner_id, idempotency_key)` 添加唯一约束。如果插入时违反约束，说明在查询和插入之间有另一个相同请求进入，此时应重新查询并返回已有记录。

---

7. 安全、观测性与运维

7.1. 安全与合规 (Security)

- **目标域名白名单**：已在“接入层”提及，是防止 SSRF 的首要防线。
- **敏感值加密**：对于 `headers` 或 `body` 中可能包含的 `Authorization`、`token` 等敏感信息，应在存储前使用 KMS 或配置中心提供的加密服务进行**字段级加密**，并在派发时由 Worker 解密注入。绝不应明文存储。
- **出站代理**：所有出站的网络请求应通过公司的标准出站代理进行，便于统一的网络策略管理和流量审计。
- **TLS 校验**：默认**必须校验**外部 API 的 TLS 证书。仅在特殊情况下（如内部测试），才允许配置跳过。

7.2. 观测性 (Observability)

- **日志 (Logging)**：
    - **接入层**：记录每个请求的 `trace_id`，`partner_id`，`idempotency_key`。
    - **Worker**：记录每个任务执行的 `task_id`，`attempt_count`，`latency_ms`，`status_code`，`error`。日志应为结构化格式（如 JSON），便于机器解析。
- **核心指标 (Metrics)**：
    - **入站**：`notify_requests_total` (by `partner_id`, `status`) - 任务接收总量。
    - **派发**：`notify_attempts_total` (by `partner_id`, `status_code`) - 派发尝试次数。
    - **成功率**：`succeeded_rate` = `succeeded_tasks / total_tasks`。
    - **延迟**：`notify_attempt_latency_seconds` (Histogram) - 派发延迟分布。

    - **死信率**：`dead_letter_rate` = `dead_tasks / total_tasks`。
- **告警 (Alerting)**：
    - **死信队列增长**：`dead` 状态的任务数量持续增加，表明存在持久性问题。
    - **派发成功率骤降**：某个 `partner_id` 的成功率在短时间内大幅下降。
    - **任务积压**：`pending` 状态的任务数量持续超过阈值，且 `next_attempt_at` 远早于当前时间。
    - **值班处置建议**：检查对应合作方的 API 状态、网络连通性；根据日志排查请求/响应内容。

7.3. 运维与数据治理

- **数据留存周期 (TTL)**：
    - `notification_tasks` 表中的终态任务（`succeeded`，`failed`，`dead`）应设置 TTL（如 30 天）。
    - `notification_attempts` 表的数据应设置更短的 TTL（如 7 天）。
- **归档与清理**：定期运行归档任务，将过期的记录从主库移动到成本更低的历史库或直接删除。
- **容量估算与扩容**：
    - **Worker**：无状态，可**水平扩容**，增加实例数即可提高处理能力。
    - **数据库**：
        - **垂直扩容**：升级 DB 实例规格。
        - **分片**：当单表数据量过大时，可按 `partner_id` 或时间进行分片。

---

8. 部署与 SLO

8.1. 最小可用拓扑

- **1 个** API 接入层实例 (无状态，可多实例)
- **1 个** 后台 Worker 实例 (可多实例)
- **1 个** 关系型数据库实例 (如 RDS)

此架构已具备基本的高可用性。接入层和 Worker 均可水平扩展。数据库是单点，但云服务商的 RDS 通常提供主备切换能力。

8.2. 服务等级目标 (SLO)

- **任务受理成功率**：`99.99%` (API 层面)
- **99% 的通知在 1 小时内送达**：此 SLO 衡量系统的最终送达能力，允许一定时间的重试。

- **95% 的通知在 5 分钟内首次尝试**：衡量 Worker 的处理延迟。

---

9. 开放问题与可选增强

- **自定义成功判定逻辑**：除了状态码，是否需要支持基于响应体内容（如 `{"code": 0}`）或响应头的成功判定？这会增加复杂性。
- **回调/Webhook 支持**：当任务终态（成功/失败）时，是否需要反向通知业务系统？
- **高级认证支持**：是否需要为特定合作方内置签名逻辑（如 HMAC-SHA256）、OAuth2 客户端凭证模式等？初期可由业务方在 `headers` 中自行构造。
- **跨地域部署**：为实现灾备，是否需要支持跨地域多活？这将对数据库同步提出更高要求。

这些问题可在系统成熟后，根据业务发展需求再行评估。