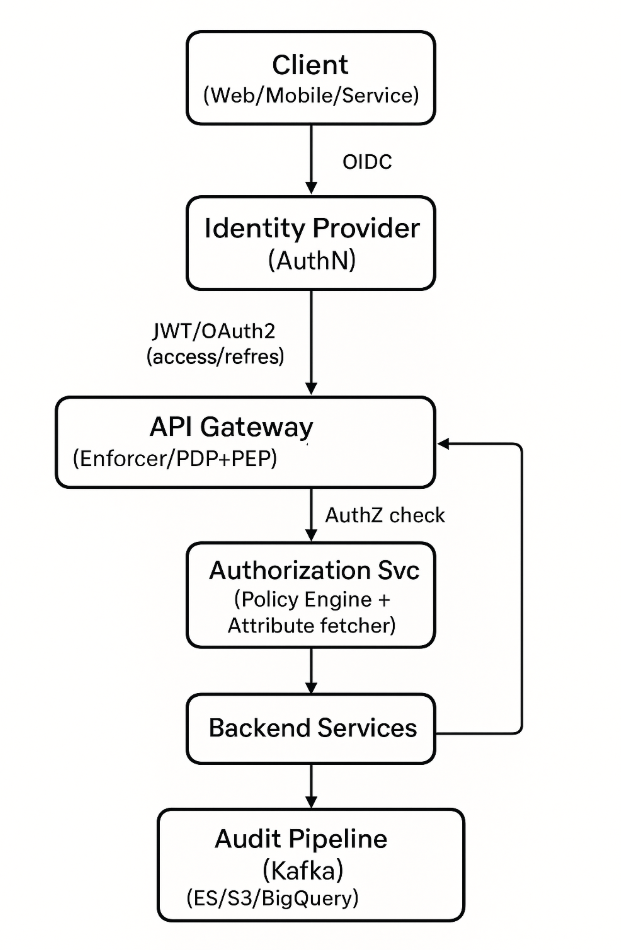
**一、开场 & 澄清（1–2 分钟）**

**五个最关键的澄清问题 / Five Key Clarifying Questions**

1. **用户与租户  
   ZH: 用户是谁（企业员工、外部合作方、服务账号）？系统是否需要支持多租户？  
   EN: Who are the users (employees, external partners, service accounts)? Do we need multi-tenant support?**
2. **资源范围  
   ZH: 我们要保护哪些资源（API、数据记录、对象存储、页面路由）？  
   EN: What resources do we need to protect (APIs, data records, object storage, page routes)?**
3. **权限模型  
   ZH: 权限控制是只用 RBAC，还是也需要 ABAC 来做更细粒度的策略？  
   EN: Should we stick to RBAC, or also include ABAC for fine-grained policies?**
4. **认证方式  
   ZH: 登录方式需要支持哪些？比如用户名密码、OIDC/SAML SSO、MFA？  
   EN: What authentication methods are required—username/password, OIDC/SAML SSO, MFA?**
5. **Admin 管理后台  
   ZH: 是否需要一个管理控制台或管理 API，用来创建用户、分配角色、配置策略？  
   EN: Do we need an admin console or admin APIs to create users, assign roles, and manage policies?**

**二、总体架构（2–3 分钟）**



**1. 用户登录 / User Sign-in**

**ZH:**  
首先，用户会通过客户端（Web、Mobile 或 Service）去访问 Identity Provider，比如 Azure AD 或 Okta。IdP 会负责验证用户身份，比如检查用户名密码、触发 MFA，甚至支持单点登录。如果验证成功，IdP 会颁发一个 Access Token（通常是 JWT）以及可选的 Refresh Token。这些 token 就是后续请求里证明身份的凭证。

**EN:**  
First, the user signs in through a client such as web, mobile, or service, which connects to the Identity Provider like Azure AD or Okta. The IdP handles authentication, checking username and password, triggering MFA, or supporting SSO. If authentication succeeds, the IdP issues an Access Token (usually a JWT) and optionally a Refresh Token. These tokens will serve as the credentials in subsequent requests.

**2. 请求 API / Calling the API**

**ZH:**  
当用户拿到 Token 之后，就会带着它去请求 API。请求首先会经过 API Gateway，这里会做 Token 验证，包括校验签名、检查过期时间和受众等。这样可以确保请求者的身份有效，而且没有被篡改。

**EN:**  
With the token in hand, the client sends requests to the API. The API Gateway sits in front of all services, and its first job is to validate the token—checking the signature, verifying expiration, and confirming the audience. This ensures that the requester’s identity is valid and that the token has not been tampered with.

**3. 授权检查 / Authorization Check**

**ZH:**  
验证 Token 通过后，Gateway 会将请求交给 Authorization Service，也就是 PDP。PDP 会从 Policy Store 中读取相关策略，结合用户的角色、属性以及访问的资源，来判断是否允许这个操作。最终它会返回一个决策，比如 Allow 或 Deny。

**EN:**  
Once the token is validated, the Gateway forwards the request to the Authorization Service, also known as the PDP （Policy Decision Point）. The PDP retrieves the relevant policies from the Policy Store and evaluates them against the user’s roles, attributes, and the resource being accessed. Based on this evaluation, it makes a decision and returns either Allow or Deny.

**4. 访问服务 / Accessing Backend Services**

**ZH:**  
如果 PDP 返回的是 Allow，API Gateway 就会把请求转发到后端服务去处理。这样，后端服务可以专注于业务逻辑，而不需要自己实现复杂的权限控制。如果返回的是 Deny，Gateway 会直接拦截请求，返回相应的错误码。

**EN:**  
If the PDP returns Allow, the API Gateway forwards the request to the backend service for processing. This way, backend services can focus on business logic without having to implement complex access control themselves. If the decision is Deny, the Gateway blocks the request immediately and responds with the appropriate error code.

**5. 审计记录 / Audit Logging**

**ZH:**  
不管结果是允许还是拒绝，每一次访问和决策都会被记录到审计流水线，比如通过 Kafka 推送到日志系统，最终存储到 Elasticsearch、S3 或 BigQuery 里。这些记录可以用于合规要求、异常检测和安全分析，确保系统在安全和合规上有可追溯性。

**EN:**  
Regardless of whether the request is allowed or denied, every access and decision is logged into the audit pipeline, often through Kafka, and stored in systems like Elasticsearch, S3, or BigQuery. These records serve compliance needs, anomaly detection, and security analysis, ensuring that the system remains traceable and accountable.

👉 一句话总结 / One-line recap  
**ZH:** 用户登录 IdP → 拿到 Token 请求 API → Gateway 验证后问 PDP → PDP 查策略返回 Allow/Deny → Gateway 放行或拒绝 → 全程写审计。  
**EN:** User signs in via IdP → gets a token and calls API → Gateway validates and consults PDP → PDP checks policies and returns Allow/Deny → Gateway enforces → everything is logged in audit.

**Access Control 流程（简洁版）**

1. **用户登录**
   * **Client → Identity Provider (IdP) Azure AD 安定Okta**
   * **IdP 验证身份，发 JWT / OAuth2 Token**
2. **请求 API**
   * **Client 带 Token → API Gateway**
   * **Gateway 校验 Token（签名/有效期）**
3. **授权检查**
   * **Gateway → Authorization Service (PDP)**
   * **PDP 从 Policy Store 取策略，做判断 → 返回 Allow/Deny**
4. **访问服务**
   * **Allow → 请求转发到 Backend Service**
   * **Deny → Gateway 拒绝**
5. **审计记录**
   * **所有请求与决策写入 Audit Pipeline（Kafka → ES/S3/BigQuery）**

**IdP 负责认证发 Token，Gateway 拦截请求，PDP 查策略做授权，最后记录到审计日志。**

**角色说明 / Role Descriptions**

**IdP (AuthN)**

* **ZH: 负责用户注册与登录，支持多因素认证 (MFA)，颁发 Token（例如 JWT），提供 JWKS 公钥用于校验签名，并处理 Refresh Token 流程。**
* **EN: Handles user registration and login, supports multi-factor authentication (MFA), issues tokens (e.g., JWT), publishes JWKS for signature verification, and manages refresh token flow.**

**API Gateway (PEP)**

* **ZH: 作为策略执行点，负责验证 JWT，并根据授权结果进行强制执行（Allow/Deny）。在需要时，它会调用授权服务进行进一步判断。**
* **EN: Acts as the Policy Enforcement Point (PEP), responsible for verifying JWTs and enforcing decisions (Allow/Deny). When necessary, it calls the Authorization Service for further evaluation.**

**Authorization Service (PDP)**

* **ZH: 作为策略决策点，负责对访问请求进行评估，可以基于 RBAC、ABAC 或 OPA 策略。同时，它可以从目录服务或属性服务中获取用户/资源上下文信息来辅助决策。**
* **EN: Serves as the Policy Decision Point (PDP), evaluating access requests based on RBAC, ABAC, or OPA policies. It may also fetch user or resource context from directory/attribute services to support decision-making.**

**Policy Store**

* **ZH: 存放角色、权限和策略，并确保租户之间的隔离。支持策略的版本化与灰度发布，便于策略逐步上线和回滚。**
* **EN: Stores roles, permissions, and policies with tenant isolation. Supports policy versioning and gradual rollout, allowing safe updates and rollback when needed.**

**Audit**

* **ZH: 记录所有授权决策和敏感操作，将数据异步写入存储系统。支持合规检索和安全分析，满足审计要求。**
* **EN: Captures all authorization decisions and sensitive operations, asynchronously writing them into storage systems. Supports compliance queries and security analysis to meet auditing requirements.**

**三、认证（AuthN）设计要点（2–3 分钟）**

**ZH:**

* **协议：主要使用 OIDC/OAuth2（Auth Code with PKCE）；在企业环境可以额外支持 SAML。**
* **Token 策略：**
  + **Access Token：JWT 格式，短期有效（5–15 分钟），通过 JWKS 公钥做无状态验证。**
  + **Refresh Token：长期有效（7–30 天），可撤销，存放在 Redis/DB，并支持旋转（每次使用后更换）。**
* **多因素认证 (MFA)：优先使用 TOTP 或 WebAuthn，短信作为备选，并提供注册与恢复流程。**
* **会话撤销机制：  
  1）缩短 Access Token 的有效期 + 定期轮换 JWKS 公钥；  
  2）在高风险场景下触发 Token Introspection 或短 TTL 的 deny-list；  
  3）在 Refresh Token 刷新时进行风险评估（比如 IP 变更、地理位置、设备指纹）。**

**EN:**

* **Protocols: Primarily OIDC/OAuth2 (Authorization Code with PKCE); add SAML for enterprise integration.**
* **Token Strategy:**
  + **Access Token: JWT, short-lived (5–15 minutes), validated statelessly via JWKS.**
  + **Refresh Token: Long-lived (7–30 days), revocable, stored in Redis/DB, and rotated (replaced upon each use).**
* **MFA: Prefer TOTP or WebAuthn; SMS as a fallback; provide enrollment and recovery flows.**
* **Session Revocation:**
  + **Shorten access token TTL + rotate JWKS keys regularly.**
  + **For high-risk cases, trigger token introspection or use a short-TTL deny-list.**
  + **During refresh, enforce risk checks (e.g., IP change, geolocation, device fingerprint).**

**四、授权（AuthZ）模型（4–6 分钟）**

**4.1 RBAC**

**ZH:**

* **实体关系：User ↔ Role（多对多），Role ↔ Permission（多对多），Permission 定义为 Action + Resource Pattern。**
* **适用场景：大部分后台管理系统或企业角色权限（如 Admin、Approver、Viewer）。**

**EN:**

* **Entities: User ↔ Role (many-to-many), Role ↔ Permission (many-to-many), and Permission maps to Action + Resource Pattern.**
* **Use Case: Most back-office or enterprise roles, such as Admin, Approver, Viewer.**

**4.2 ABAC（基于属性/条件）**

**ZH:**

* **规则：基于用户属性 (user.attr)、资源属性 (resource.attr)、环境条件（时间/IP/租户）来做判断。**
* **示例：**
  + **allow if user.department == "Finance" AND resource.tag contains "Quarterly"**
  + **deny if env.ip not in corpCIDR**
* **适用场景：需要细粒度控制、跨团队资源共享、数据分片或行级权限的场景。**

**EN:**

* **Rules: Evaluations are based on user attributes, resource attributes, and environment conditions (e.g., time/IP/tenant).**
* **Examples:**
  + **allow if user.department == "Finance" AND resource.tag contains "Quarterly"**
  + **deny if env.ip not in corpCIDR**
* **Use Case: Fine-grained access, cross-team shared resources, data sharding, or row-level permissions.**

**4.3 混合与优先级**

**ZH:**

* **策略合并顺序：Deny > Allow；显式 Deny 优先。**

**EN:**

* **Policy Combination Order: Deny overrides Allow; explicit Deny takes precedence.**

**五、数据模型（2–3 分钟） / Data Model (2–3 min)**

**ZH:  
核心表包括：**

* **Users：存储用户信息（id, email, status, attrs, tenant\_id）。**
* **Roles：定义角色（id, name, tenant\_id）。**
* **Permissions：权限表，定义 action + resource\_pattern。**
* **User\_Roles：用户与角色多对多关系。**
* **Role\_Permissions：角色与权限多对多关系。**
* **Policies：策略表（RBAC/ABAC），支持条件和版本化。**
* **Audit\_Logs：审计日志，记录访问请求、决策和原因。**

**EN:  
The core tables include:**

* **Users: Stores user info (id, email, status, attrs, tenant\_id).**
* **Roles: Defines roles (id, name, tenant\_id).**
* **Permissions: Defines action + resource\_pattern.**
* **User\_Roles: Many-to-many relation between users and roles.**
* **Role\_Permissions: Many-to-many relation between roles and permissions.**
* **Policies: Policy table (RBAC/ABAC), with conditions and versioning.**
* **Audit\_Logs: Audit logs recording access requests, decisions, and reasons.**

**六、API 接口 / API Design**

**ZH:**

* **认证相关（IdP 提供）：**
  + **POST /oauth/token：换取 Access Token / Refresh Token。**
  + **GET /.well-known/openid-configuration：发现端点。**
* **授权决策（PDP 提供）：**
  + **POST /v1/authorize:evaluate：输入 subject/action/resource/env，返回 Allow/Deny。**
* **管理接口（Admin API）：**
  + **POST /v1/users：创建用户**
  + **POST /v1/roles：创建角色**
  + **POST /v1/permissions：定义权限**
  + **POST /v1/policies：新建策略**
  + **GET /v1/audit：查询审计日志**

**EN:**

* **Authentication (IdP):**
  + **POST /oauth/token: Exchange authorization code for access/refresh tokens.**
  + **GET /.well-known/openid-configuration: OIDC discovery.**
* **Authorization (PDP):**
  + **POST /v1/authorize:evaluate: Input subject/action/resource/env, returns Allow/Deny.**
* **Admin APIs:**
  + **POST /v1/users: Create user**
  + **POST /v1/roles: Create role**
  + **POST /v1/permissions: Define permission**
  + **POST /v1/policies: Create policy**
  + **GET /v1/audit: Query audit logs**

**面试口语总结 / Interview Oral Summary**

**ZH:  
“在数据模型上，我会有 Users、Roles、Permissions 三张基础表，通过 User\_Roles 和 Role\_Permissions 形成多对多关系，另外加上 Policies 表来支持 RBAC/ABAC 策略，以及 Audit\_Logs 来做合规审计。API 方面，认证走标准 OIDC 接口，授权有一个统一的 /authorize:evaluate 决策接口，管理端提供用户、角色、权限和策略的 CRUD，还有审计查询。整体设计简洁清晰，易于扩展。”**

**EN:  
“For the data model, I’d keep core tables for Users, Roles, and Permissions, linked by User\_Roles and Role\_Permissions. On top of that, I’d add a Policies table to support RBAC and ABAC, and an Audit\_Logs table for compliance. On the API side, authentication follows standard OIDC endpoints, authorization is handled by a unified /authorize:evaluate endpoint, and admin APIs allow CRUD on users, roles, permissions, and policies, plus audit queries. The design is straightforward and extensible.”**

**七、性能与扩展（3–4 分钟） / Performance & Scalability (3–4 min)**

**ZH:  
我们优先采用无状态架构，使用 JWT + JWKS。授权结果会做本地缓存（30–120 秒），按用户、操作和资源来索引。针对热点数据，比如角色和权限映射，可以放在 Redis，并通过版本号和 Kafka 消息来做缓存失效。ABAC 属性也可以用只读缓存，写入由后台同步。授权请求是高读低写的场景，所以策略发布采用蓝绿或灰度，保证引擎热更新不影响服务。高可用方面，多 AZ 部署，Gateway/IdP/AuthZ 都可以水平扩展。JWKS 支持双 Key 窗口来做平滑轮换。延迟目标是：Gateway 自身 5ms，本地缓存命中 1ms，未命中时 PDP 评估 5–10ms，整体 P99 小于 20ms。**

**EN:  
We prioritize stateless design with JWT + JWKS. Authorization decisions are cached locally (30–120s) by subject, action, and resource. Hot data such as role-permission mappings go into Redis, with versioning and Kafka events for cache busting. ABAC attributes can use read-only caching with async sync for writes. Since authorization is high-read/low-write, policy rollout uses blue-green or canary deployments to ensure hot updates without downtime. For availability, we deploy across multiple AZs, with horizontal scaling for Gateway, IdP, and AuthZ. JWKS supports dual-key rotation for smooth key rollover. Latency targets: ~5ms at Gateway, ~1ms on cache hit, 5–10ms on cache miss, with P99 under 20ms overall.**

**八、安全与合规（2–3 分钟） / Security & Compliance (2–3 min)**

**ZH:  
安全方面坚持最小权限原则，显式 Deny 优先，策略变更必须有审批和审计。密钥由 KMS 管理，定期轮换，全链路强制 HTTPS 和 HSTS。敏感字段，比如用户属性和 PII，采用列级或字段级加密，日志里要脱敏。威胁模型包括：**

* **Token 窃取：短期 access token + IP/设备绑定 + refresh rotation + 异常检测。**
* **重放攻击：使用 jti、nonce 和时钟偏移窗口。**
* **策略绕过：所有检查统一由 Gateway 执行，后端拒绝直连。**

**合规方面，审计日志支持按用户、操作、资源和时间进行检索，满足合规调查。日志要设置保留期和 Legal Hold，数据驻留可按租户选择区域。**

**EN:  
For security, we enforce least privilege, explicit deny precedence, and approval plus auditing for policy changes. Keys are managed by KMS, rotated regularly, and all traffic is protected with HTTPS and HSTS. Sensitive fields such as user attributes and PII are encrypted at column/field level, with masked logs. Threat models include:**

* **Token theft: mitigate with short-lived access tokens, IP/device binding, refresh rotation, and anomaly detection.**
* **Replay attacks: use jti, nonce, and clock skew tolerance.**
* **Policy bypass: enforce checks uniformly at the Gateway, backend rejects direct access.**

**For compliance, audit logs support queries by user, action, resource, and time, fulfilling regulatory requirements. Logs have retention policies and legal hold. Data residency is tenant-specific, allowing regional isolation.**

**IAM / Access Control 系统设计 (面试口语版)**

**1. Clarify 需求 / 澄清问题**

**EN:**  
“First, let me clarify the scope. Is this system for internal employees only, or also external partners? Do we need multi-tenant support? And do we want just RBAC, or should we also support ABAC for more fine-grained policies?”

**ZH:**  
“我先确认下范围。这个系统是只给公司员工用，还是也给外部合作方？要不要支持多租户？权限模型是只要 RBAC，还是也要支持 ABAC 做更细粒度的控制？”

**2. High-level Design / 高层设计**

**EN:**  
“At a high level, I’ll split it into Authentication and Authorization. Authentication is handled by the Identity Provider, issuing tokens like JWT. Authorization is handled by a Policy Engine. API Gateway sits in front, checking tokens and asking the AuthZ service whether to allow or deny. All decisions are logged into Audit.”

**ZH:**  
“整体来说，我会分成认证和授权。认证由 Identity Provider 负责，发 token（比如 JWT）。授权由策略引擎来做。前面有 API Gateway，先验证 token，再问授权服务要不要放行。所有决策都会写进审计日志。”

**3. AuthN (认证)**

**EN:**  
“IdP (Identity Provider) supports login, MFA, and SSO. It issues short-lived JWT Access Tokens and long-lived Refresh Tokens. Access tokens are stateless, verified via JWKS. Refresh tokens are stored in DB or Redis and can be revoked.”

**ZH:**  
“IdP 负责登录、MFA 和单点登录。它会发短期的 JWT Access Token，还有长期的 Refresh Token。Access Token 是无状态的，用 JWKS 验证。Refresh Token 会存数据库/Redis，可以随时撤销。”

**4. AuthZ (授权)**

**EN:**  
“For authorization, we can support RBAC first, then ABAC. RBAC is simple: user → role → permission. ABAC adds flexibility: allow if user.department == Finance && resource.tag == Quarterly. We’ll store all policies in a Policy Store, and the Authorization Service (PDP) evaluates them and returns allow or deny.”

**ZH:**  
“授权这块，可以先做 RBAC，再扩展 ABAC。RBAC 很简单：用户 → 角色 → 权限。ABAC 更灵活，比如 ‘允许财务部的人查看季度报表’。所有策略存到 Policy Store，授权服务 (PDP) 会去查，然后返回允许还是拒绝。”

**5. Data Model / 数据模型**

**EN:**  
“Key tables: users, roles, permissions, user\_roles, role\_permissions, policies, and audit logs. Each policy has effect (allow/deny), conditions, and version.”

**ZH:**  
“主要表：users、roles、permissions、user\_roles、role\_permissions、policies，还有 audit 日志。每个策略都有 allow/deny、条件和版本号。”

**6. Flow / 请求流程**

**EN:**  
“User logs in → gets JWT → calls API → Gateway validates token → calls AuthZ service → PDP checks policy → allow or deny → log the decision.”

**ZH:**  
“用户登录 → 拿到 JWT → 调 API → Gateway 验证 token → 调授权服务 → PDP 查策略 → 决定放行还是拒绝 → 记日志。”

**7. Scaling & Trade-offs / 扩展与权衡**

**EN:**  
“To scale: tokens are stateless, PDP can be horizontally scaled, cache decisions in Redis or local LRU. Trade-offs: JWT is fast but hard to revoke; introspection is safer but slower. RBAC is easy to manage, ABAC is flexible but complex.”

**ZH:**  
“扩展方面：token 无状态，PDP 可以横向扩展，授权决策可以用 Redis 或本地缓存。权衡点：JWT 快，但不好撤销；introspection 安全，但慢。RBAC 好管，ABAC 灵活但复杂。”

**8. Security & Audit / 安全与审计**

**EN:**  
“All requests and decisions are logged for compliance. Tokens are rotated, keys managed by KMS. Sensitive data is encrypted and logs are masked.”

**ZH:**  
“所有请求和决策都会写审计日志，满足合规。Token 会轮换，密钥用 KMS 管理。敏感数据加密，日志脱敏。”

**9. Ending / 总结**

**EN:**  
“So to summarize: IdP handles Authentication with OIDC and JWT, Gateway enforces security, PDP + Policy Store handle Authorization with RBAC and ABAC, and all actions are logged for compliance.”

**ZH:**  
“总结一下：IdP 负责认证（OIDC + JWT），Gateway 做安全入口，PDP + Policy Store 负责授权（RBAC + ABAC），所有操作都进审计日志，保证合规。”

**Clarification Questions:**

**1. 用户 & 租户**

**EN:** “Who are the users of this system? Just internal employees, or also external partners and service accounts? Do we need multi-tenant support?”  
**ZH:** “用户是谁？只有内部员工，还是也包含外部合作方、服务账号？需要支持多租户吗？”

**2. 资源范围**

**EN:** “What kind of resources do we need to protect? APIs, web pages, databases, files, or all of them?”  
**ZH:** “要保护的资源有哪些？是 API、网页、数据库、文件，还是全部都要管？”

**3. 权限模型**

**EN:** “Do you expect a simple RBAC model, or do we also need ABAC or more fine-grained policies?”  
**ZH:** “权限模型是只要 RBAC 就行，还是也要支持 ABAC 这种更细的策略？”

**4. 登录与认证**

**EN:** “How do users log in? Just username/password, or do we need SSO and MFA? Any specific IdP to integrate with, like Okta or Azure AD?”  
**ZH:** “用户怎么登录？只要用户名密码，还是要 SSO 和 MFA？需不需要和现有的 IdP（比如 Okta、Azure AD）集成？”

**5. 安全与合规**

**EN:** “Are there compliance requirements like GDPR, SOX, or HIPAA? How long do we need to keep audit logs?”  
**ZH:** “有没有合规要求，比如 GDPR、SOX、HIPAA？审计日志要保存多久？”

**6. 性能与规模**

**EN:** “What’s the expected scale? Number of users, peak QPS, and latency SLA for authorization decisions?”  
**ZH:** “预期规模是多少？用户量、峰值 QPS，以及授权决策的延迟 SLA 有什么要求？”

**7. 系统边界**

**EN:** “Should the system only handle AuthZ decisions, or also provide full IdP features like user registration and password reset?”  
**ZH:** “系统只负责授权决策，还是也要做完整的 IdP 功能，比如注册、密码重置？”

**简单版 IAM / Access Control 系统设计总结（口语化）**

**EN:**  
“My design separates Authentication and Authorization.

* The **Identity Provider** handles Authentication — login, MFA, SSO — and issues short-lived JWT access tokens and refresh tokens.
* The **API Gateway** sits in front of all services. It verifies tokens, then calls the **Authorization Service**.
* The **Authorization Service (PDP)** checks rules from the **Policy Store**. We can support RBAC first, then ABAC for more flexible conditions.
* If allowed, the request goes to the backend; if denied, the gateway blocks it.
* Every decision is written into the **Audit Pipeline** for compliance and security review.

In short: IdP authenticates who you are, Authorization Service decides what you can do, Gateway enforces the decision, and Audit logs everything.”

**ZH:**  
“我的设计把认证和授权分开。

* **IdP** 负责认证：登录、MFA、SSO，发短期的 JWT access token 和 refresh token。
* **API Gateway** 在前面拦截所有请求，先验 token，再调用 **授权服务**。
* **授权服务 (PDP)** 去 **策略库 (Policy Store)** 查规则，先支持 RBAC，再扩展 ABAC。
* 如果允许，请求就进后台服务；如果拒绝，Gateway 直接挡掉。
* 所有决策都会写进 **审计日志**，保证安全合规。

一句话总结：IdP 负责‘你是谁’，AuthZ 服务负责‘你能做什么’，Gateway 负责执行，Audit 负责记录。”

**DB Schema（PostgreSQL 为例）**

-- Tenants / Users

CREATE **TABLE tenants** (

id BIGSERIAL PRIMARY KEY,

name TEXT NOT NULL,

plan TEXT,

data\_region TEXT,

created\_at TIMESTAMPTZ DEFAULT now()

);

CREATE **TABLE users** (

id BIGSERIAL PRIMARY KEY,

tenant\_id BIGINT NOT NULL REFERENCES tenants(id),

email CITEXT UNIQUE,

status TEXT CHECK (status IN ('active','disabled','locked')) DEFAULT 'active',

attrs JSONB DEFAULT '{}'::jsonb,

created\_at TIMESTAMPTZ DEFAULT now()

);

CREATE INDEX idx\_users\_tenant ON users(tenant\_id);

CREATE INDEX idx\_users\_email ON users(email);

-- RBAC

CREATE **TABLE roles** (

id BIGSERIAL PRIMARY KEY,

tenant\_id BIGINT NOT NULL REFERENCES tenants(id),

name TEXT NOT NULL,

desc TEXT,

UNIQUE (tenant\_id, name)

);

CREATE **TABLE permissions (**

id BIGSERIAL PRIMARY KEY,

tenant\_id BIGINT NOT NULL REFERENCES tenants(id),

action TEXT NOT NULL,

resource\_pattern TEXT NOT NULL, -- e.g. "report:\*" / "report:{id}"

effect TEXT NOT NULL CHECK (effect IN ('ALLOW','DENY')) DEFAULT 'ALLOW'

);

CREATE **TABLE user\_roles** (

user\_id BIGINT NOT NULL REFERENCES users(id) ON DELETE CASCADE,

role\_id BIGINT NOT NULL REFERENCES roles(id) ON DELETE CASCADE,

PRIMARY KEY (user\_id, role\_id)

);

CREATE INDEX idx\_user\_roles\_role ON user\_roles(role\_id);

CREATE **TABLE role\_permissions** (

role\_id BIGINT NOT NULL REFERENCES roles(id) ON DELETE CASCADE,

permission\_id BIGINT NOT NULL REFERENCES permissions(id) ON DELETE CASCADE,

PRIMARY KEY (role\_id, permission\_id)

);

-- Policies（ABAC / PBAC / 显式 Deny）

CREATE **TABLE policies** (

id BIGSERIAL PRIMARY KEY,

tenant\_id BIGINT NOT NULL REFERENCES tenants(id),

type TEXT NOT NULL CHECK (type IN ('RBAC','ABAC','DENY','MIXED')),

name TEXT NOT NULL,

expr JSONB NOT NULL, -- 规则表达式（或 Rego/Cedar 存文本），这里用 JSONB

priority INT DEFAULT 100, -- 越小优先级越高

version TEXT DEFAULT 'v1',

enabled BOOLEAN DEFAULT TRUE,

created\_by BIGINT,

created\_at TIMESTAMPTZ DEFAULT now(),

updated\_at TIMESTAMPTZ DEFAULT now()

);

CREATE INDEX idx\_policies\_tenant ON policies(tenant\_id);

CREATE INDEX idx\_policies\_enabled ON policies(tenant\_id, enabled, priority);

CREATE **TABLE resources** (

id BIGSERIAL PRIMARY KEY,

tenant\_id BIGINT NOT NULL REFERENCES tenants(id),

type TEXT NOT NULL, -- e.g. "report"

external\_id TEXT, -- 业务系统里的ID

attrs JSONB DEFAULT '{}'::jsonb

);

CREATE INDEX idx\_resources\_lookup ON resources(tenant\_id, type, external\_id);

-- 审计日志（热数据放 PG / 冷数据落 ES/S3/BigQuery）

CREATE **TABLE audit\_logs** (

id BIGSERIAL PRIMARY KEY,

tenant\_id BIGINT NOT NULL REFERENCES tenants(id),

ts TIMESTAMPTZ NOT NULL DEFAULT now(),

request\_id TEXT,

subject\_id BIGINT,

action TEXT,

resource TEXT,

decision TEXT CHECK (decision IN ('ALLOW','DENY','ERROR')),

reason TEXT,

policy\_version TEXT,

extras JSONB

);

CREATE INDEX idx\_audit\_tenant\_ts ON audit\_logs(tenant\_id, ts DESC);

CREATE INDEX idx\_audit\_subject ON audit\_logs(tenant\_id, subject\_id, ts DESC);

Scalability：

**1) 无状态与水平扩展 / Stateless & Horizontal Scale**

**ZH:** IdP、Gateway、PDP 都做无状态；前挂负载均衡，横向加副本即可扩容。  
**EN:** Keep IdP, Gateway, and PDP stateless; put them behind a load balancer and scale out replicas.

**要点 / Notes:** 短 TTL 的 Access Token + JWKS 校验；避免粘性会话。  
**Tip:** Short-lived access tokens + JWKS; avoid sticky sessions.

**2) 决策缓存 / Decision Caching**

**ZH:** 在 Gateway/PDP 本地做 LRU 决策缓存（30–120s），Key 用 (sub, action, resourceKey, policy\_version)；Redis 作为共享缓存，策略变更通过 Kafka 广播失效。  
**EN:** LRU decision cache at Gateway/PDP (30–120s), key by (sub, action, resourceKey, policy\_version); Redis for shared cache; invalidate via Kafka on policy updates.

**要点 / Notes:** 热点资源可提前预热；防止“热 key”，可做随机抖动 TTL。  
**Tip:** Warm hot keys; add jitter to TTL to avoid thundering herds.

**3) 策略评估加速 / Faster Policy Evaluation**

**ZH:** 预编译策略（如 OPA Rego→WASM）；RBAC 做“角色闭包”预计算（role→permissions 展开并缓存）；ABAC 属性走只读缓存，写入异步同步。  
**EN:** Pre-compile policies (e.g., OPA Rego→WASM); precompute RBAC role→permission closure and cache; use read-only caches for ABAC attributes with async write sync.

**4) 架构分层与部署形态 / Topology Choices**

**ZH:** 中心式 PDP 适合统一管理；超低延迟可选 **PDP sidecar**（就近评估）；跨区域采用 active-active，多租户可按租户或区域分片。  
**EN:** Central PDP for consistency; use PDP sidecars for ultra-low latency; active-active across regions; shard by tenant or region.

**5) 数据分片与读扩展 / Data Sharding & Read Scale**

**ZH:** Policy Store 做按租户或区域分片；读多写少场景上读副本；审计日志冷热分层（热数据在 OLTP，冷数据落 ES/S3/BigQuery）。  
**EN:** Shard Policy Store by tenant/region; add read replicas; tier audit logs (hot in OLTP, cold in ES/S3/BigQuery).

**6) 弹性与发布 / Elasticity & Rollouts**

**ZH:** HPA/自动扩容（CPU/QPS/队列长度）；蓝绿/金丝雀发布策略与引擎，支持影子评估（shadow eval）与一键回滚。  
**EN:** HPA based on CPU/QPS/queue depth; blue-green/canary for policies and engines; shadow evaluation and instant rollback.

**7) 流控与熔断 / Rate Limit & Resilience**

**ZH:** Gateway 做速率限制、令牌桶；对 PDP 设置超时、重试、熔断与回退策略（fail-closed 用于高敏感路径，fail-open+告警用于低敏感）。  
**EN:** Gateway rate limit (token bucket); PDP timeouts/retries/circuit breakers; fail-closed for sensitive paths, fail-open+alerts for low-risk ones.

**8) 审计管道可扩展 / Scalable Audit Pipeline**

**ZH:** 审计走异步 Kafka，消费者可水平扩展；批量写入与压缩；后端存储分区按时间/租户；查询侧建索引与数据生命周期（TTL/Legal Hold）。  
**EN:** Audit via async Kafka; scale consumers; batch + compression; partition by time/tenant; indexing + lifecycle (TTL/Legal Hold).

**9) 密钥与 Token 策略 / Keys & Token Strategy**

**ZH:** JWKS 双 Key 轮换窗口；控制 Token 体积（只放必要 claims）；下游用受信头部透传身份，减少重复解码。  
**EN:** JWKS dual-key rotation; keep tokens small (essential claims only); propagate identity via trusted headers downstream.

**10) Schema 与索引 / Schema & Indexing**

**ZH:** 高频查询加复合索引（如 policies(tenant\_id, enabled, priority)，permissions(tenant\_id, action)）；审计按 (tenant\_id, ts) 分区；避免跨租户全表扫。  
**EN:** Composite indexes for hot queries; partition audit by (tenant\_id, ts); avoid cross-tenant full scans.

**11) 观察性与容量规划 / Observability & Capacity**

**ZH:** 关键指标：PDP P99、缓存命中率、策略评估时长、拒绝原因分布、Kafka 积压、Redis 命中/热点。按峰值×冗余做容量，留 30–50% 余量。  
**EN:** Track PDP P99, cache hit rate, policy eval time, deny reasons, Kafka lag, Redis hot keys. Plan for peak × redundancy with 30–50% headroom.