SPEC-001-ABAC for Spring Boot Microservices (Vendor-Neutral, Keycloak-friendly)

Background

The platform currently uses Role-Based Access Control (RBAC) in a Spring Boot microservices architecture. New business rules require more granular, context-aware decisions (e.g., resource ownership, data classification, tenant, region, time-of-day). We will add Attribute-Based Access Control (ABAC) while:

- Keeping RBAC for backward compatibility (hybrid RBAC + ABAC).
- Reading user/resource attributes from Keycloak initially (ID Token/Access Token claims and UserInfo), but **not** coupling evaluation to Keycloak, so the same evaluation layer can later source attributes from any IdP or datastore.

Requirements

Must (M): - M1: Introduce ABAC decisions for HTTP and method-level authorization without breaking existing RBAC paths. - M2: Vendor-neutral **Policy Decision Point (PDP)** API with pluggable engines (e.g., in-process SpEL, external OPA, future Cedar) and pluggable **Attribute Resolvers** (Keycloak-JWT, DB, UserInfo, HTTP headers). - M3: Policy model stored in DB and/or external policy engine; policies versioned and auditable. - M4: Compatible with Spring Boot 3.x / Spring Security 6.x Resource Server (JWT) and Gateway. - M5: Evaluate decisions using subject, action, resource, and environment attributes. - M6: Centralized logging/metrics for allow/deny with reasons.

Should (S): - S1: Policy caching with invalidation hooks. - S2: Multi-tenant policy namespaces. - S3: Idempotent migration path from RBAC → hybrid RBAC+ABAC.

Could (C): - C1: Compile policies to WASM (OPA) for edge enforcement. - C2: Admin UI for policy authoring with linting and dry-runs.

Won’t (W): - W1: Build a bespoke IdP; we will integrate with existing OIDC providers.

Method

Production Choice: External PDP via OPA (Open Policy Agent)

Rationale: decouples authorization decisions from service code; supports hot-reloaded policies (bundles), standard REST API, rich decision logs, and horizontal scaling. Services remain vendor-neutral via a common **PDP interface**.

Architecture (PEP/PDP/PIP) – Production

- **PEP (enforcement):**

- HTTP layer using Spring Security `AuthorizationManager<RequestAuthorizationContext>` at gateway and services.
- Method layer via custom `PermissionEvaluator` (optional for service-to-service or domain methods).
- **PDP (decision):** external OPA cluster (central) or sidecar per pod. Accessed over mTLS HTTP with timeouts/retries and **fail-closed** semantics.
- **PIP (attributes):** AttributeResolver chain merges JWT claims (OIDC), request metadata, and resource attributes from services/DB.
- **Policy distribution:** CI builds **OPA bundles** from Rego + data; served by an internal bundle service/S3; OPAs poll on interval with ETag.
- **Observability:** OPA **decision logs** to Kafka/HTTP; PEP emits app logs/metrics; tracing spans annotate authz latency.

PlantUML – Deployment View

```
@startuml
skinparam componentStyle rectangle
node "Kubernetes Cluster" {
  node "Namespace: prod" {
    node "API Gateway Pod" {
      [Gateway] - [OPA Sidecar]
    }
    node "Service A Pod" {
      [Service A] - [OPA Sidecar]
    }
    node "Service B Pod" {
      [Service B] - [OPA Sidecar]
    }
    node "Policy Bundle Server" as PBS
    node "Observability" {
      [Logs/ELK]
      [Metrics/Prom]
      [Tracing/Jaeger]
    }
  }
}
PBS --> [OPA Sidecar] : HTTPS bundle downloads
[Service A] --> [OPA Sidecar] : mTLS /v1/data/app/allow
[Gateway] --> [OPA Sidecar] : mTLS /v1/data/app/allow
[OPA Sidecar] --> [Logs/ELK] : decision logs
@enduml
```

PlantUML – Class/Interfaces (abstractions)

```
@startuml
interface PolicyDecisionPoint {
```

```
  +Decision evaluate(action, resourceType, context)
}
class OpaPolicyDecisionPoint
class AttributeResolver
class AttributeResolverChain
class AbacAuthorizationManager
PolicyDecisionPoint <|.. OpaPolicyDecisionPoint
AttributeResolverChain o-- AttributeResolver
AbacAuthorizationManager ..> PolicyDecisionPoint
AbacAuthorizationManager ..> AttributeResolverChain
@enduml
```

PlantUML – Sequence (request with cache)

```
@startuml
actor User
participant Gateway
participant PEP
participant Attrs as AttributeResolvers
participant Cache
participant PDP as OPA
User -> Gateway: HTTP request (JWT)
Gateway -> PEP: authorize(request)
PEP -> Attrs: resolve attributes
Attrs --> PEP: {subject,resource,env}
PEP -> Cache: get(key=hash(input))
alt hit
  Cache --> PEP: ALLOW/DENY
else miss
  PEP -> PDP: POST /v1/data/app/allow {input}
  PDP --> PEP: {result: {allow:true}}
  PEP -> Cache: put(key, decision, ttl)
end
PEP --> Gateway: proceed/deny
@enduml
```

Data & Policy

- **Bundle layout**

```
/bundles/app/
  policies/
    http.rego
    documents.rego
  data/
```

```
    classifications.json
  manifest.json
```

- **Input schema (to OPA)**

```
{
  "action": "GET|POST|...",
  "resourceType": "document",
  "ctx": {
    "subject": {"sub": "...", "roles": ["..."], "tenant": "t1"},
    "resource": {"id": "123", "ownerId": "...", "classification":
"internal"},
    "env": {"method": "GET", "path": "/documents/123", "ip": "..."}
  }
}
```

- **Rego example (documents.rego)**

```
package app

default allow = false

# Admins can do anything
allow {
  some r
  input.ctx.subject.roles[r] == "admin"
}

# Owner can read
allow {
  input.action == "GET"
  input.resourceType == "document"
  input.ctx.resource.ownerId == input.ctx.subject.sub
}

# Block secrets unless admin
deny_secret {
  input.resourceType == "document"
  input.ctx.resource.classification == "secret"
}

allow {
  not deny_secret
  input.action == "GET"
}
```

Security & Resilience

- mTLS between services and OPA; restrict egress to OPA only.
- Timeouts (e.g., 100–200ms), small retries with jitter, circuit-breaker fallback **deny**.
- Decision caching (e.g., Caffeine, TTL 30–60s) keyed by (subject, action, resource hash).
- Rate limits and bulkheads on PDP calls.

Implementation

1) Dependencies (Gradle)

```
dependencies {
  implementation("org.springframework.boot:spring-boot-starter-web")
  implementation("org.springframework.boot:spring-boot-starter-security")
  implementation("org.springframework.boot:spring-boot-starter-oauth2-resource-server")
  implementation("io.github.resilience4j:resilience4j-spring-boot3:2.2.0")
  implementation("com.github.ben-manes.caffeine:caffeine:3.1.8")
}
```

2) Config Properties

```
// AbacProps.java
@ConfigurationProperties(prefix = "abac.opa")
public record AbacProps(String baseUrl, Duration timeout, int maxRetries, Duration cacheTtl) {}
```

```
# application.yaml
abac:
  opa:
    base-url: https://localhost:8181/v1/data/app/allow
    timeout: 200ms
    max-retries: 1
    cache-ttl: 45s
spring:
  security:
    oauth2:
      resourceserver:
        jwt:
          issuer-uri: https://auth.example.com/realms/prod
```

3) Robust OPA Client + PDP

```
// OpaClientConfig.java
@Configuration
@EnableConfigurationProperties(AbacProps.class)
class OpaClientConfig {
  @Bean WebClient opaWebClient(AbacProps p){
    HttpClient http = HttpClient.create()
      .responseTimeout(java.time.Duration.ofMillis(p.timeout().toMillis()))
      .compress(true);
    return WebClient.builder()
      .baseUrl(p.baseUrl())
      .clientConnector(new ReactorClientHttpConnector(http))
      .defaultHeader(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE)
      .build();
  }
}
```

```
// Caffeine cache for decisions
@Configuration
class DecisionCacheConfig {
  @Bean Cache<String, PolicyDecisionPoint.Decision> decisionCache(AbacProps p){
    return
Caffeine.newBuilder().expireAfterWrite(p.cacheTtl()).maximumSize(10000).build();
  }
}
```

```
// OpaPolicyDecisionPoint.java (prod)
@Component
class OpaPolicyDecisionPoint implements PolicyDecisionPoint {
  private final WebClient opa; private final Retry retry; private final
Cache<String, Decision> cache;
  OpaPolicyDecisionPoint(WebClient opaWebClient, AbacProps props, Cache<String,
Decision> cache){
    this.opa = opaWebClient; this.cache = cache;
    this.retry = Retry.of("opa",
RetryConfig.custom().maxAttempts(props.maxRetries()
+1).waitDuration(Duration.ofMillis(50)).build());
  }
  @Override public Decision evaluate(String action, String resourceType,
Map<String, Object> ctx){
    String key = Integer.toHexString(Objects.hash(action, resourceType, ctx));
    Decision cached = cache.getIfPresent(key); if (cached != null) return
cached;
```

```
    Map<String,Object> input = Map.of("action", action, "resourceType",
resourceType, "ctx", ctx);
    try {
      Map<?,?> result = Try.ofSupplier(Retry.decorateSupplier(retry, () ->
        opa.post().bodyValue(Map.of("input", input)).retrieve()
           .onStatus(HttpStatusCode::is5xxServerError, r -> Mono.error(new
IllegalStateException("OPA 5xx")))
           .bodyToMono(Map.class).block()))
        .get();
      boolean allow = Boolean.TRUE.equals(((Map<?,?
>)result.get("result")).get("allow"));
      Decision d = new Decision(allow ? Effect.ALLOW : Effect.DENY, "opa",
Map.of());
      cache.put(key, d); return d;
    } catch (Exception ex){
      // fail-closed
      return new Decision(Effect.DENY,
"opa_error:"+ex.getClass().getSimpleName(), Map.of());
    }
  }
}
```

4) HTTP PEP

```
@Component
class AbacAuthorizationManager implements
AuthorizationManager<RequestAuthorizationContext> {
  private final AttributeResolverChain attrs; private final PolicyDecisionPoint
pdp;
  AbacAuthorizationManager(AttributeResolverChain a, PolicyDecisionPoint p){
this.attrs=a; this.pdp=p; }
  @Override public AuthorizationDecision check(Supplier<Authentication>
authentication, RequestAuthorizationContext ctx){
    Authentication auth = authentication.get();
    Map<String,Object> context = attrs.buildContext(ctx, auth);
    String action = ctx.getRequest().getMethod();
    String resourceType = ((Map<String,Object>)context.getOrDefault("resource",
Map.of())).getOrDefault("type","unknown").toString();
    var decision = pdp.evaluate(action, resourceType, context);
    return new AuthorizationDecision(decision.effect() ==
PolicyDecisionPoint.Effect.ALLOW);
  }
}
```

5) Attribute Resolvers (JWT + URI + DB)

```
@Component
class JwtClaimsAttributeResolver implements AttributeResolver {
  @Override public Map<String, Object> resolve(RequestAuthorizationContext ctx,
Authentication auth){
    if (!(auth instanceof JwtAuthenticationToken token)) return Map.of();
    Jwt jwt = token.getToken();
    Map<String,Object> subject = new HashMap<>();
    subject.put("sub", jwt.getSubject());
    subject.put("tenant", jwt.getClaimAsString("tenant"));
    subject.put("roles",
Optional.ofNullable(jwt.getClaimAsStringList("realm_access.roles")).orElse(List.of()));
    return Map.of("subject", subject,
                  "env", Map.of("method", ctx.getRequest().getMethod(), "path",
ctx.getRequest().getRequestURI()));
  }
}
```

```
@Component
class ResourceParamAttributeResolver implements AttributeResolver {
  @Override public Map<String, Object> resolve(RequestAuthorizationContext ctx,
Authentication auth){
    var path = ctx.getRequest().getRequestURI();
    String id = path.replaceAll("^.*/(\w+)$", "$1");
    return Map.of("resource", Map.of("type", "document", "id", id));
  }
}
```

```
@Component
class ResourceDbAttributeResolver implements AttributeResolver {
  private final DocumentRepository repo;
  ResourceDbAttributeResolver(DocumentRepository r){ this.repo = r; }
  @Override public Map<String, Object> resolve(RequestAuthorizationContext ctx,
Authentication auth){
    String id = ctx.getRequest().getRequestURI().replaceAll("^.*/(\w+)$", "$1");
    return repo.findById(id)
      .map(doc -> Map.of("resource", Map.of("ownerId", doc.getOwnerId(),
"classification", doc.getClassification())))
      .orElse(Map.of());
  }
}
```

6) Spring Security Wiring

```
@Configuration
@EnableWebSecurity
@EnableMethodSecurity
class SecurityConfig {
  private final AuthorizationManager<RequestAuthorizationContext> abac;
  SecurityConfig(AuthorizationManager<RequestAuthorizationContext> abac){
this.abac = abac; }
  @Bean SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.oauth2ResourceServer(o -> o.jwt());
    http.csrf(csrf -> csrf.disable());
    http.authorizeHttpRequests(reg -> reg
      .requestMatchers("/actuator/health").permitAll()
      .requestMatchers(HttpMethod.GET, "/documents/**").access(abac)
      .anyRequest().authenticated());
    return http.build();
  }
}
```

7) OPA Sidecar Config (example)

```
# opa-config.yaml
services:
  bundle_svc:
    url: https://bundles.internal
    credentials:
      bearer:
        token: ${BUNDLE_TOKEN}
bundles:
  app:
    service: bundle_svc
    resource: bundles/app/bundle.tar.gz
    polling:
      min_delay_seconds: 10
      max_delay_seconds: 30
decision_logs:
  console: true
  reporting:
    min_delay_seconds: 5
    max_delay_seconds: 15
plugins:
  envoy_ext_authz_grpc: # if using gateway envoy
    addr: :9191
```

8) Kubernetes Snippet (sidecar)

```
containers:
- name: service-a
  image: ghcr.io/org/service-a:1.0.0
  env:
  - name: ABAC_OPA_BASE_URL
    value: https://localhost:8181/v1/data/app/allow
  ports:
  - containerPort: 8080
- name: opa
  image: openpolicyagent/opa:0.66.0-rootless
  args: ["run","--server","--config-file=/config/opa-config.yaml"]
  ports:
  - containerPort: 8181
  volumeMounts:
  - name: opa-config
    mountPath: /config
```

9) Observability Hooks

- Log each decision (effect, action, resourceType, rule/trace) at **DEBUG**, and only denials at **WARN**.
- Export metrics: `abac.pdp.latency`, `abac.pdp.errors`, `abac.cache.hit_ratio`.
- Add tracing span `abac.evaluate` with attributes `pdp=opa`, `decision=allow|deny`.

Below is a runnable **MVP** outline. Choose either **A) In-process SpEL PDP** or **B) OPA PDP**; both implement the same `PolicyDecisionPoint`.

1) Dependencies (Gradle)

```
// build.gradle.kts
plugins {
  id("org.springframework.boot") version "3.3.5"
  id("io.spring.dependency-management") version "1.1.6"
  kotlin("jvm") version "1.9.24" // or Java 21
}

dependencies {
  implementation("org.springframework.boot:spring-boot-starter-web")
  implementation("org.springframework.boot:spring-boot-starter-security")
  implementation("org.springframework.boot:spring-boot-starter-oauth2-resource-
server")
  implementation("org.springframework.boot:spring-boot-starter-validation")
  implementation("org.springframework.boot:spring-boot-starter-data-jpa")
  runtimeOnly("org.postgresql:postgresql")
```

```
  // SpEL policy engine
  implementation("org.springframework:spring-expression")

  // Optional: OPA client (HTTP) or Styra OPA Spring SDK
  // implementation("com.styra:opa-spring-boot-sdk:<latest>")
}
```

2) Security Setup (HTTP + Method security)

```
// SecurityConfig.java
@Configuration
@EnableWebSecurity
@EnableMethodSecurity // replaces deprecated @EnableGlobalMethodSecurity
public class SecurityConfig {
  private final AuthorizationManager<RequestAuthorizationContext> abacAuthz;

  public SecurityConfig(AuthorizationManager<RequestAuthorizationContext>
abacAuthz) {
    this.abacAuthz = abacAuthz;
  }

  @Bean
  SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
      .oauth2ResourceServer(oauth2 -> oauth2.jwt(Customizer.withDefaults()))
      .csrf(csrf -> csrf.disable())
      .authorizeHttpRequests(reg -> reg
          .requestMatchers("/actuator/**").permitAll()
          // Hybrid: first enforce ABAC for protected APIs
          .requestMatchers(HttpMethod.GET, "/documents/**").access(abacAuthz)
          // Fallback or parallel RBAC checks
          .anyRequest().authenticated()
      );
    return http.build();
  }
}
```

3) ABAC Core Contracts

```
// PolicyDecisionPoint.java
public interface PolicyDecisionPoint {
  enum Effect { ALLOW, DENY }
  record Decision(Effect effect, String reason, Map<String, Object>
obligations) {}
  Decision evaluate(String action, String resourceType, Map<String, Object>
```

```
context);
}

// AttributeResolver.java
public interface AttributeResolver {
  Map<String, Object> resolve(RequestAuthorizationContext requestCtx,
Authentication auth);
}

// AttributeResolverChain.java
@Component
public class AttributeResolverChain {
  private final List<AttributeResolver> resolvers;
  public AttributeResolverChain(List<AttributeResolver> resolvers){
this.resolvers = resolvers; }
  public Map<String, Object> buildContext(RequestAuthorizationContext ctx,
Authentication auth){
    Map<String,Object> out = new HashMap<>();
    for (var r: resolvers) out.putAll(r.resolve(ctx, auth));
    return out;
  }
}
```

4A) Attribute Resolvers (Keycloak-friendly, vendor-neutral)

```
// JwtClaimsAttributeResolver.java
@Component
public class JwtClaimsAttributeResolver implements AttributeResolver {
  @Override public Map<String, Object> resolve(RequestAuthorizationContext ctx,
Authentication auth){
    Map<String,Object> m = new HashMap<>();
    if (auth instanceof JwtAuthenticationToken token){
      Jwt jwt = token.getToken();
      m.put("subject", Map.of(
          "sub", jwt.getSubject(),
          "email", jwt.getClaimAsString("email"),
          "roles", jwt.getClaimAsStringList("realm_access.roles")
      ));
      m.put("env", Map.of(
          "method", ctx.getRequest().getMethod(),
          "path", ctx.getRequest().getRequestURI()
      ));
    }
    return m;
  }
}
```

```
// ResourceParamAttributeResolver.java (example: pull resource id/type from URI)
@Component
public class ResourceParamAttributeResolver implements AttributeResolver {
  @Override public Map<String, Object> resolve(RequestAuthorizationContext ctx,
Authentication auth){
    var path = ctx.getRequest().getRequestURI(); // e.g., /documents/{id}
    String id = path.replaceAll("^.*/(\\w+)$", "$1");
    return Map.of("resource", Map.of("type", "document", "id", id));
  }
}
```

4B) In-Process PDP (SpEL) – MVP

```
// SpelPolicyDecisionPoint.java
@Component
public class SpelPolicyDecisionPoint implements PolicyDecisionPoint {
  private final ExpressionParser parser = new SpelExpressionParser();
  private final PolicyRepository repo; // JPA or DAO
  public SpelPolicyDecisionPoint(PolicyRepository repo){ this.repo = repo; }

  @Override
  public Decision evaluate(String action, String resourceType, Map<String,
Object> ctx){
    var policies = repo.findActivePolicies(resourceType, action);
    for (Policy p : policies){
      for (PolicyRule r : p.rulesByPriority()){
        var evalCtx = new StandardEvaluationContext(ctx);
        Boolean matched =
parser.parseExpression(r.getConditionExpr()).getValue(evalCtx, Boolean.class);
        if (Boolean.TRUE.equals(matched)){
          return new Decision(r.getEffect() == EffectType.ALLOW ?
Effect.ALLOW : Effect.DENY,
                              "policy="+p.getName()+" rule="+r.getId(),
Map.of());
        }
      }
      if (p.getDefaultEffect() == EffectType.ALLOW) return new
Decision(Effect.ALLOW, "default", Map.of());
    }
    return new Decision(Effect.DENY, "no-match", Map.of());
  }
}
```

Example `condition_expr` (SpEL):

- `#subject.sub == #resource.ownerId`
- `#subject.roles.contains('admin')`
- `#env.method == 'GET' and #resource.classification != 'secret'`

4C) External PDP (OPA) – optional, same interface

```
// OpaPolicyDecisionPoint.java (simplified)
@Component
public class OpaPolicyDecisionPoint implements PolicyDecisionPoint {
  private final WebClient opa;
  public OpaPolicyDecisionPoint(WebClient.Builder builder){
    this.opa = builder.baseUrl("http://opa:8181/v1/data/app/allow").build();
  }
  @Override
  public Decision evaluate(String action, String resourceType, Map<String,
Object> ctx){
    Map<String,Object> input = Map.of("action", action, "resourceType",
resourceType, "ctx", ctx);
    var result = opa.post().bodyValue(Map.of("input",
input)).retrieve().bodyToMono(Map.class).block();
    boolean allow =
Boolean.TRUE.equals(((Map)result.get("result")).get("allow"));
    return new Decision(allow ? Effect.ALLOW : Effect.DENY, "opa", Map.of());
  }
}
```

5) HTTP PEP via AuthorizationManager

```
// AbacAuthorizationManager.java
@Component
public class AbacAuthorizationManager implements
AuthorizationManager<RequestAuthorizationContext> {
  private final AttributeResolverChain attrs; private final PolicyDecisionPoint
pdp;
  public AbacAuthorizationManager(AttributeResolverChain a, PolicyDecisionPoint
p){ this.attrs=a; this.pdp=p; }

  @Override
  public AuthorizationDecision check(Supplier<Authentication> authentication,
RequestAuthorizationContext ctx){
    Authentication auth = authentication.get();
    Map<String,Object> context = attrs.buildContext(ctx, auth);
    String action = ctx.getRequest().getMethod();
    String resourceType = ((Map<String,Object>)context.getOrDefault("resource",
```

```
Map.of())).getOrDefault("type","unknown").toString();
    var decision = pdp.evaluate(action, resourceType, context);
    boolean granted = decision.effect() == PolicyDecisionPoint.Effect.ALLOW;
    return new AuthorizationDecision(granted);
  }
}
```

6) Method-Level PEP (optional) with custom expression

```
// AbacPermissionEvaluator.java
@Component
public class AbacPermissionEvaluator implements PermissionEvaluator {
  private final PolicyDecisionPoint pdp;
  private final AttributeResolverChain attrs;
  public AbacPermissionEvaluator(PolicyDecisionPoint p, AttributeResolverChain
a){ this.pdp=p; this.attrs=a; }

  @Override
  public boolean hasPermission(Authentication auth, Object targetDomainObject,
Object permission){
    var ctx = new RequestAuthorizationContext(new DummyHttpServletRequest(),
null);
    Map<String,Object> map = attrs.buildContext(ctx, auth);
    map.put("resource", Map.of("type",
targetDomainObject.getClass().getSimpleName().toLowerCase(), "id",
extractId(targetDomainObject)));
    return pdp.evaluate(String.valueOf(permission), "object", map).effect() ==
PolicyDecisionPoint.Effect.ALLOW;
  }
  @Override public boolean hasPermission(Authentication a, Serializable id,
String type, Object perm){ return false; }
}

// MethodSecurityConfig.java
@Configuration
public class MethodSecurityConfig extends MethodSecurityExpressionHandler {
  private final AbacPermissionEvaluator pe;
  public MethodSecurityConfig(AbacPermissionEvaluator pe){ this.pe = pe; }
  @Bean
  public DefaultMethodSecurityExpressionHandler
methodSecurityExpressionHandler(){
    var h = new DefaultMethodSecurityExpressionHandler();
    h.setPermissionEvaluator(pe);
    return h;
  }
}
```

```
// Usage
@PreAuthorize("hasPermission(#doc, 'read')")
public Document get(Document doc) { ... }
```

7) Controller Example (hybrid RBAC + ABAC)

```
@RestController
@RequestMapping("/documents")
public class DocumentController {
  private final DocumentService svc;
  public DocumentController(DocumentService svc){ this.svc = svc; }

  // HTTP PEP already enforces ABAC; keep RBAC role check too
  @GetMapping("/{id}")
  @PreAuthorize("hasAnyRole('doc_reader','admin')")
  public DocumentDto get(@PathVariable String id){
    return svc.get(id);
  }
}
```

8) Example Policy Records (SpEL)

```
policy: name="docs_view", version=1, default=DENY
rule(priority=10, action="GET", resource_type="document", effect=ALLOW,
     condition_expr="#subject.roles.contains('admin')")
rule(priority=20, action="GET", resource_type="document", effect=ALLOW,
     condition_expr="#subject.sub == #resource.ownerId")
rule(priority=90, action="GET", resource_type="document", effect=DENY,
     condition_expr="#resource.classification == 'secret'")
```

9) Migration Plan (RBAC → Hybrid)

1. Enable Resource Server (JWT) and keep RBAC checks untouched.
2. Introduce ABAC `AuthorizationManager` on read-only endpoints; shadow-mode logging only.
3. Add DB policy store and admin workflow; enable hard enforcement per endpoint.
4. Expand to write operations; deprecate coarse RBAC permissions over time.

Milestones

- **M0** (1 wk): Enable JWT resource server; add logging-only PDP; collect metrics.
- **M1** (2 wks): Implement SpEL PDP + AttributeResolvers; enforce on two services.
- **M2** (2 wks): Policy store + admin APIs + audit logs.
- **M3** (1–2 wks): Optional OPA integration; canary on one service.

- **M4** (ongoing): Authoring UX, caching, tenant isolation, blue/green policy rollout.

Visual Diagrams (Structural & Sequence)

Structural — Component Diagram (System-Level)

```
@startuml
skinparam componentStyle rectangle
rectangle "Client" as Client
node "API Gateway" as GW {
  [Gateway PEP] as GWPEP
  [OPA Sidecar] as GWOPA
}
node "Service A Pod" as SvcA {
  [Service A] as SA
  [PEP A] as PEPA
  [Attribute Resolver Chain] as AR
  [OPA Sidecar] as OPA_A
}
node "Service B Pod" as SvcB {
  [Service B] as SB
  [PEP B] as PEPB
  [OPA Sidecar] as OPA_B
}
node "Policy Bundle Server" as PBS
node "IdP (OIDC)" as IdP

Client --> GWPEP : HTTPS (JWT)
GWPEP --> GWOPA : authz query
GWPEP --> SA : forward on allow
SA --> PEPA : internal authz check (optional)
PEPA --> AR : resolve attributes
AR --> OPA_A : PDP query
GWOPA --> PBS : pull bundles (poll)
OPA_A --> PBS : pull bundles (poll)
Client --> IdP : login/token
@enduml
```

Structural — Package Diagram (Code Modules)

```
@startuml
package "authz.core" {
  class PolicyDecisionPoint
  class Decision
  interface AttributeResolver
```

```
  class AttributeResolverChain
}
package "authz.pdp" {
  class OpaPolicyDecisionPoint
}
package "authz.pep" {
  class AbacAuthorizationManager
}
package "authz.attr" {
  class JwtClaimsAttributeResolver
  class ResourceParamAttributeResolver
  class ResourceDbAttributeResolver
}

PolicyDecisionPoint <|.. OpaPolicyDecisionPoint
AttributeResolverChain o-- AttributeResolver
AbacAuthorizationManager ..> PolicyDecisionPoint
AbacAuthorizationManager ..> AttributeResolverChain
@enduml
```

Structural — Deployment Diagram (Production w/ Sidecars)

```
@startuml
skinparam componentStyle rectangle
node "Kubernetes Cluster (prod)" {
  node "Namespace: prod" {
    node "Gateway Pod" {
      [Gateway App]
      [OPA Sidecar]
    }
    node "Service A Pod" {
      [Service A]
      [OPA Sidecar A]
    }
    node "Service B Pod" {
      [Service B]
      [OPA Sidecar B]
    }
    node "Policy Bundle Server" as PBS
    node "Observability" {
      [Logs]
      [Metrics]
      [Tracing]
    }
  }
}
```

```
[Gateway App] -down- [OPA Sidecar]
[Service A] -down- [OPA Sidecar A]
[Service B] -down- [OPA Sidecar B]
PBS --> [OPA Sidecar]
PBS --> [OPA Sidecar A]
PBS --> [OPA Sidecar B]
[OPA Sidecar] --> [Logs]
[OPA Sidecar A] --> [Logs]
[OPA Sidecar B] --> [Logs]
@enduml
```

Sequence — HTTP Authorization (Gateway PEP with Cache)

```
@startuml
actor User
participant Gateway as GW
participant PEP
participant Attrs as "AttributeResolverChain"
participant Cache
participant OPA
User -> GW: HTTP request (JWT)
GW -> PEP: authorize(request)
PEP -> Attrs: resolve {subject, resource, env}
Attrs --> PEP: attributes
PEP -> Cache: get(hash(input))
alt cache hit
  Cache --> PEP: decision
else miss
  PEP -> OPA: POST /v1/data/app/allow {input}
  OPA --> PEP: {result:{allow:true}}
  PEP -> Cache: put(key, decision, ttl)
end
PEP --> GW: ALLOW/DENY
GW -> User: 200/403
@enduml
```

Sequence — Method-Level Authorization (Service Layer)

```
@startuml
participant Controller as C
participant Service as S
participant MethodPEP as "PermissionEvaluator"
participant Attrs as "AttributeResolvers"
participant OPA
C -> S: getDocument(id)
```

```
activate S
S -> MethodPEP: hasPermission(doc, 'read')
MethodPEP -> Attrs: resolve ctx
Attrs --> MethodPEP: ctx
MethodPEP -> OPA: evaluate(action='READ', resourceType='document', ctx)
OPA --> MethodPEP: allow = true
MethodPEP --> S: true
S --> C: Document
@enduml
```

Sequence — Policy Bundle Authoring & Rollout

```
@startuml
actor PolicyAuthor as Author
participant Git
participant CI as CI/CD
participant BundleServer as PBS
participant OPA as OPA_Sidecars
Author -> Git: push Rego + data
Git -> CI: webhook
CI -> CI: test (opa fmt, test, eval)
CI -> PBS: publish bundle (versioned)
OPA_Sidecars -> PBS: poll + download bundle
OPA_Sidecars -> OPA_Sidecars: hot-reload policies
@enduml
```

Gathering Results

- Record decision logs (allow/deny, rule, attributes) → ship to SIEM.
- SLOs: <50ms p95 PDP latency; zero authz regressions during shadow period; <1% policy evaluation errors.
- Game-days: simulate token claims changes and policy hot-reload.

- Record decision logs (allow/deny, rule, attributes) → ship to SIEM.
- SLOs: <50ms p95 PDP latency; zero authz regressions during shadow period; <1% policy evaluation errors.
- Game-days: simulate token claims changes and policy hot-reload.

Need Professional Help in Developing Your Architecture?

Please contact me at sammuti.com :)