**Final Architecture Overview for ABC Application**

**1. Shared Libraries (Spring Boot Starters)**

| **Starter Name** | **Purpose & Contents** |
| --- | --- |
| **abc-spring-boot-starter-core** | Core infrastructure and utilities shared across microservices: |

* ConfigurationManager (externalized config)
* ExceptionHandlers (global error handling)
* Logging (centralized logging setup)
* HealthCheckers (using Spring Boot Actuator) |  
  | **abc-spring-boot-starter-authorization** | Authorization framework:
* AuthorizationManager (annotations, interceptors, AOP)
* Flexible, declarative authorization handling |  
  | **abc-spring-boot-starter-cloud-aws** | AWS integration:
* AWS SDK wrappers
* Configuration and helper utilities for AWS services |  
  | **abc-spring-boot-starter-monitoring** | Monitoring and observability:
* Metrics collection (Micrometer, Prometheus)
* Audit logging
* Health and performance monitoring |  
  | **abc-spring-boot-starter-idp-okta** (and future providers) | Identity Provider Adapter library:
* Implements IdentityProviderAdapter interface for Okta
* Encapsulates Okta API calls, data transformation, error handling
* Enables pluggable identity provider integration |

**2. Microservices**

| **Microservice Name** | **Responsibility** |
| --- | --- |
| **AuthenticationService** | Authentication and token management integrating with external IdPs (Okta, etc.) |
| **UserService** | Business logic for user management; depends on IdentityProviderAdapter interface |
| **GroupService** | Business logic for group management; depends on IdentityProviderAdapter interface |
| **ApplicationService** | Manages applications metadata and lifecycle; domain-focused, provider-agnostic |
| **AttributeService** | Manages extended attributes and metadata |
| **AccessControlService** | Manages authorization policies, scopes, and access control assignments to groups and users |
| **ClientCredentialsService** | Manages OAuth2 client registrations and credentials |

**3. Identity Provider Adapter Pattern**

* Define a **provider-agnostic interface** IdentityProviderAdapter in domain services.
* Use **provider-specific adapter libraries** (e.g., abc-spring-boot-starter-idp-okta) that implement this interface.
* Inject the adapter implementation into services via dependency injection, enabling easy provider replacement without service code changes.
* Adapter libraries handle all external API calls, data transformations, and error handling internally, avoiding extra HTTP calls between microservices.

**4. Architectural Highlights**

* **Modularity:** Clear separation between core infrastructure, authorization, cloud integration, monitoring, and identity provider integration.
* **Provider Agnostic:** Domain services remain decoupled from specific identity providers, promoting flexibility and future-proofing.
* **Performance Optimized:** Embedding identity provider adapters as libraries reduces network overhead and latency.
* **Scalability:** Microservices are domain-focused and independently deployable and scalable.
* **Consistency:** Shared starters ensure uniform logging, configuration, authorization, and monitoring across all services.
* **Extensibility:** Easy to add new identity providers or cloud integrations by adding new adapter libraries or starters.

**5. Technology & Design Patterns**

* **Spring Boot Starters & Auto-Configuration** for reusable libraries.
* **Aspect-Oriented Programming (AOP)** for cross-cutting concerns like authorization and logging.
* **Dependency Injection** for adapter implementations.
* **Domain-Driven Design (DDD)** for microservice boundaries.
* **Externalized Configuration** for secure and flexible environment management.
* **Health Checks & Metrics** via Spring Boot Actuator and Micrometer.

If you want, I can help you with detailed project structures, sample code, and API contracts for each microservice and library!