Feature/Capability	**Description**	**Value / Benefit**
Extensive Metadata	Includes attributes such as `ruleName`, `uid`, `domain`, `description`, `approvedBy`, `createdDate`, `status`, etc.	• Provides **governance** and **auditability**. • Allows easy **version management** and change tracking. • Improves **collaboration** with clear, in-rule documentation.
Security & Permissions	Defines who can view, edit, copy, or execute the rule (`security.permissions`). Also supports log encryption, field-level encryption, etc.	• Ensures **compliance** with internal or external regulations (e.g., GDPR). • Manages **fine-grained access control** to sensitive rules and data.
Caching & Performance	Offers a `cache` block with parameters like `cachable`, `ttl`, and a custom cache key.	• Allows caching of rule results to **reduce compute overhead** for frequently-run rules. • Improves **performance** and lowers response times.
Rich Data Context	Defines data sources (databases, APIs, GraphQL, AWS Lambda, S3), connections, queries, and how to fetch/transform/aggregate data in a `dataContext` block.	• Centralizes **data fetching** and **transformation** logic. • Allows a **federated approach** for multiple data sources in a single rule definition.
Advanced Rule Logic	Supports logical operators (`and`, `or`, `not`), array handling (`any`, `all`), aggregator logic	• Enables **complex conditions** (e.g., aggregator, nested

Feature/Capability	**Description**	**Value / Benefit**
	(`sum`, `count`), functions, external service calls, and more.	filters) without additional code. - Broad set of **operators** for real-world scenarios (e.g., partial string matches, `inDataset` checks, etc.).
Preconditions & Postconditions	A `preCondition` block checks data or conditions before running the main logic. Post-rule actions (`onSuccess`, `onFailure`, `onError`, `afterRun`) define follow-up steps.	- **PreCondition** ensures required data/state is present before evaluating. - **PostCondition** or **actions** let you orchestrate subsequent workflows.
Actions & Orchestration	Defines `actions` for each outcome path (success, failure, error). Supports sending emails/SMS, logging, queue operations, invoking other rules, custom scripts, etc.	- Combines **rule evaluation** with **workflow automation**. - Minimizes the need for separate orchestration services.
Logging & Monitoring	Configurable logging (`logging` block) with `defaultLevel`, `defaultDestination`, `appendTimestamp`, etc. Also includes `performance` metrics like `expectedExecutionTimeMs` and `alertThresholdMs`.	- Provides **observability** into rule execution (info, debug, error logs). - Allows **monitoring** of execution time and proactive alerting on timeouts.
Parameterization	A `parameters` section that defines placeholders (e.g., `$customerId`) for dynamic values, each	- Allows **reusability** of the same rule in multiple contexts by

Feature/Capability	**Description**	**Value / Benefit**
	with `mandatory`, `default`, and `description` fields.	passing different parameter values. - Makes the rule **flexible** and environment-agnostic.
Rule Lifecycle Management	Includes fields like `ruleVersion`, `dependencies`, `usageScopes`, `ruleExpiry`, etc.	- Enables **versioned deployments** of rules with **go-live** and **expiry** dates. - Tracks **dependencies** among rules or with external references.
Error Handling	`onError` sections define fallbacks, retries, or custom behaviors if an exception occurs during rule evaluation.	- Ensures **robustness** by handling unexpected errors gracefully. - Minimizes system crashes with **fallback strategies**.
Post-Execution Metadata	A `result` block describing how outcomes or extracted data should be structured and returned (`failureReasons`, `returnValues`, etc.).	- Facilitates **auditing** by storing detailed result data. - Simplifies **downstream consumption** of results, since it's in a known format.