The Smart Icon Mapper

A Deep Dive into the Algorithm for Automated Icon Selection

The Quest for the Perfect Icon

The Smart Icon Mapper automatically selects the most appropriate icon for a diagram node based on its label and type. It follows a three-stage process to ensure the best possible match, from highly specific custom icons to sensible generic defaults.

1. Custom Icon Matching

High-fidelity matching using a smart scoring algorithm.

2. Built-in Rules

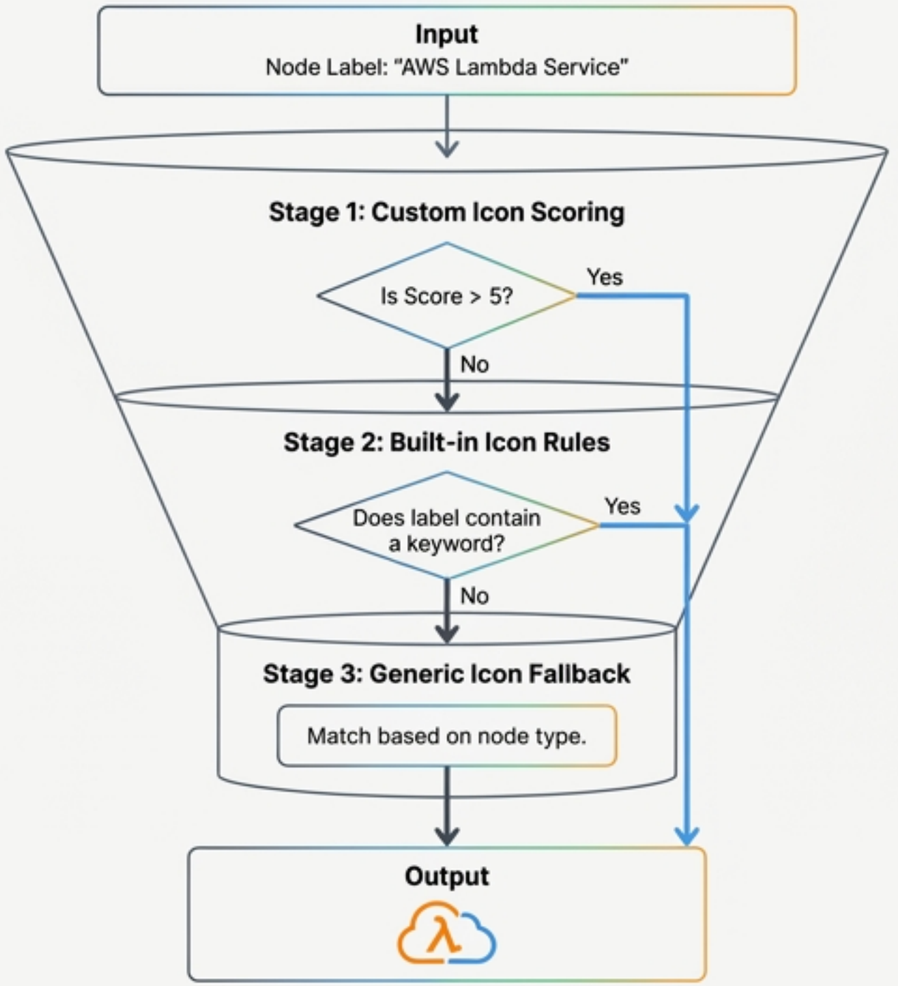
Fallback to a curated library of common tech icons.

3. Generic Defaults

A final safety net based on the node's type.

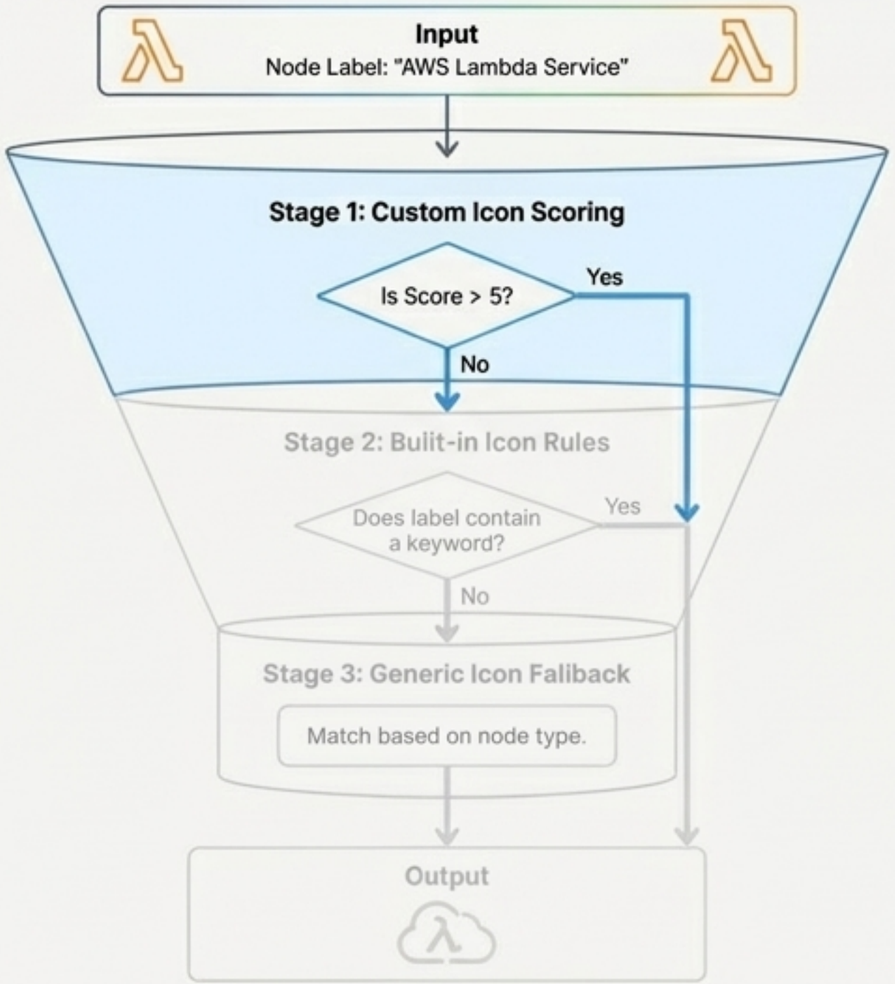
Following the Decision Flow

We will explore each stage of the icon mapping process, from the most sophisticated logic to the final fallbacks. This diagram will guide us through the journey.



Stage 1: Custom Icon Matching with Token-Based Scoring

The system first attempts to find a match from user-uploaded custom icons. This uses a ‘smart’ scoring algorithm designed to be robust against multi-word labels, capitalization, and generic terms.



Step A: Normalization and Tokenization

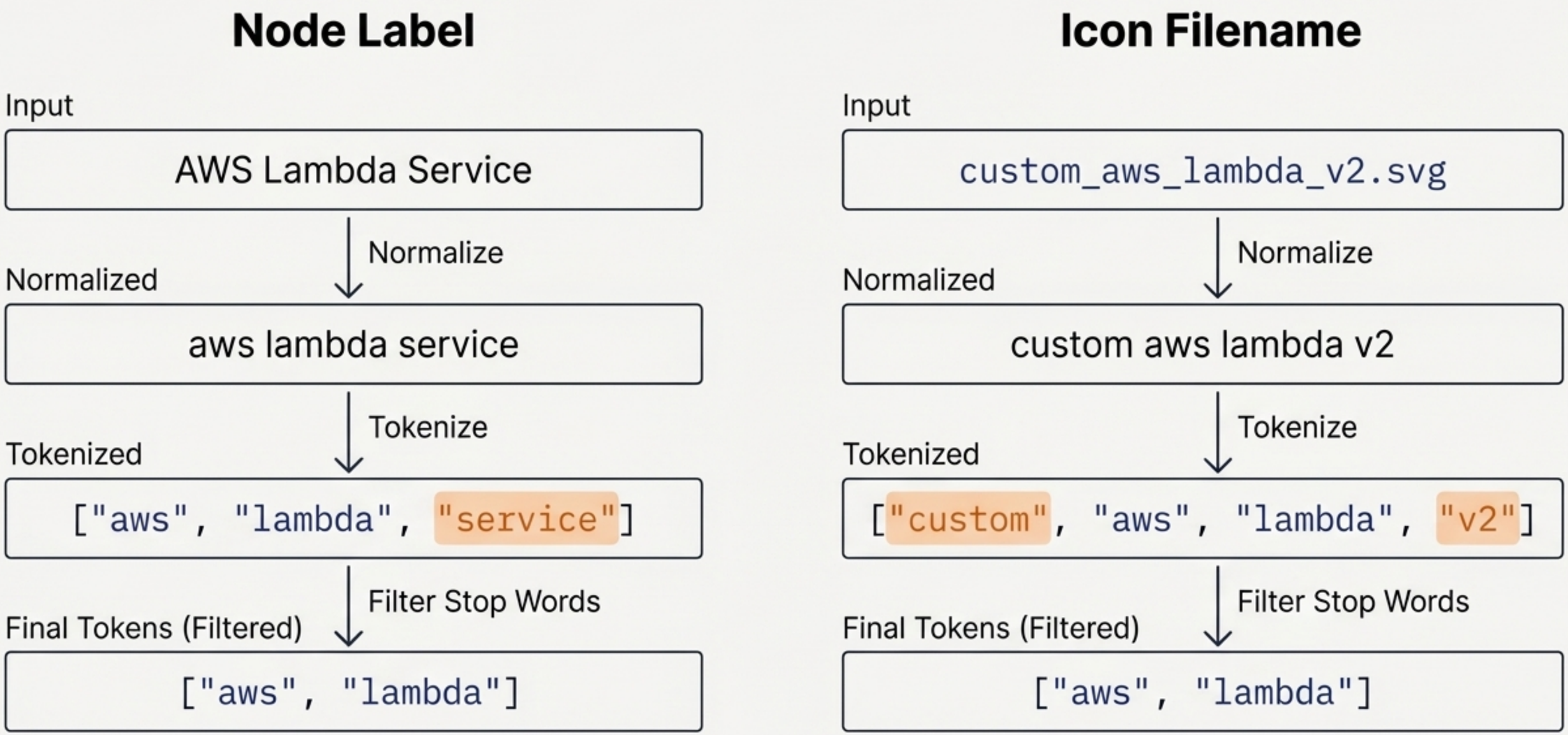
Before comparison, both the node label and icon filename are cleaned and broken down into essential keywords (tokens). This ensures a fair comparison.

1. **Normalization:** Convert to lowercase. Replace special characters (`-`, `_`, `.`) with spaces.
2. **Tokenization:** Split the string into individual words.
3. **Stop Word Removal:** Discard generic, low-value words to prevent false positives.

Stop Words List

```
service   server   app   application   component   node   system
database   db   icon   logo   v1   v2   v3
```

Normalization in Action: An Example



Step B: The Scoring Algorithm

A Match Score is calculated by comparing the filtered tokens from the label against the tokens from the icon name. A higher score indicates a better match. The icon with the highest score is chosen, provided it meets the minimum threshold.

Match Type	Points	Description
Exact Token Match	**+10**	The word appears exactly in both sets of tokens.
Partial Token Match	**+3**	One word is a substring of another (e.g., "**mongo**" in "**mongodb**").
Exact Phrase Bonus	**+5**	The full normalized string matches exactly.

Selection Threshold

A match is only considered valid if the **`Total Score > 5`**.

Scenario 1: A Strong Match

Let’s see how a well-named icon gets a high score for a corresponding node.

Input	**Node Label**
	`AWS Lambda Service`
	↓
	`["aws", "lambda"]`
	
	Icon Filename
	`custom_aws_lambda.svg`
	↓
	`["aws", "lambda"]`	

Calculation	`"aws"` — “aws” matches “aws” (Exact Match) — `"aws"`	**+10 points**
	`"lambda"` — “lambda” matches “lambda” (Exact Match) — `"lambda"`	**+10 points**
Result	Total Score: **20** Outcome: **MATCH** (20 > 5)	

Scenario 2: Preventing a False Positive

The stop word list and scoring threshold are crucial for preventing incorrect matches between nodes that share generic terms.

Input

Node Label	Icon Filename
`User Service`	`custom_auth_service.svg`
↓	↓
`["user"]`	`["auth"]`

Calculation

Comparing **["user"]** vs **["auth"]**

No token matches.

Result

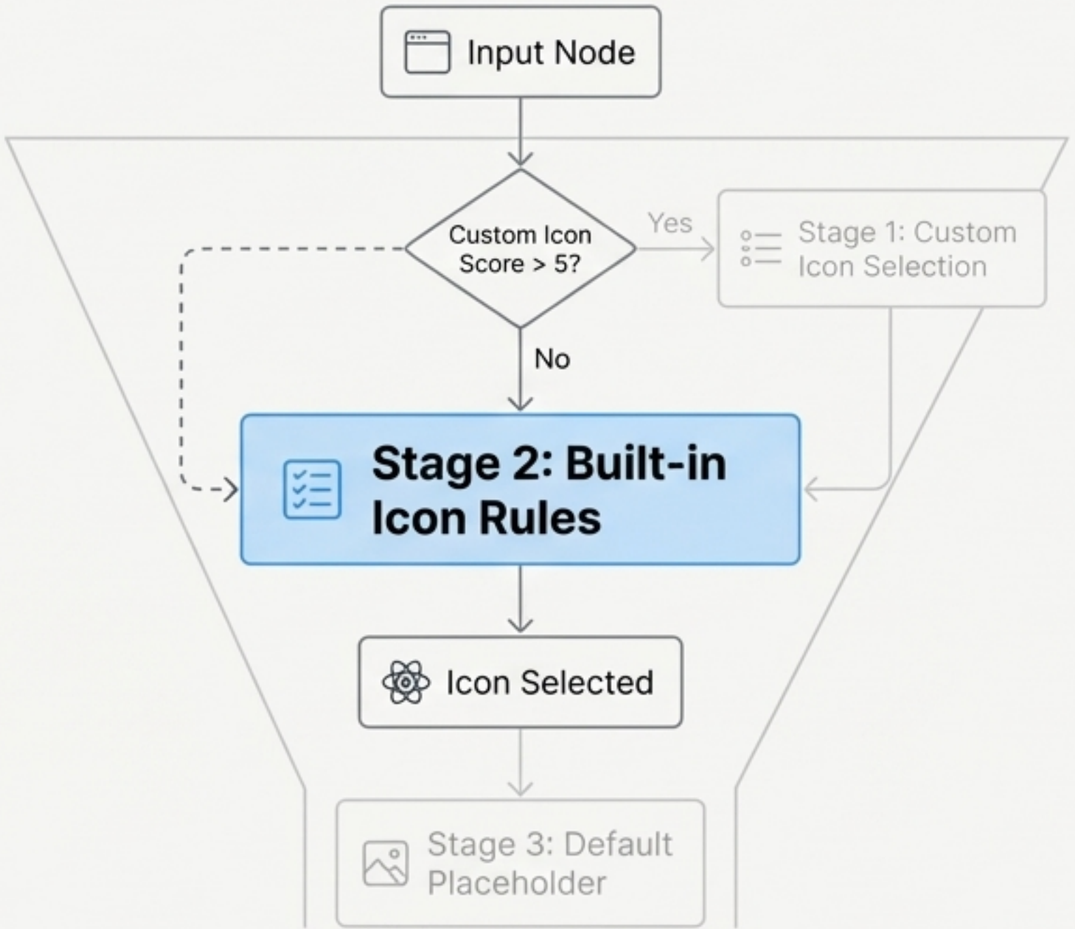
0

Total Score

NO MATCH (0 is not > 5)

Stage 2: Fallback to Built-in Icons

If no custom icon achieves a score greater than 5, the system falls back to a curated list of hardcoded rules for common technologies, using standard icons from the React Icons library.



Simple Keyword Matching for Common Technologies

The fallback logic uses a simple substring check on the normalized node label to find a match. This covers a wide range of popular services and technologies.

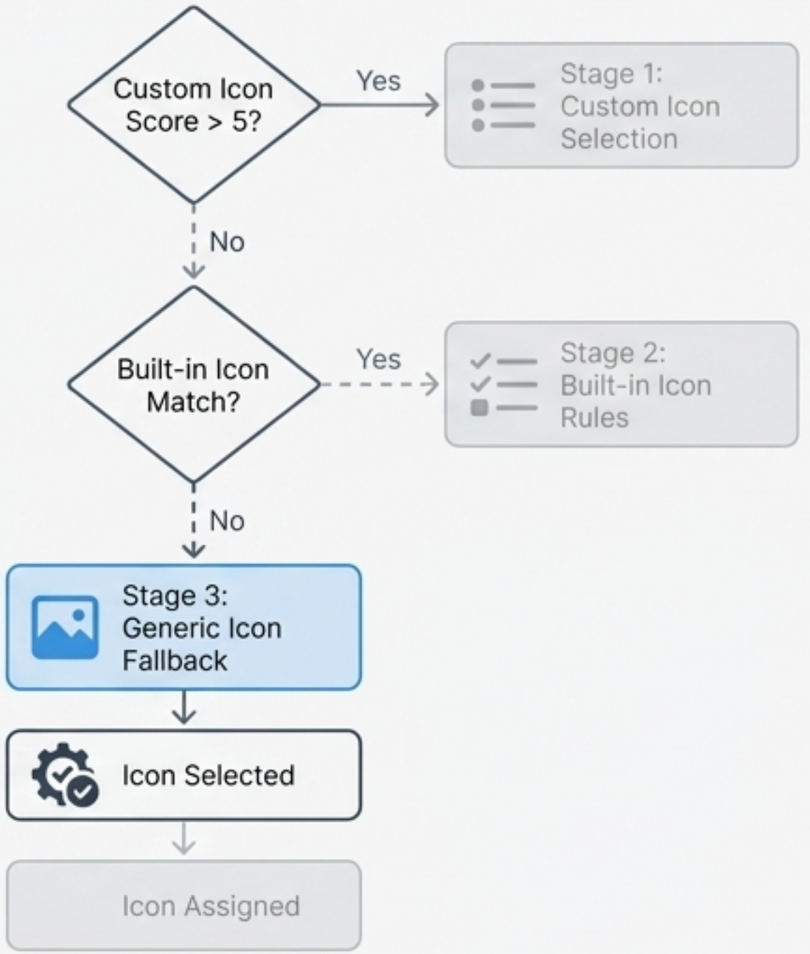
```
// Example Logic
if (text.includes('aws') || text.includes('amazon')) return <FaAws />;
if (text.includes('redis')) return <FaRedis />;
if (text.includes('postgres')) return <DiPostgresql />;
```

Categories Covered

- Cloud Providers (AWS, Azure, GCP)
- Infrastructure & DevOps (Kubernetes, Docker)
- Databases (Redis, Postgres, Mongo)
- Backend & Languages (Node, Python, Java)
- Frontend & Mobile (React, Vue)
- Messaging (Kafka, RabbitMQ)
- SaaS (Stripe, Auth0)

Stage 3: The Generic Default Fallback

If no custom or specific built-in icon is found, the system assigns a generic icon based on the node's fundamental type. This ensures every node has a visually representative icon.



Generic Icons Assigned by Node Type

The final fallback maps a small set of node types to a default icon, with minor contextual adjustments.

Node Type	Default Icon & Logic
`database`	 `FaDatabase`
`queue`	 `MdQueue`
`group`	 `FaCloud`
`client`	 `FaLaptopCode` (or  **FaMobileAlt** if “mobile” is in the label)
`service`	 `FaServer` (or  **MdApi** if “api” is in the label)

Putting It All Together: From Smart to Simple

The Smart Icon Mapper uses a cascading logic to guarantee the best icon is always found, prioritizing your specific custom icons before falling back to sensible defaults.

Getting the Best Results

Guidance for Naming Custom Icons
To maximize match scores for your custom icons, use **descriptive**, multi-word filenames that mirror your node labels.

Example
Node Label **Production Redis Cache**
✅ GOOD: `icon-prod-redis-cache.svg` ❌ POOR: `redis.svg`