Yaga2 Anomaly Detection Guide

Welcome to the comprehensive guide for the Yaga2 Anomaly Detection system. This documentation covers how anomalies are detected, evaluated against SLOs, and managed through their incident lifecycle.

What is Yaga2?

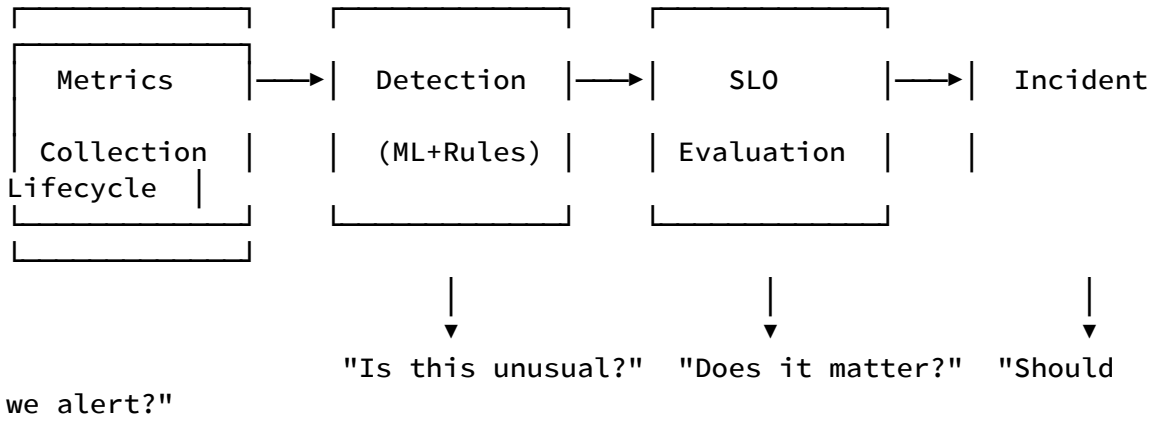
Yaga2 is an ML-based anomaly detection system for Smartbox services. It:

- **Detects** statistical anomalies using Isolation Forest and pattern matching
- **Evaluates** anomalies against operational SLO thresholds
- **Adjusts** severity based on business impact
- **Manages** incident lifecycle with confirmation and grace periods

Key Concepts

Concept	Description
Anomaly	A statistically unusual metric pattern detected by ML
SLO Evaluation	Comparing metrics against operational thresholds
Severity	Alert priority: `low`, `medium`, `high`, `critical`
Incident	A confirmed anomaly requiring attention
Fingerprint	Unique identifier for an anomaly pattern

Pipeline Flow



Quick Reference

When does an alert fire?

1. ML detects anomaly → Pattern interprets it → Anomaly created as `SUSPECTED`
2. Anomaly persists for 2+ cycles → Status becomes `OPEN` → **Alert sent**
3. SLO evaluation adjusts severity based on operational impact

When does an incident resolve?

1. Anomaly not detected → Status becomes `RECOVERING`
2. Not detected for 3 cycles → Status becomes `CLOSED` → **Resolution sent**

Getting Started

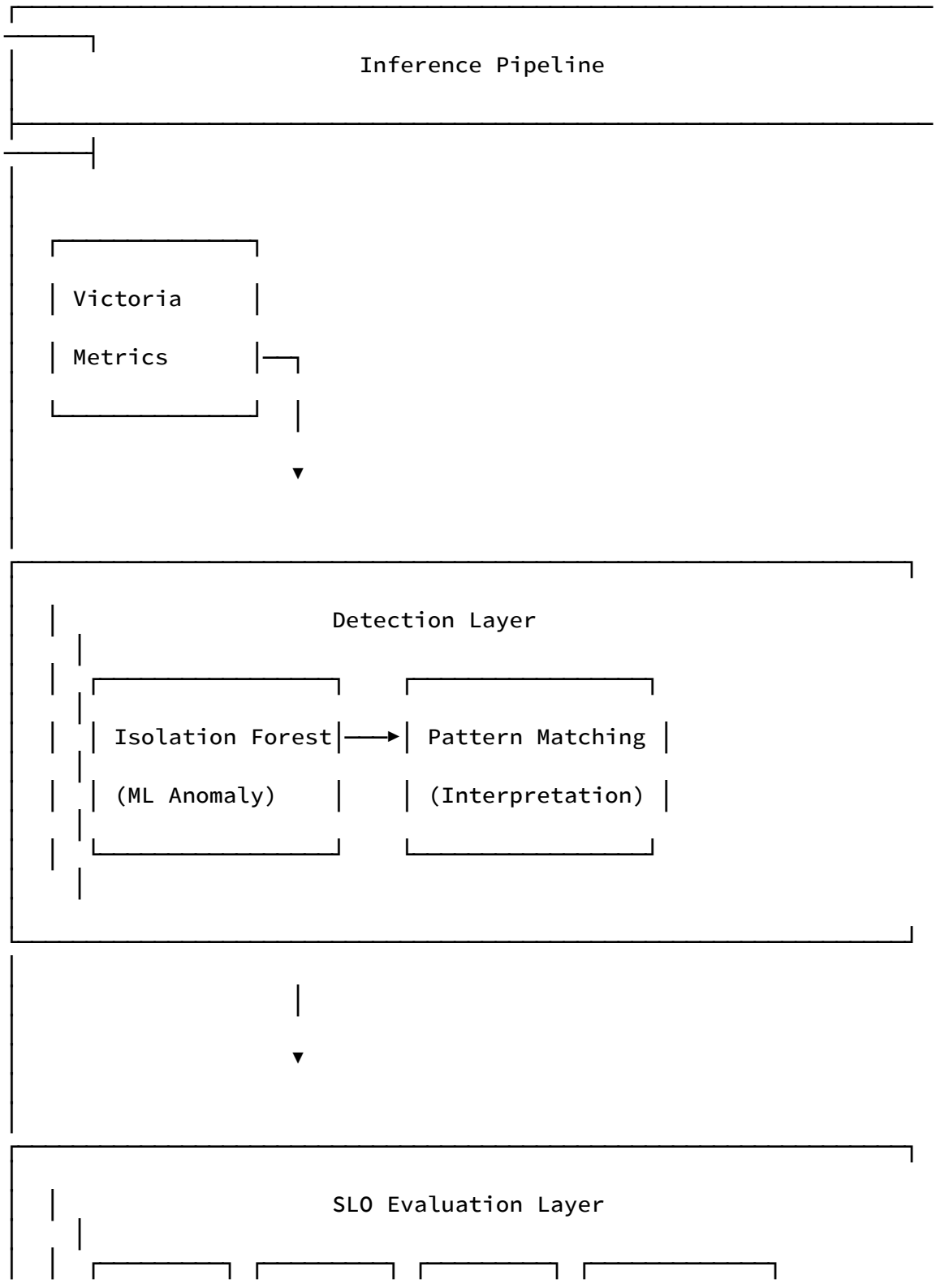
- Overview - High-level system architecture
- Detection Layer - How anomalies are detected
- SLO Evaluation - How severity is adjusted
- Incident Lifecycle - How alerts are managed

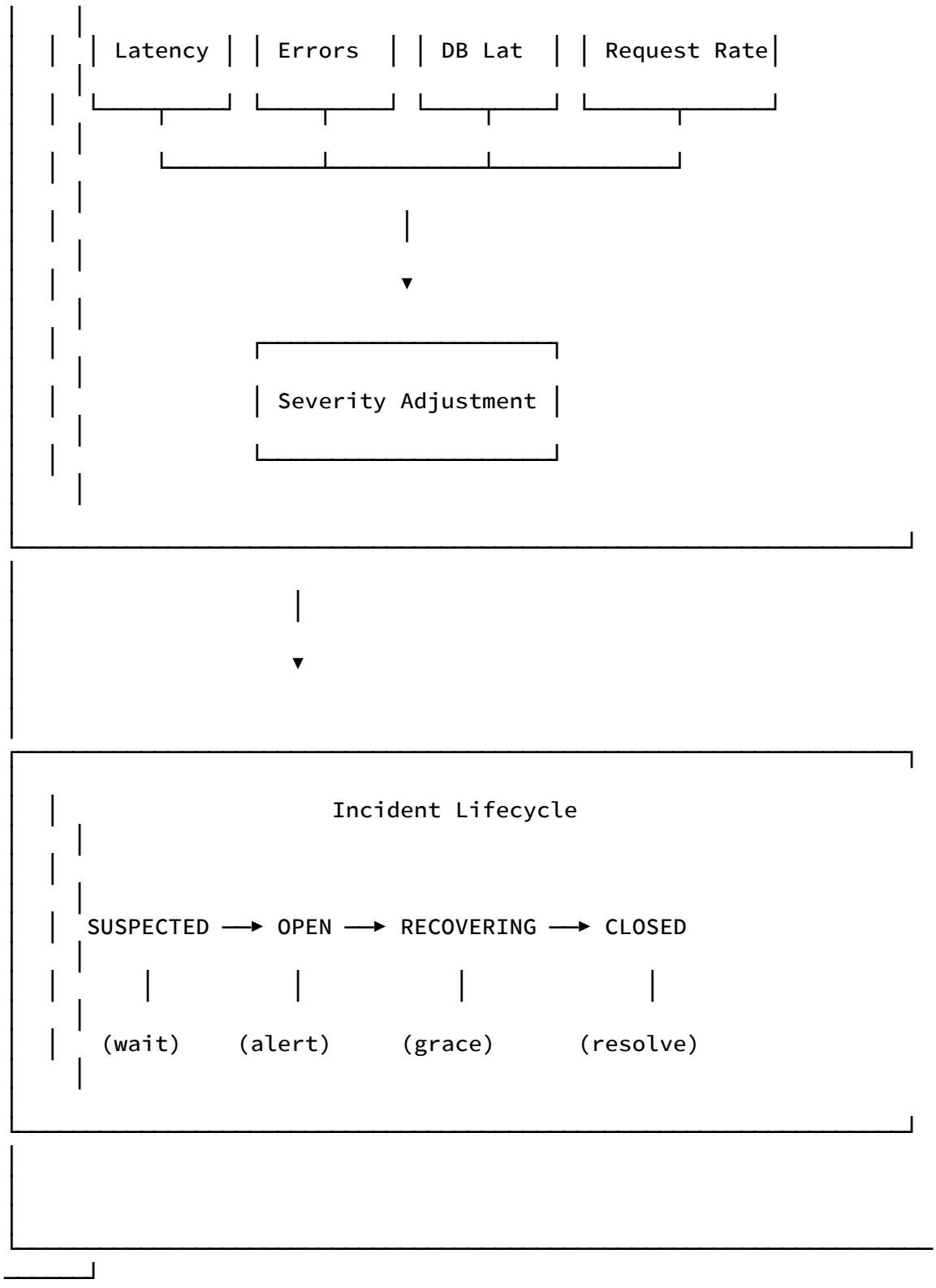
- Decision Matrix - Complete reference table

System Overview

The Yaga2 anomaly detection system processes metrics through multiple layers to produce actionable alerts.

Architecture




Latency
Errors
DB Lat
Request Rate
Severity Adjustment
Incident Lifecycle
SUSPECTED
OPEN
RECOVERING
CLOSED
(wait)
(alert)
(grace)
(resolve)

Metrics Collected

Metric	Description	Unit
`request_rate`	Incoming requests per second	req/s
`application_latency`	Server-side processing time	ms
`client_latency`	External dependency call time	ms
`database_latency`	Database query time	ms
`error_rate`	Percentage of failed requests	ratio (0-1)

Time-Aware Detection

The system uses **time-aware models** trained separately for each behavioral period:

Period	Hours	Days
`business_hours`	08:00 - 18:00	Mon-Fri
`evening_hours`	18:00 - 22:00	Mon-Fri
`night_hours`	22:00 - 06:00	Mon-Fri
`weekend_day`	08:00 - 22:00	Sat-Sun
`weekend_night`	22:00 - 08:00	Sat-Sun

This prevents false positives from expected behavioral differences (e.g., low traffic at 3 AM is normal).

Two-Pass Detection

For accurate cascade analysis, detection runs in two passes:

1. **Pass 1**: Detect anomalies for all services (no dependency context)
2. **Pass 2**: Re-analyze latency anomalies with dependency context

This enables identifying root cause services in dependency chains.

Output

Each inference run produces:

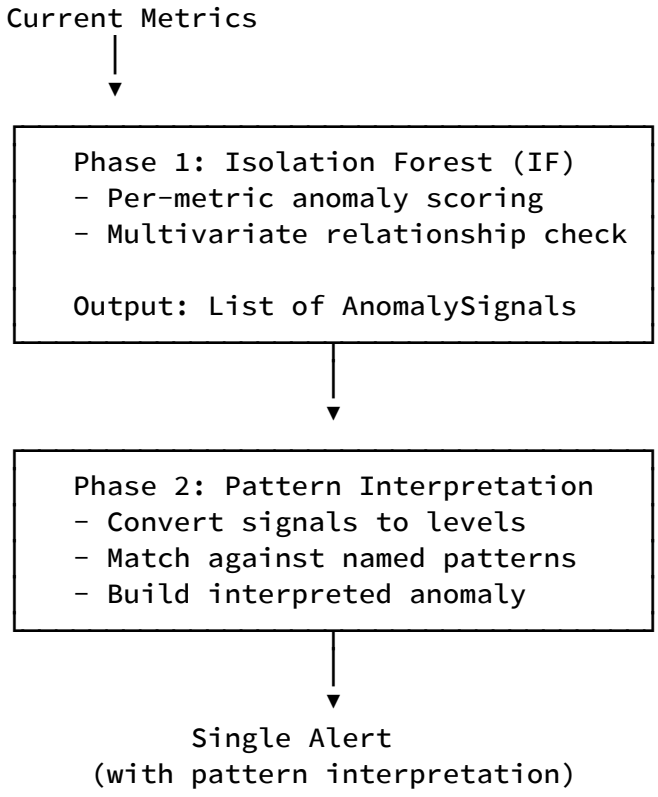
- **Anomaly alerts** for services with detected issues
- **Resolution notifications** for incidents that cleared
- **Enrichment data** (exceptions, service graph) when available

See API Payload for the complete output format.

Detection Layer

The detection layer identifies anomalies using a **sequential pipeline** where ML detection triggers pattern interpretation.

Pipeline Flow



Key Concepts

Anomaly Signal

When IF detects an anomaly, it produces a signal:

```
{
  "metric": "application_latency",
  "score": -0.35,
  "direction": "high",
  "percentile": 92.0
}
```

Metric Levels

Signals are converted to semantic levels for pattern matching:

Percentile	Level
> 95	`very_high`
90 - 95	`high`
80 - 90	`elevated`
70 - 80	`moderate`
10 - 70	`normal`
5 - 10	`low`
< 5	`very_low`

Lower-is-Better Metrics

Some metrics treat low values as improvements, not anomalies:

- `database_latency` - Faster queries are good
- `client_latency` - Faster dependencies are good
- `error_rate` - Fewer errors are good

For these metrics, when IF flags them as “low”, they are treated as `normal`.

Detection Methods

Method	Type	Best For
Isolation Forest	ML	Novel/unknown anomalies
Pattern Matching	Rule-based	Known operational scenarios

Sections

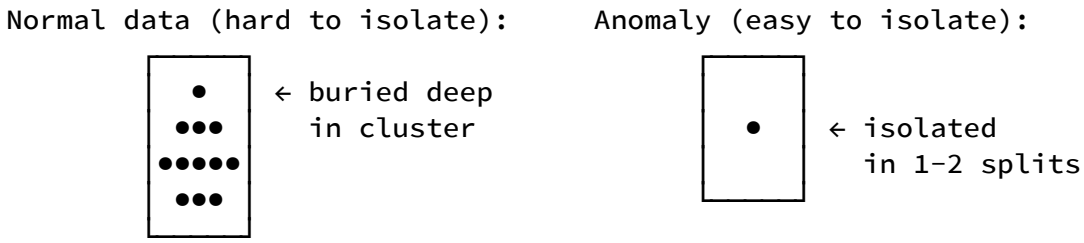
- Isolation Forest (ML) - How ML detection works
- Pattern Matching - Named patterns and interpretations
- Detection Pipeline - End-to-end detection flow

Isolation Forest (ML Detection)

Isolation Forest is an unsupervised machine learning algorithm that detects anomalies by measuring how easy it is to “isolate” a data point.

Core Insight

Anomalies are easier to isolate than normal points.



How It Works

1. **Random Partitioning**: Build random decision trees by:
    - Randomly selecting a feature (metric)
    - Randomly selecting a split value
    - Recursively partitioning data
2. **Path Length**: Count splits needed to isolate each point:
    - **Short path** = Easy to isolate = **Anomaly**
    - **Long path** = Hard to isolate = **Normal**
3. **Anomaly Score**: Average path length produces a score:
    - Score < 0 = Anomalous (more negative = more anomalous)
    - Score > 0 = Normal

Model Types

Univariate Models

Separate model for each metric:

Model	Detects
`request_rate_isolation`	Traffic anomalies
`application_latency_isolation`	Response time issues
`error_rate_isolation`	Error spikes
`client_latency_isolation`	Dependency slowness
`database_latency_isolation`	Database issues

Multivariate Model

One model considering all metrics together. Detects unusual **combinations**:

```
Example: Normal individually, anomalous together
{
  "request_rate": 500,         // Normal
  "application_latency": 50,   // Normal (fast!)
  "error_rate": 0.30           // High
}
// Fast responses + high errors = fast-fail pattern
```

Severity Thresholds

Scores map to severity levels (calibrated per model):

Severity	Percentile	Fallback Score
Critical	Bottom 0.1%	< -0.6
High	Bottom 1%	-0.6 to -0.3

Severity	Percentile	Fallback Score
Medium	Bottom 5%	-0.3 to -0.1
Low	Bottom 10%	>= -0.1

Training

Models are trained on 30 days of historical data with:

- **Temporal split**: 80% train, 20% validation (chronological)
- **Minimum samples**: 500 univariate, 1000 multivariate
- **Contamination**: Estimated from data distribution

Strengths

Strength	Why It Matters
Unsupervised	No labeled “anomaly” data required
Fast	Linear time complexity
Robust	Works with skewed distributions
Multivariate	Detects unusual combinations

Limitations

Limitation	Mitigation
No semantic understanding	Pattern matching adds context
Context-blind	Time-aware models help
False positives	SLO evaluation filters noise
Training data quality	Data quality checks

Pattern Matching

Pattern matching interprets ML signals by comparing metric levels against known operational scenarios.

How It Works

1. **Level Classification**: Each metric is classified based on IF signals
2. **Pattern Matching**: Current levels are compared against defined patterns
3. **Interpretation**: Matched patterns provide context and recommendations

Named Patterns

Traffic Patterns

Pattern	Conditions	Severity	Meaning
`traffic_surge_healthy`	high traffic, normal latency/errors	low	Handling load well
`traffic_surge_degrading`	high traffic, high latency, normal errors	high	Approachi capacity
`traffic_surge_failing`	high traffic, high latency, high errors	critical	Beyond capacity
`traffic_cliff`	very low traffic (sudden drop)	critical	May be unreachab

Error Patterns

Pattern	Conditions	Severity	Meaning
`error_rate_elevated`	normal traffic/latency, high errors	high	Specific operation failing
`error_rate_critical`	normal traffic/latency, very high errors	critical	Significant failures

Latency Patterns

Pattern	Conditions	Severity	Meaning
`latency_spike_recent`	normal traffic, high latency, normal errors	high	Recent change caused slowdown
`latency_elevated`	high latency above threshold	high	Response time degraded
`internal_latency_issue`	high app latency, healthy deps	high	Internal problem
`database_degradation`	high DB latency, normal app latency	medium	DB slow but compensating
`database_bottleneck`	high DB latency, DB	high	DB is constraint

Pattern	Conditions	Severity	Meaning
	>70% of total		
`downstream_cascade`	high client latency, >60% of total	high	External dependency issue

Fast-Fail Patterns

Pattern	Conditions	Severity	Meaning
`fast_rejection`	very low latency, very high errors	critical	Rejected before processing
`fast_failure`	low latency, high errors	critical	Quick failures
`partial_rejection`	low latency, moderate errors	high	Some requests rejected

Cascade Patterns

Pattern	Conditions	Severity	Meanir
`upstream_cascade`	high latency + upstream anomaly	high	Root cause upstrea
`dependency_chain_degradation`	high latency + multiple deps affected	high	Chain degrade

Pattern Definition Example

```
"traffic_surge_failing": PatternDefinition(
    name="traffic_surge_failing",
    conditions={
        "request_rate": "high",
        "application_latency": "high",
        "error_rate": "high",
    },
    severity="critical",
    message_template=(
        "Traffic surge overwhelming service: {request_rate:.1f}
req/s "
        "with {application_latency:.0f}ms latency and
{error_rate:.2%} errors"
    ),
    interpretation=(
        "Service at or beyond capacity. "
        "Both latency and errors elevated indicates system cannot
handle load."
    ),
    recommended_actions=[
        "IMMEDIATE: Scale horizontally or enable rate limiting",
        "INVESTIGATE: Confirm this is legitimate traffic",
        "PREPARE: Have rollback ready",
    ],
)
```

Detection Output

When a pattern matches:

```
{
  "traffic_surge_degrading": {
    "type": "multivariate_pattern",
    "pattern_name": "traffic_surge_degrading",
    "severity": "high",
    "score": -0.5,
    "description": "Traffic surge causing slowdown: 850 req/s
driving latency to 450ms",
    "interpretation": "Service slowing under load - approaching
capacity",
    "detection_method": "named_pattern_matching",
    "recommended_actions": [
      "IMMEDIATE: Scale horizontally if possible",
      "CHECK: Resource bottlenecks",
      "MONITOR: Error rate for impending failure"
    ],
    "metric_levels": {
      "request_rate": "high",
      "application_latency": "high",
      "error_rate": "normal"
    }
  }
}
```


Adding New Patterns

Edit `smartbox_anomaly/detection/interpretations.py`:

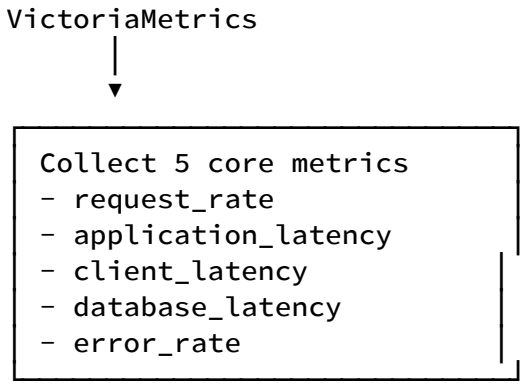
```
MULTIVARIATE_PATTERNS["my_new_pattern"] = PatternDefinition(
    name="my_new_pattern",
    conditions={
        "request_rate": "normal",
        "application_latency": "high",
        "error_rate": "low",
    },
    severity="medium",
    message_template="Description with {metrics}",
    interpretation="What this means...",
    recommended_actions=["Action 1", "Action 2"],
)
```

Detection Pipeline

The complete detection pipeline from metrics collection to anomaly output.

Pipeline Phases

Phase 1: Metrics Collection



Phase 2: Input Validation

Metrics are validated before processing:

Check	Action
NaN/Inf values	Replaced with 0.0
Negative rates	Capped at 0.0
Negative latencies	Capped at 0.0
Extreme latencies (>5 min)	Capped at 300,000ms
Extreme request rates (>1M/s)	Capped at 1,000,000
Error rates > 100%	Capped at 1.0

Validation warnings are included in the output.

Phase 3: Pass 1 Detection

First pass detects anomalies without dependency context:

```
For each service:
    1. Load time-aware model (based on current time period)
    2. Run univariate IF on each metric
    3. Run multivariate IF on combined metrics
    4. Collect anomaly signals
    5. Match signals against patterns
    6. Store result
```

Phase 4: Pass 2 Detection (Cascade Analysis)

Second pass re-analyzes services with latency anomalies:

```
For each service with latency anomaly:
    1. Build dependency context from Pass 1 results
    2. Check upstream dependencies for anomalies
    3. Re-run pattern matching with dependency info
    4. Add cascade_analysis if root cause found
```

Dependency Context:

```
{
  "upstream_status": "anomaly",
  "dependencies": {
    "vms": {"has_anomaly": true, "type": "database_bottleneck"},
    "titan": {"has_anomaly": true, "type": "latency_elevated"}
  },
  "root_cause_service": "titan"
}
```

Phase 5: SLO Evaluation

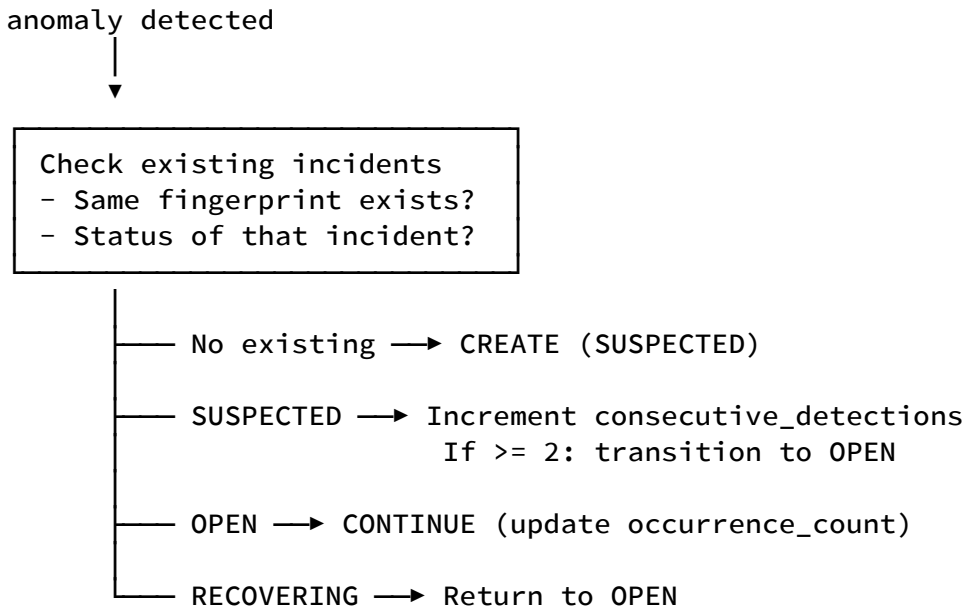
Each anomaly is evaluated against SLO thresholds:

```
For each detected anomaly:
    1. Evaluate latency vs SLO
    2. Evaluate error rate vs SLO
    3. Evaluate database latency vs baseline
    4. Check request rate (surge/cliff)
    5. Determine combined SLO status
    6. Adjust severity if needed
```

See SLO Evaluation for details.

Phase 6: Incident Lifecycle

Anomalies enter the incident lifecycle:



See Incident Lifecycle for details.

Output Structure

Final output for each service:

```
{
  "alert_type": "anomaly_detected",
  "service_name": "booking",
  "timestamp": "2024-01-15T10:30:00",
  "time_period": "business_hours",
  "overall_severity": "high",
  "anomaly_count": 1,

  "anomalies": {
    "latency_spike_recent": {
      "type": "consolidated",
      "severity": "high",
      "detection_signals": [...],
      "cascade_analysis": {...}
    }
  },

  "current_metrics": {...},
  "slo_evaluation": {...},
  "fingerprinting": {...}
}
```

Error Handling

Metrics Unavailable

When VictoriaMetrics is unreachable:

```
{
  "alert_type": "metrics_unavailable",
  "service_name": "booking",
  "error": "Metrics collection failed",
  "failed_metrics": ["request_rate", "application_latency"],
  "skipped_reason": "critical_metrics_unavailable"
}
```

Detection is skipped to prevent false alerts from missing data.

Model Not Found

When no trained model exists:

```
{
  "alert_type": "error",
  "service_name": "new-service",
  "error_message": "No trained model found for time period"
}
```

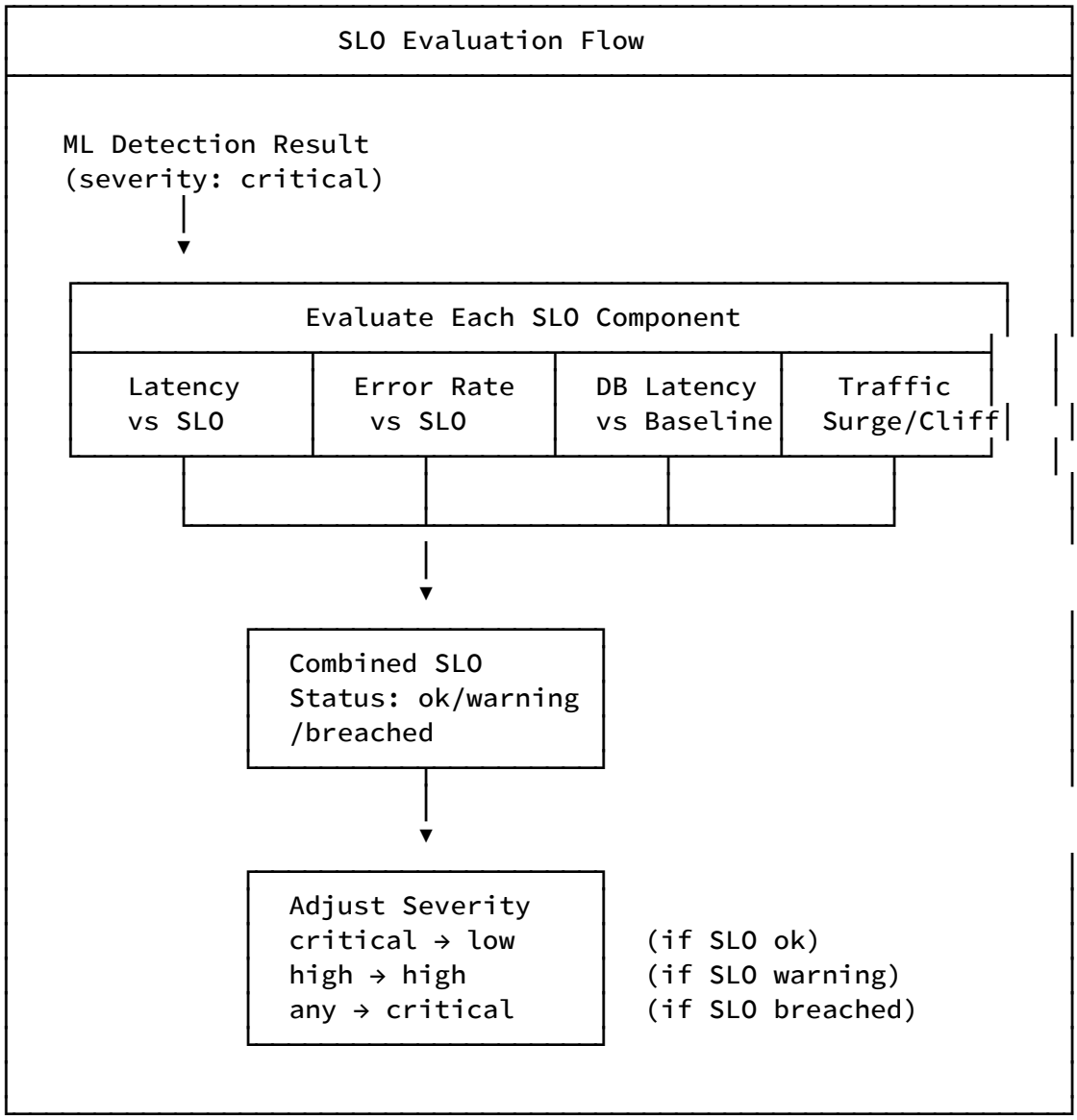
Run training to create models for new services.

SLO Evaluation Layer

The SLO (Service Level Objective) evaluation layer adjusts anomaly severity based on operational thresholds, answering: **“Does this anomaly matter operationally?”**

Purpose

ML detection answers: *“Is this behavior unusual?”* SLO evaluation answers: *“Does it impact users or breach operational limits?”*



SLO Components

Component	Threshold Type	What It Measures
Latency	Absolute (ms)	Response time vs SLO targets
Error Rate	Absolute (%)	Error percentage vs SLO targets

Component	Threshold Type	What It Measures
Database Latency	Ratio-based	DB latency vs training baseline
Request Rate	Ratio-based	Traffic surge or cliff detection

SLO Status

Status	Meaning	Severity Impact
`ok`	All metrics within acceptable limits	→ `low`
`warning`	Approaching SLO breach	stays `high`
`breached`	SLO threshold exceeded	→ `critical`

Key Insight

An anomaly can be **statistically significant** (detected by ML) but **operationally acceptable** (within SLO). The SLO layer filters noise by only escalating anomalies that actually impact service quality.

Example:

- ML detects latency at 280ms (unusual, p95 deviation)
- SLO threshold is 300ms (acceptable limit)
- Result: Severity adjusted to `low` (anomaly noted but no action needed)

Configuration

SLOs are configured per-service in `config.json`:

```
{
  "slos": {
    "enabled": true,
    "defaults": {
      "latency_acceptable_ms": 500,
      "latency_warning_ms": 800,
      "latency_critical_ms": 1000,
      "error_rate_acceptable": 0.005,
      "error_rate_warning": 0.01,
      "error_rate_critical": 0.02
    },
    "services": {
      "booking": {
        "latency_acceptable_ms": 300,
        "latency_critical_ms": 500
      }
    }
  }
}
```

Sections

- Latency Evaluation - Response time SLO checking
- Error Rate Evaluation - Error percentage SLO checking
- Database Latency - Ratio-based DB latency evaluation
- Request Rate - Traffic surge and cliff detection
- Severity Adjustment - How severity is adjusted based on SLO status

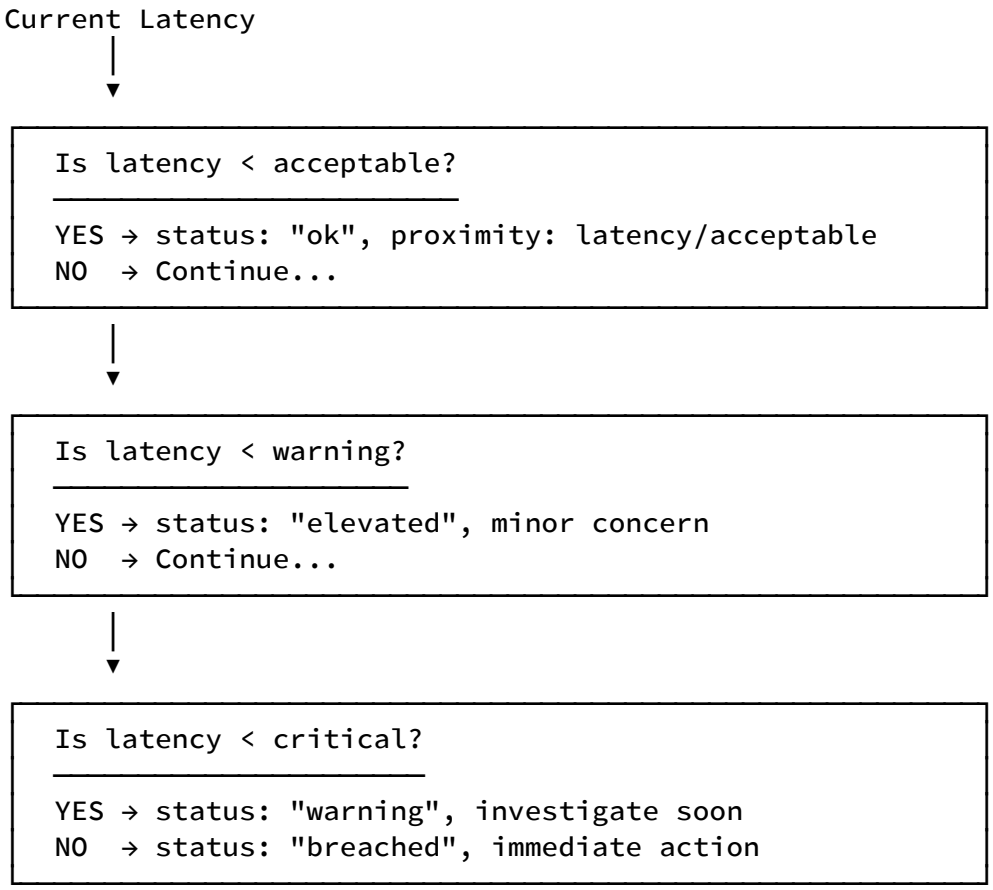
Latency Evaluation

Latency evaluation compares current response time against SLO thresholds.

Thresholds

Level	Default	Meaning
`acceptable`	500ms	Latency below this is fine
`warning`	800ms	Approaching SLO limit
`critical`	1000ms	SLO breach

Evaluation Logic



Proximity Score

The `proximity` value indicates how close to breach (0.0 - 1.0+):

Proximity	Interpretation
0.0 - 0.5	Comfortable margin
0.5 - 0.8	Getting close
0.8 - 1.0	Near threshold
> 1.0	Threshold exceeded

Output Example

```
{
  "latency_evaluation": {
    "status": "warning",
    "value": 750.0,
    "threshold_acceptable": 500,
    "threshold_warning": 800,
    "threshold_critical": 1000,
    "proximity": 0.94
  }
}
```

Per-Service Configuration

Critical services can have stricter thresholds:

```
{
  "slos": {
    "services": {
      "booking": {
        "latency_acceptable_ms": 300,
        "latency_warning_ms": 400,
        "latency_critical_ms": 500
      },
      "search": {
        "latency_acceptable_ms": 200,
        "latency_critical_ms": 400
      }
    }
  }
}
```

Busy Period Handling

During configured busy periods, thresholds are relaxed by `busy_period_factor`:

```
{
  "slos": {
    "defaults": {
      "busy_period_factor": 1.5
    },
    "busy_periods": [
      {"start": "2024-12-20T00:00:00", "end": "2025-01-05T23:59:59"}
    ]
  }
}
```

During busy periods:

- 500ms acceptable → 750ms acceptable
- 1000ms critical → 1500ms critical

Error Rate Evaluation

Error rate evaluation compares current error percentage against SLO thresholds.

Thresholds

Level	Default	Meaning
`acceptable`	0.5%	Error rate below this is fine
`warning`	1.0%	Approaching SLO limit
`critical`	2.0%	SLO breach

Error Rate Floor (Suppression)

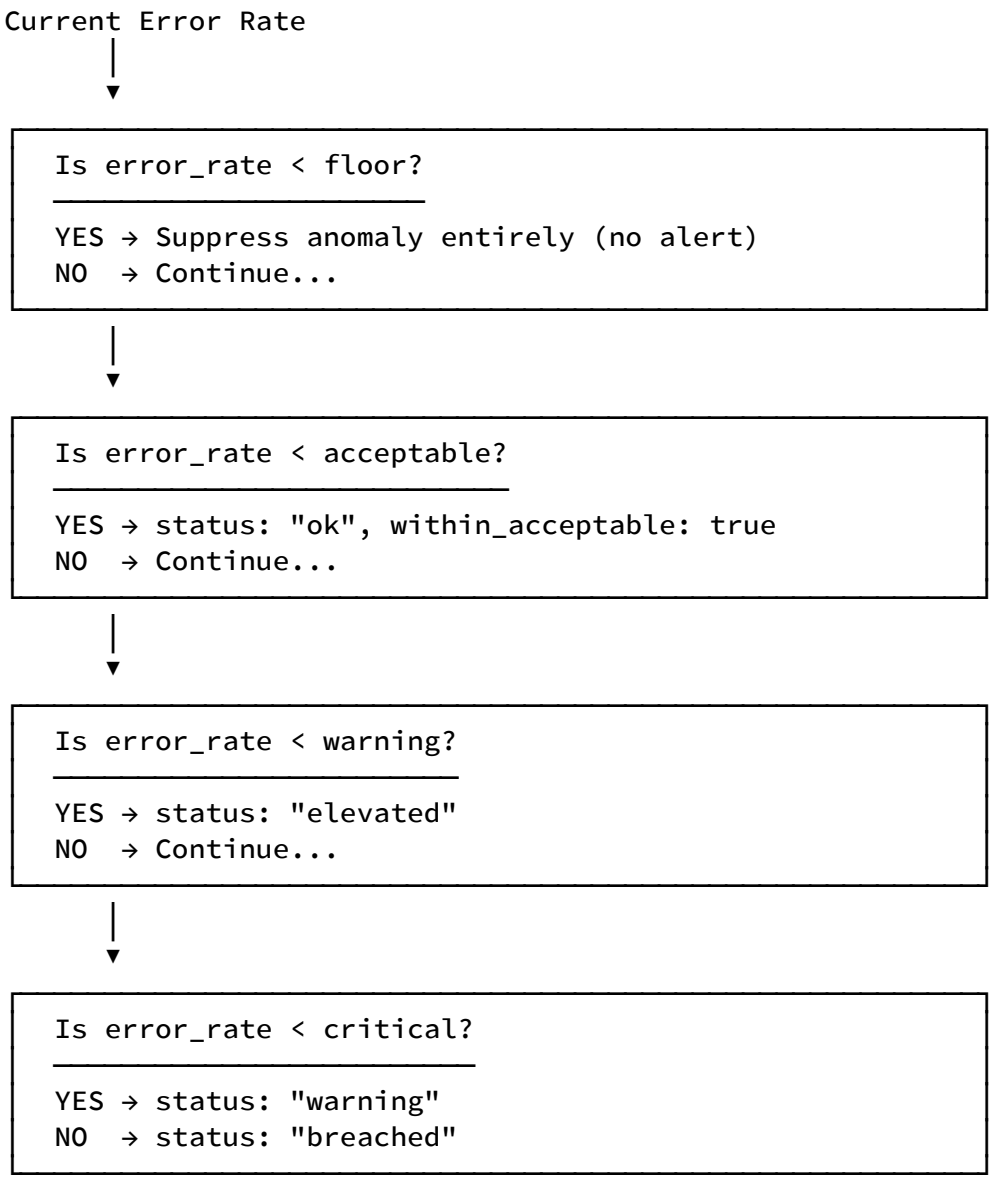
To prevent alert noise from tiny error rate deviations, an optional `error_rate_floor` can suppress anomalies when errors are operationally insignificant.

```
{
  "slos": {
    "services": {
      "booking": {
        "error_rate_acceptable": 0.002,
        "error_rate_floor": 0.002
      }
    }
  }
}
```

Error Rate	Floor (0.2%)	Result
0.01%	Below floor	**Suppressed** (no alert)

Error Rate	Floor (0.2%)	Result
0.15%	Below floor	**Suppressed** (no alert)
0.25%	Above floor	Alert fires

Evaluation Logic



Output Example


```
{
  "error_rate_evaluation": {
    "status": "ok",
    "value": 0.001,
    "value_percent": "0.10%",
    "threshold_acceptable": 0.005,
    "threshold_warning": 0.01,
    "threshold_critical": 0.02,
    "within_acceptable": true
  }
}
```


When Floor is Active


```
{
  "slo_context": {
    "current_value": 0.005,
    "current_value_percent": "0.50%",
    "acceptable_threshold": 0.002,
    "critical_threshold": 0.01,
    "suppression_threshold": 0.002,
    "within_acceptable": false
  }
}
```

Per-Service Configuration

```
{
  "slos": {
    "services": {
      "booking": {
        "error_rate_acceptable": 0.002,
        "error_rate_warning": 0.005,
        "error_rate_critical": 0.01,
        "error_rate_floor": 0.002
      },
      "admin-api": {
        "error_rate_acceptable": 0.01,
        "error_rate_critical": 0.05
      }
    }
  }
}
```

Exception Enrichment

When error SLO is breached (HIGH or CRITICAL severity), exception context is automatically queried and added to the alert:

```
{
  "exception_context": {
    "service_name": "search",
    "total_exception_rate": 0.35,
    "top_exceptions": [
      {"type": "R2D2Exception", "rate": 0.217, "percentage": 62.0},
      {"type": "UserInputException", "rate": 0.083, "percentage":
23.7}
    ]
  }
}
```

This helps identify which exception types are causing the error spike.

Database Latency Evaluation

Database latency uses a **hybrid approach**: noise floor filtering + ratio-based thresholds against training baseline.

Why Ratio-Based?

Unlike application latency with fixed SLO targets, database latency varies significantly:

- A fast service might have 5ms DB latency normally
- A reporting service might have 500ms DB latency normally

Ratio-based thresholds adapt to each service's baseline.

Noise Floor

Very low database latency values are filtered as operationally meaningless:

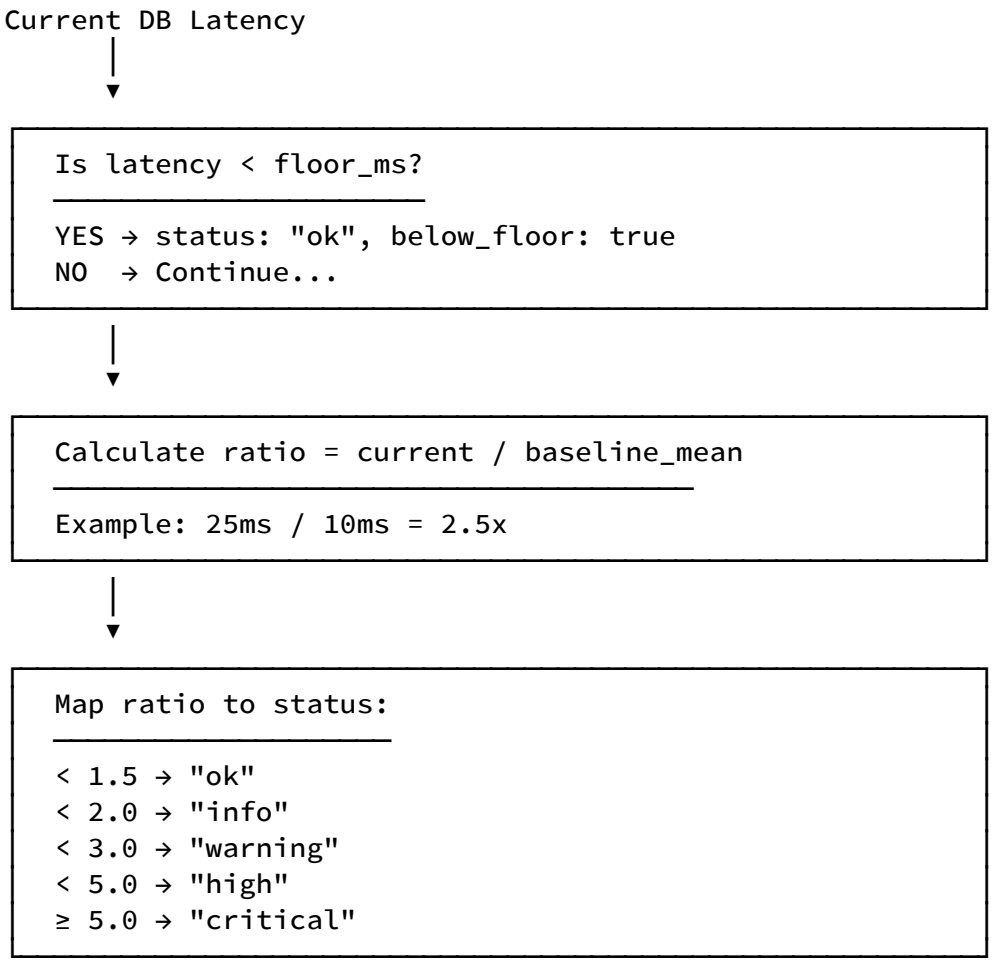
Latency	Floor (5ms)	Result
2ms	Below floor	Always `ok`
0.3ms → 0.4ms	Below floor	Ignored
10ms	Above floor	Evaluate ratio

This prevents alerts for sub-millisecond changes that the ML might flag as anomalous.

Ratio Thresholds

Status	Ratio	Meaning
`ok`	< 1.5x	Within normal variance
`info`	1.5x - 2x	Slightly elevated
`warning`	2x - 3x	Elevated, investigate
`high`	3x - 5x	Significantly elevated
`critical`	≥ 5x	SLO breach

Evaluation Logic



Output Examples

Below Floor


```
{
  "database_latency_evaluation": {
    "status": "ok",
    "value_ms": 2.0,
    "baseline_mean_ms": 1.0,
    "ratio": 0.0,
    "below_floor": true,
    "floor_ms": 5.0,
    "explanation": "Below noise floor (2.0ms < 5.0ms)"
  }
}
```


Elevated Ratio


```
{
  "database_latency_evaluation": {
    "status": "warning",
    "value_ms": 25.0,
    "baseline_mean_ms": 10.0,
    "ratio": 2.5,
    "below_floor": false,
    "floor_ms": 5.0,
    "thresholds": {
      "info": 1.5,
      "warning": 2.0,
      "high": 3.0,
      "critical": 5.0
    },
    "explanation": "DB latency elevated: 25.0ms is 2.5x baseline
(10.0ms)"
  }
}
```

Per-Service Configuration

Services with different DB performance characteristics can have custom thresholds:

```
{
  "slos": {
    "services": {
      "search": {
        "database_latency_floor_ms": 2.0,
        "database_latency_ratios": {
          "info": 1.3,
          "warning": 1.5,
          "high": 2.0,
          "critical": 3.0
        }
      },
      "reporting": {
        "database_latency_floor_ms": 50.0,
        "database_latency_ratios": {
          "info": 2.0,
          "warning": 3.0,
          "high": 5.0,
          "critical": 10.0
        }
      }
    }
  }
}
```

Relationship to Pattern Matching

Database latency evaluation complements pattern matching:

Pattern	Database Latency Evaluation
`database_degradation`	DB ratio 2-3x, compensating
`database_bottleneck`	DB ratio ≥3x, dominant latency

The SLO evaluation provides the **ratio context** that pattern matching uses for severity.

Request Rate Evaluation (Surge/Cliff)

Request rate evaluation detects significant traffic changes: **surges** (spikes) and **cliffs** (drops).

Key Insight

Traffic anomalies use **correlation-based severity**:

- A surge alone is often benign (marketing campaign, organic growth)
- A surge becomes problematic when causing SLO issues
- A cliff is inherently concerning (may indicate upstream failure)

Thresholds

Type	Threshold	Default
Surge	≥ 200% of baseline	2x normal traffic
Cliff	≤ 50% of baseline	Half normal traffic

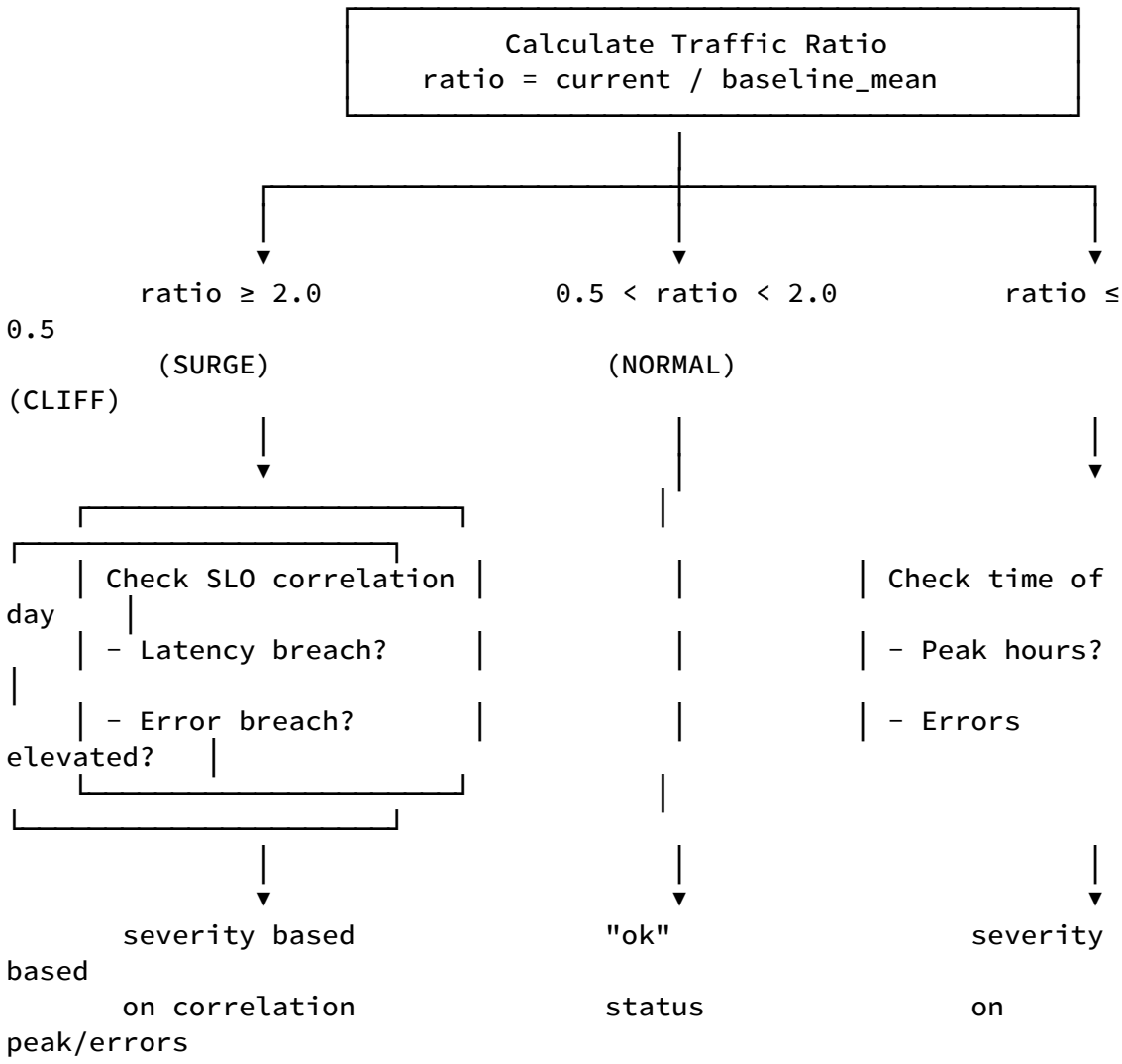
Surge Severity Logic

Condition	Severity	Rationale
Surge only	`informational`	Normal growth or campaign
Surge + latency SLO breach	`warning`	Capacity stress
Surge + error SLO breach	`high`	Active incident

Cliff Severity Logic

Condition	Severity	Rationale
Cliff off-peak	`warning`	May be expected
Cliff peak hours	`high`	Likely incident
Cliff + errors	`critical`	Confirmed incident

Evaluation Flow



Output Examples

Surge (Standalone)


```
{
  "request_rate_evaluation": {
    "status": "info",
    "type": "surge",
    "severity": "informational",
    "value_rps": 250.0,
    "baseline_mean_rps": 100.0,
    "ratio": 2.5,
    "threshold_percent": 200.0,
    "correlated_with_latency": false,
    "correlated_with_errors": false,
    "explanation": "Traffic surge (2.5x baseline) without SLO
impact. Normal growth or campaign traffic."
  }
}
```


Surge with Latency Impact


```
{
  "request_rate_evaluation": {
    "status": "warning",
    "type": "surge",
    "severity": "warning",
    "value_rps": 350.0,
    "baseline_mean_rps": 100.0,
    "ratio": 3.5,
    "correlated_with_latency": true,
    "correlated_with_errors": false,
    "explanation": "Traffic surge (3.5x baseline) correlating with
latency SLO breach - capacity issue."
  }
}
```

Cliff (Peak Hours)


```
{
  "request_rate_evaluation": {
    "status": "high",
    "type": "cliff",
    "severity": "high",
    "value_rps": 30.0,
    "baseline_mean_rps": 100.0,
    "ratio": 0.3,
    "threshold_percent": 50.0,
    "is_peak_hours": true,
    "correlated_with_errors": false,
    "explanation": "Traffic cliff (0.3x baseline) during peak hours
- investigate potential incident."
  }
}
```


Cliff with Errors


```
{
  "request_rate_evaluation": {
    "status": "critical",
    "type": "cliff",
    "severity": "critical",
    "value_rps": 15.0,
    "baseline_mean_rps": 100.0,
    "ratio": 0.15,
    "is_peak_hours": true,
    "correlated_with_errors": true,
    "explanation": "Traffic cliff (0.15x baseline) with errors -
likely upstream failure or routing issue."
  }
}
```

Configuration

```
{
  "slos": {
    "defaults": {
      "request_rate_surge_threshold": 2.0,
      "request_rate_cliff_threshold": 0.5
    },
    "services": {
      "booking": {
        "request_rate_evaluation": {
          "surge": {
            "threshold": 3.0
          },
          "cliff": {
            "standalone_severity": "high",
            "peak_hours_severity": "critical"
          }
        }
      }
    }
  }
}
```

Relationship to Pattern Matching

Pattern	Request Rate Evaluation
`traffic_surge_healthy`	Surge, no SLO correlation
`traffic_surge_degrading`	Surge + latency correlation
`traffic_surge_failing`	Surge + error correlation
`traffic_cliff`	Cliff detected

Severity Adjustment

The final step of SLO evaluation: adjusting ML-assigned severity based on operational impact.

Core Principle

SLO status determines final severity, not ML confidence.

SLO Status	Final Severity	Rationale
`ok`	`low`	Anomaly detected but operationally acceptable
`warning`	`high`	Approaching limits, should investigate
`breached`	`critical`	SLO exceeded, requires action

Adjustment Matrix

ML Severity	SLO Status	Final Severity	Example
critical	ok	**low**	280ms latency (unusual but < 300ms SLO)
critical	warning	high	750ms latency (approaching 800ms SLO)
critical	breached	critical	1200ms latency (> 1000ms SLO)
high	ok	**low**	Same principle
high	warning	high	No change needed
high	breached	critical	Escalated

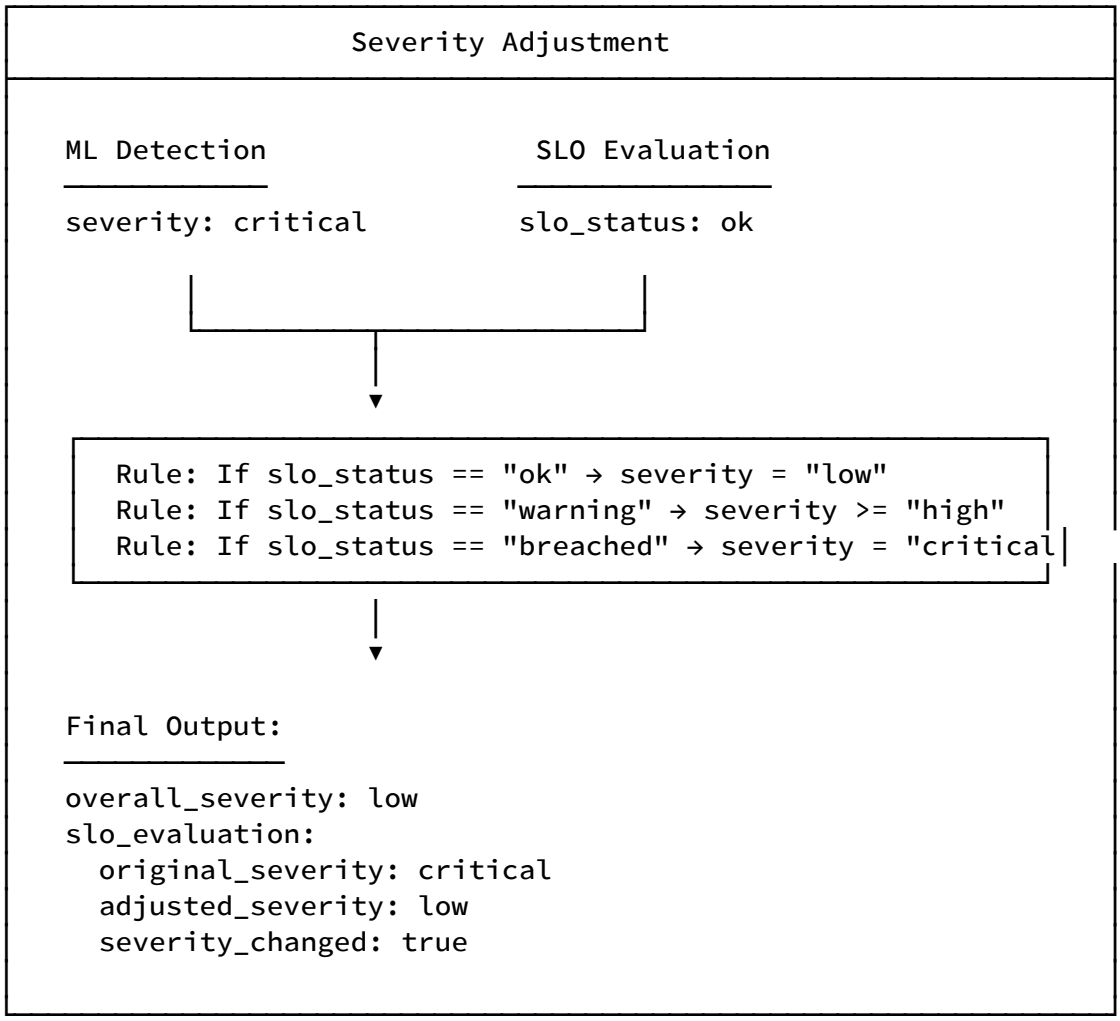
ML Severity	SLO Status	Final Severity	Example
medium	ok	**low**	Minor anomaly, within SLO
medium	warning	high	Escalated due to SLO proximity
low	breached	critical	Even low anomaly escalated if SLO breached

Key Behavior Change (v1.3.1)

Before v1.3.1: SLO `ok` adjusted to `medium` **After v1.3.1**: SLO `ok` adjusts to `low`

This ensures “operationally acceptable” consistently means “low priority.”

Adjustment Flow



Output Example

```
{
  "overall_severity": "low",
  "slo_evaluation": {
    "original_severity": "critical",
    "adjusted_severity": "low",
    "severity_changed": true,
    "slo_status": "ok",
    "slo_proximity": 0.56,
    "operational_impact": "informational",
    "explanation": "Severity adjusted from critical to low based on
SLO evaluation. Anomaly detected but metrics within acceptable SLO
thresholds (latency: 280ms < 300ms, errors: 0.10% < 0.50%)."
  }
}
```

Operational Impact Levels

Level	Meaning	Action
`none`	No anomaly or fully normal	No action
`informational`	Anomaly noted but acceptable	Log only
`actionable`	Approaching limits	Investigate
`critical`	SLO breached	Immediate action

Configuration Options

```
{
  "slos": {
    "enabled": true,
    "allow_downgrade_to_informational": true,
    "require_slo_breach_for_critical": true
  }
}
```

Option	Default	Effect
`enabled`	true	Enable/disable SLO evaluation
`allow_downgrade_to_informational`	true	Allow high/critical → low when SLO ok
`require_slo_breach_for_critical`	true	Only allow critical if SLO breached

root `overall_severity`

Important (v1.3.1): The root-level `overall_severity` field now correctly reflects the SLO-adjusted value.

```
{
  "overall_severity": "low",          // ← SLO-adjusted value
  "slo_evaluation": {
    "original_severity": "critical",  // ← ML-assigned value
    "adjusted_severity": "low"        // ← Same as overall_severity
  }
}
```

Previously, `overall_severity` could show `critical` while `adjusted_severity` showed `low`.

Alert Suppression

When SLO evaluation results in `low` severity:

- Alert is still generated (for logging/visibility)
- Alert may be filtered by downstream systems (e.g., skip PagerDuty)
- Dashboard shows alert with low priority indicator

To completely suppress alerts below a threshold, use downstream alert filtering rules.

Incident Lifecycle

The incident lifecycle manages anomaly state over time, reducing alert noise through confirmation and grace periods.

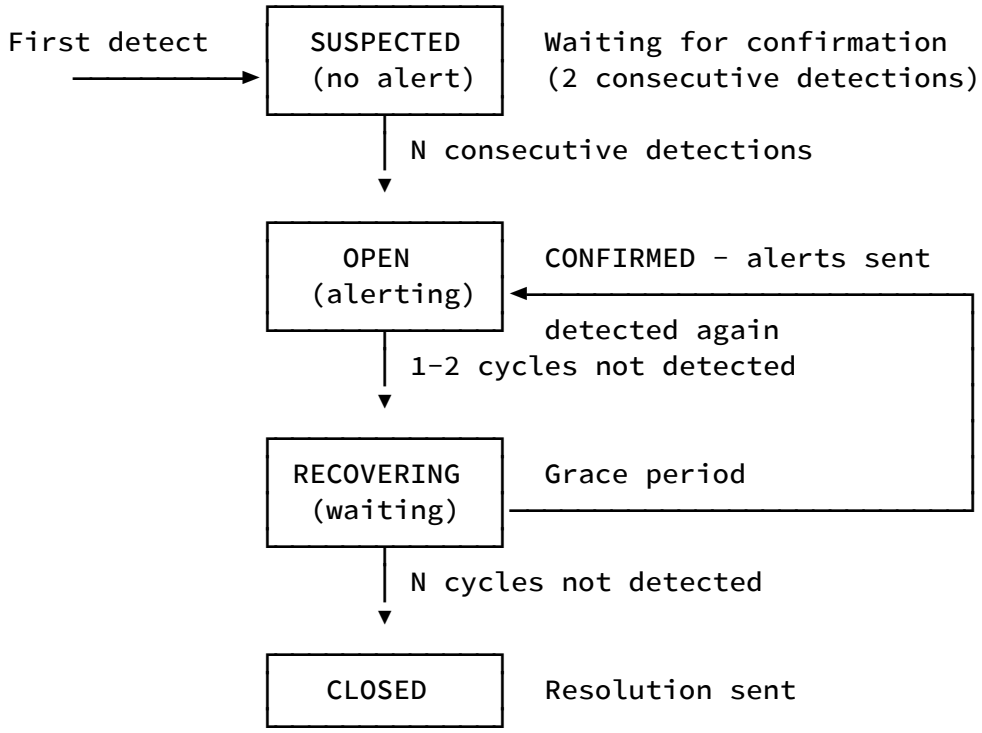
Core Concepts

Fingerprint vs Incident

Concept	Description	Example
Fingerprint ID	Pattern identifier (same pattern = same ID)	`anomaly_8d4a011b83ca`
Incident ID	Unique occurrence identifier	`incident_1dcbafc91480`

The same anomaly pattern can create multiple incidents over time. Each time the pattern reappears after resolution, a new incident is created.

State Machine



States

State	Alert Sent?	Description
`SUSPECTED`	No	First detection, waiting for confirmation
`OPEN`	Yes	Confirmed incident, alerts active
`RECOVERING`	No	Not detected recently, grace period
`CLOSED`	Resolution	Incident resolved

Cycle-Based Timing

With default configuration and 2-3 minute inference intervals:

Parameter	Default	Effect
`confirmation_cycles`	2	~4-6 min to confirm
`resolution_grace_cycles`	3	~6-9 min grace period
`incident_separation_minutes`	30	Gap > 30 min = new incident

Why Confirmation?

Problem: Single-point anomalies create alert noise.

Solution: Require 2+ consecutive detections before alerting.

```
Detection 1: Latency spike 450ms
  → Status: SUSPECTED (no alert)
  → Might be transient

Detection 2: Latency still 445ms
  → Status: OPEN (alert sent)
  → Confirmed - this is real

Detection 3: Latency still 440ms
  → Status: OPEN (continue)
  → Ongoing incident
```

Why Grace Period?

Problem: Flapping alerts when anomaly briefly clears.

Solution: Wait 3 cycles before resolving.

```
Detection 5: Latency normal 120ms
  → Status: RECOVERING (no resolution yet)
  → Might return

Detection 6: Latency still normal
  → Status: RECOVERING (grace period)
  → Waiting...

Detection 7: Latency still normal
  → Status: CLOSED (resolution sent)
  → Confirmed resolution
```

Staleness Check

If an anomaly reappears after a long gap (> 30 min), it's treated as a new incident:

```
10:00 - OPEN incident detected
10:30 - Last detection
...
11:15 - Same pattern detected (45 min gap)
  → Old incident auto-closed (reason: "auto_stale")
  → New SUSPECTED incident created
```

This prevents "zombie incidents" that continue indefinitely.

Confirmed-Only Alerts (v1.3.2)

Important: Only confirmed anomalies are sent to the web API.

State	Sent to Web API?
SUSPECTED	No
OPEN	Yes
RECOVERING	Yes
CLOSED	Resolution only

This prevents orphaned incidents when SUSPECTED anomalies expire without confirmation.

Sections

- State Machine - Detailed state transitions
- Confirmation Logic - How confirmation works
- Fingerprinting - How incidents are identified

State Machine

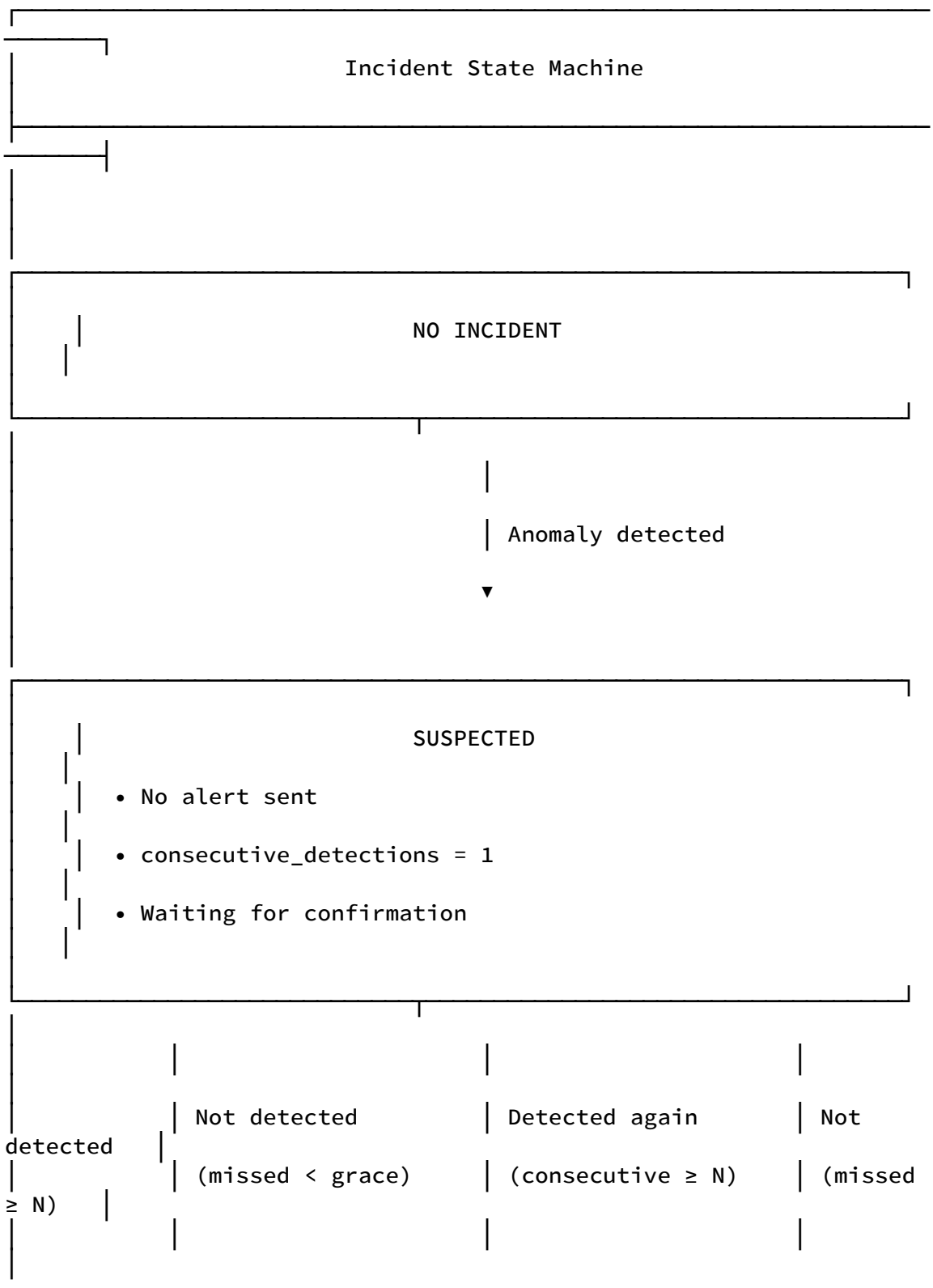
Detailed state transitions for incident lifecycle management.

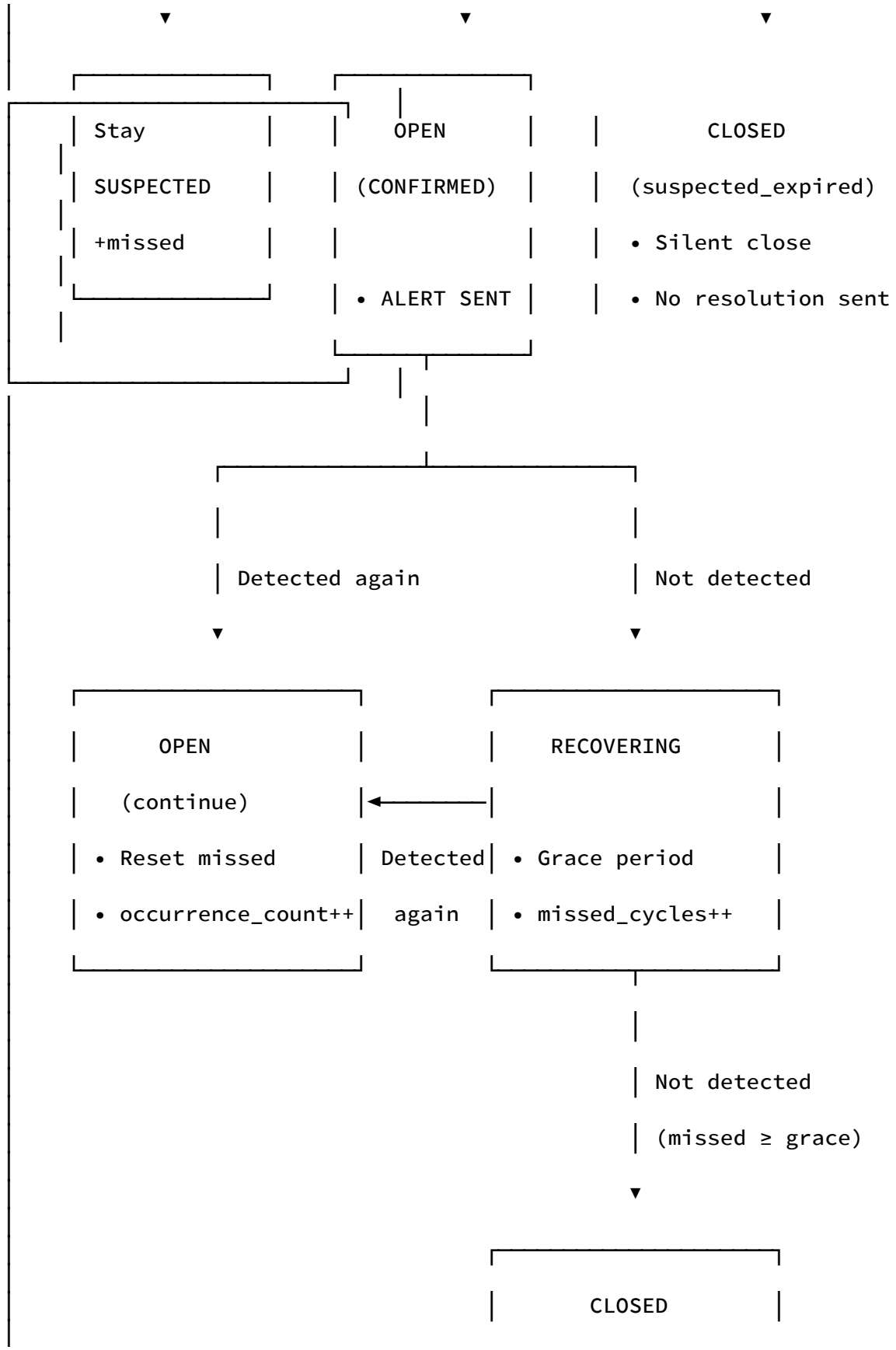
Transition Rules

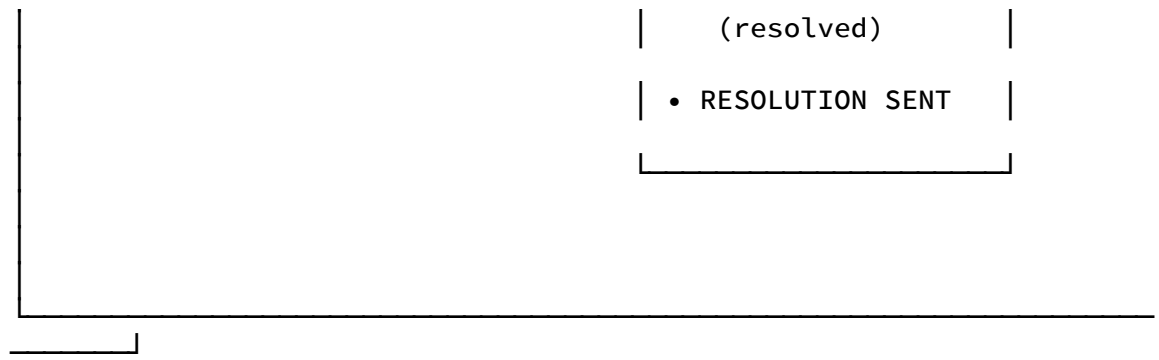
Current State	Event	Condition	New State	Actio
(none)	Detected	-	SUSPECTED	Create incident
SUSPECTED	Detected	consecutive < N	SUSPECTED	Increme counter
SUSPECTED	Detected	consecutive ≥ N	OPEN	**Send al**
SUSPECTED	Not detected	missed < N	SUSPECTED	Increme missed
SUSPECTED	Not detected	missed ≥ N	CLOSED	Silent close
OPEN	Detected	-	OPEN	Continu reset missed
OPEN	Not detected	-	RECOVERING	Start gra period
RECOVERING	Detected	-	OPEN	Resume incident
RECOVERING	Not detected	missed < N	RECOVERING	Continu grace
RECOVERING	Not detected	missed ≥ N	CLOSED	**Send resolut**
Any	Detected	gap > separation	SUSPECTED	Close stale,

Current State	Event	Condition	New State	Actio
				create n

State Transition Diagram




Stay
SUSPECTED
+missed
OPEN
(CONFIRMED)
• ALERT SENT
CLOSED
(suspected_expired)
• Silent close
• No resolution sent
Detected again
Not detected
OPEN
(continue)
• Reset missed
• occurrence_count++
Detected
again
RECOVERING
• Grace period
• missed_cycles++
Not detected
(missed ≥ grace)
CLOSED



Payload Fields Per State

SUSPECTED


```
{
  "status": "SUSPECTED",
  "incident_action": "CREATE",
  "consecutive_detections": 1,
  "missed_cycles": 0,
  "confirmation_pending": true,
  "cycles_to_confirm": 1,
  "is_confirmed": false,
  "newly_confirmed": false
}
```

OPEN (Newly Confirmed)

```
{
  "status": "OPEN",
  "previous_status": "SUSPECTED",
  "incident_action": "CONTINUE",
  "consecutive_detections": 2,
  "missed_cycles": 0,
  "confirmation_pending": false,
  "is_confirmed": true,
  "newly_confirmed": true
}
```

OPEN (Continuing)

```
{
  "status": "OPEN",
  "incident_action": "CONTINUE",
  "consecutive_detections": 5,
  "occurrence_count": 5,
  "incident_duration_minutes": 15
}
```

RECOVERING

```
{
  "status": "RECOVERING",
  "missed_cycles": 1,
  "incident_action": "CONTINUE"
}
```

Note: RECOVERING is an internal state. The fingerprinting summary shows `status_summary.recovering: 1`.

CLOSED (Resolution)

```
{
  "fingerprint_id": "anomaly_061598e9ca91",
  "incident_id": "incident_abc123def456",
  "incident_action": "CLOSE",
  "fingerprint_action": "RESOLVE",
  "final_severity": "high",
  "resolved_at": "2025-12-17T14:30:00",
  "total_occurrences": 5,
  "incident_duration_minutes": 45,
  "resolution_reason": "resolved"
}
```

Resolution Reasons

Reason	Description	Resolution Sent?
`resolved`	Normal resolution after grace period	Yes
`suspected_expired`	SUSPECTED state expired without confirmation	No
`auto_stale`	Time gap exceeded separation threshold	Yes

Configuration

```
{
  "fingerprinting": {
    "confirmation_cycles": 2,
    "resolution_grace_cycles": 3,
    "incident_separation_minutes": 30
  }
}
```

Parameter	Effect
`confirmation_cycles`	Consecutive detections needed to confirm
`resolution_grace_cycles`	Non-detection cycles before closing
`incident_separation_minutes`	Gap that triggers new incident

Confirmation Logic

Confirmation prevents alert noise by requiring multiple consecutive detections before alerting.

Why Confirmation?

Without confirmation:

```
10:00 - Latency spike 450ms → ALERT
10:03 - Latency normal 120ms → RESOLVE
10:06 - Latency spike 455ms → ALERT
10:09 - Latency normal 118ms → RESOLVE
...
```

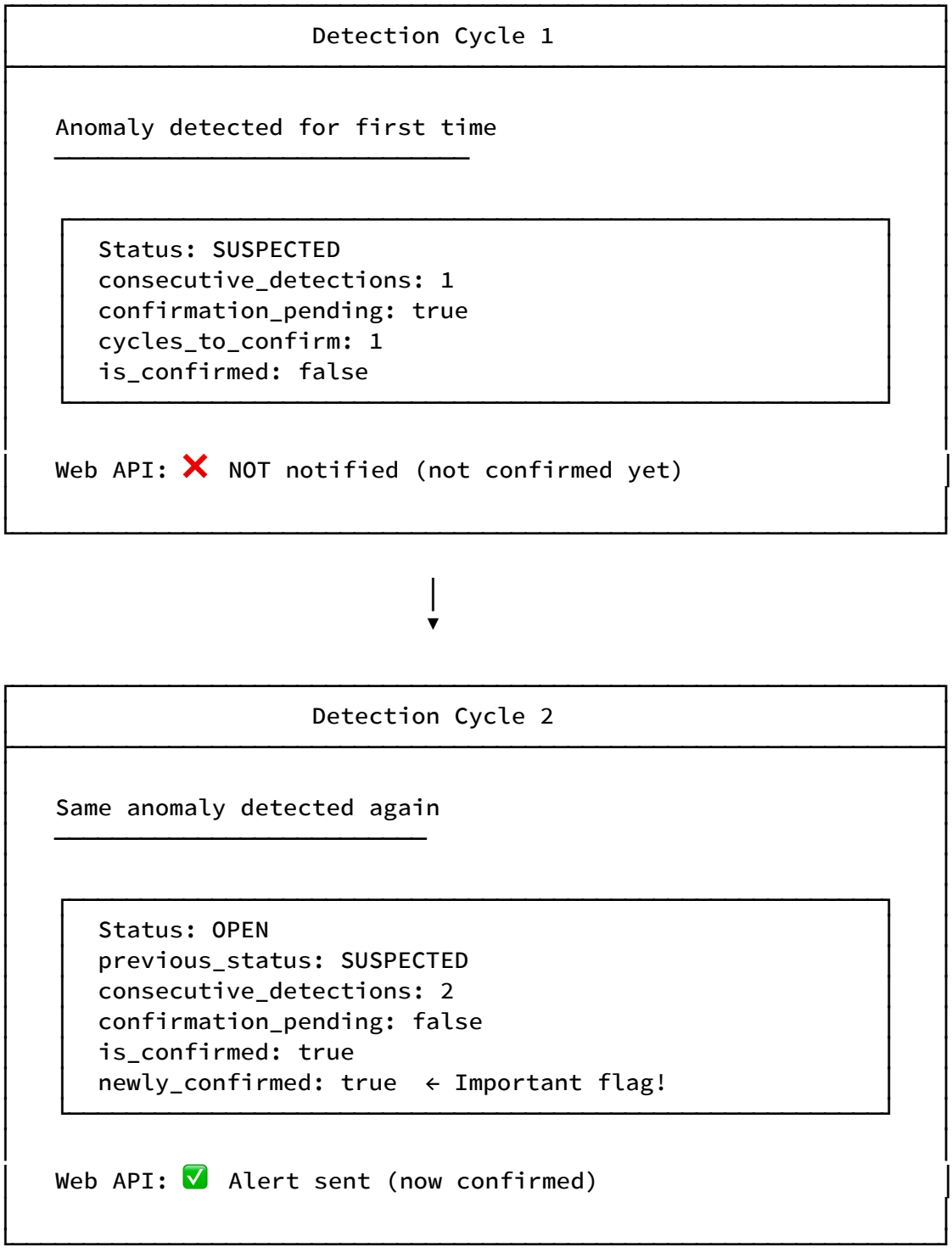
Result: Alert fatigue from flapping.

With confirmation:

```
10:00 - Latency spike 450ms → SUSPECTED (no alert)
10:03 - Latency normal 120ms → SUSPECTED expires (no alert)
10:06 - Latency spike 455ms → SUSPECTED (no alert)
10:09 - Latency spike 448ms → OPEN (alert sent!)
10:12 - Latency spike 440ms → CONTINUE
...
```

Result: Only sustained issues generate alerts.

Confirmation Flow



Key Fields

Field	Description
`consecutive_detections`	How many cycles in a row this was detected
`confirmation_pending`	True if still in SUSPECTED state
`cycles_to_confirm`	Remaining cycles needed for confirmation
`is_confirmed`	True if status is OPEN
`newly_confirmed`	True only on the cycle where SUSPECTED → OPEN

Fingerprinting Summary

The top-level `fingerprinting` object summarizes confirmation status:


```
{
  "fingerprinting": {
    "overall_action": "CONFIRMED",
    "status_summary": {
      "suspected": 0,
      "confirmed": 1,
      "recovering": 0
    },
    "newly_confirmed_incidents": [
      {
        "fingerprint_id": "anomaly_8d4a011b83ca",
        "incident_id": "incident_1dcbafc91480",
        "anomaly_name": "latency_spike_recent"
      }
    ]
  }
}
```

Overall Action Values

Action	Meaning
`CREATE`	New SUSPECTED incident(s) created
`CONFIRMED`	Incident(s) transitioned from SUSPECTED → OPEN
`UPDATE`	Existing OPEN incident(s) continued
`RESOLVE`	Incident(s) closed
`MIXED`	Multiple different actions in one cycle
`NO_CHANGE`	No significant changes

Web API Integration

Only confirmed anomalies are sent to the web API:

```
# Filter to only confirmed anomalies
confirmed_anomalies = {
    name: anomaly for name, anomaly in anomalies.items()
    if anomaly.get('is_confirmed', False) or
       anomaly.get('status') in ('OPEN', 'RECOVERING')
}

if confirmed_anomalies:
    # Send to web API
    send_alert(confirmed_anomalies)
```

SUSPECTED Expiration

If an anomaly is never confirmed (not detected for `resolution_grace_cycles`):

```
10:00 - SUSPECTED created
10:03 - Not detected (missed: 1)
10:06 - Not detected (missed: 2)
10:09 - Not detected (missed: 3)
  → CLOSED with reason: "suspected_expired"
  → No alert was ever sent
  → No resolution is sent
```

This prevents orphaned incidents in the web API.

Tuning Confirmation

```
{
  "fingerprinting": {
    "confirmation_cycles": 3
  }
}
```

Value	Trade-off
1	Immediate alerts (no confirmation)
2	Default: ~4-6 min confirmation
3	Stricter: ~6-9 min confirmation
4+	Very strict: may miss real issues

Higher values reduce false positives but increase detection latency.

Fingerprinting

Fingerprinting assigns stable identifiers to anomaly patterns, enabling tracking across detection cycles.

ID Types

Fingerprint ID

A **deterministic hash** based on anomaly content:

```
content = f"{service_name}_{model_name}_{anomaly_name}"
fingerprint_id = f"anomaly_{sha256(content)[:12]}"
```

Property	Description
Deterministic	Same pattern always gets same ID
Content-based	Based on service + model + anomaly type
Stable	Doesn't change across time

Example: `anomaly_8d4a011b83ca`

Incident ID

A **unique occurrence** identifier:

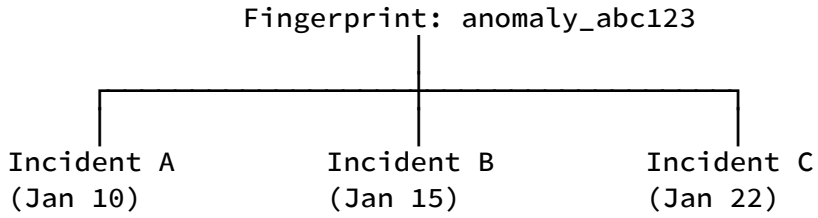
```
incident_id = f"incident_{uuid4().hex[:16]}"
```

Property	Description
Unique	Each occurrence gets new ID
UUID-based	Random generation

Property	Description
Transient	New ID when pattern reappears

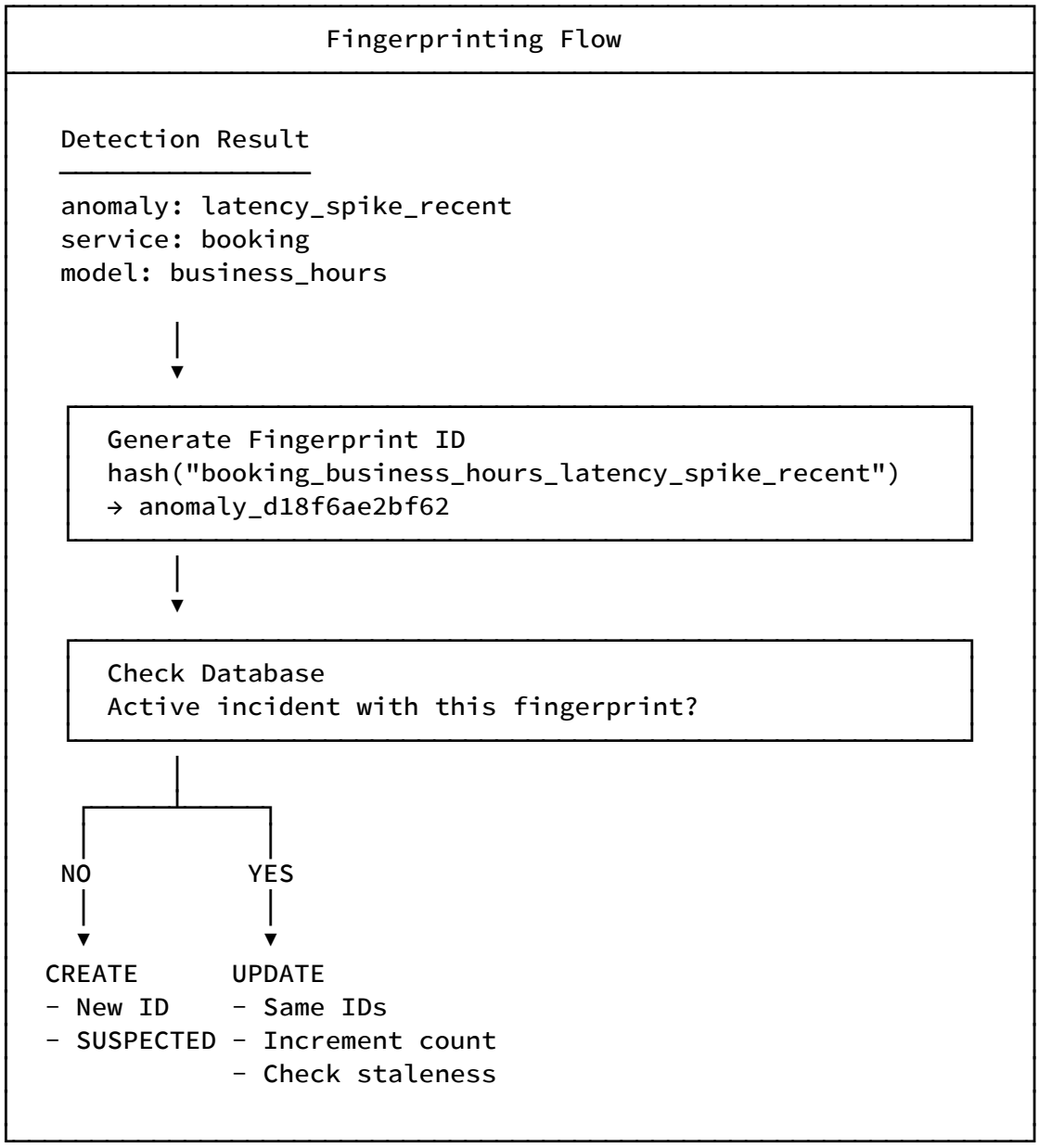
Example: `incident_1dcbafc91480`

Relationship



Same pattern can create multiple incidents over time

How Fingerprinting Works



Database Schema

```
CREATE TABLE anomaly_incidents (
    fingerprint_id TEXT NOT NULL,
    incident_id TEXT PRIMARY KEY,
    service_name TEXT NOT NULL,
    anomaly_name TEXT NOT NULL,
    status TEXT NOT NULL,  -- SUSPECTED, OPEN, RECOVERING, CLOSED
    severity TEXT NOT NULL,
    first_seen TIMESTAMP NOT NULL,
    last_updated TIMESTAMP NOT NULL,
    resolved_at TIMESTAMP NULL,
    occurrence_count INTEGER NOT NULL,
    consecutive_detections INTEGER NOT NULL,
    missed_cycles INTEGER NOT NULL,
    ...
);
```

Staleness Check

If the time gap between detections exceeds `incident_separation_minutes`:

```
Last detected: 10:00
Current time:  11:15 (75 min gap)
Threshold:     30 min

Result:
  → Old incident auto-closed (auto_stale)
  → New SUSPECTED incident created
```

This prevents an incident from continuing indefinitely across unrelated events.

Per-Anomaly Fields

Each anomaly in the output includes fingerprinting metadata:

```
{
  "latency_spike_recent": {
    "fingerprint_id": "anomaly_d18f6ae2bf62",
    "fingerprint_action": "UPDATE",
    "incident_id": "incident_31e9e23d4b2b",
    "incident_action": "CONTINUE",
    "status": "OPEN",
    "previous_status": "OPEN",
    "incident_duration_minutes": 15,
    "first_seen": "2025-12-17T13:45:00",
    "last_updated": "2025-12-17T14:00:00",
    "occurrence_count": 5,
    "consecutive_detections": 5,
    "is_confirmed": true
  }
}
```


Action Types

Fingerprint Actions

Action	Description
`CREATE`	New pattern, new fingerprint entry
`UPDATE`	Existing pattern, updating state
`RESOLVE`	Pattern resolved, closing entry

Incident Actions

Action	Description
`CREATE`	New incident created
`CONTINUE`	Incident continuing
`CLOSE`	Incident closed

Resolution Payload

When an incident resolves:

```
{
  "resolved_incidents": [
    {
      "fingerprint_id": "anomaly_d18f6ae2bf62",
      "incident_id": "incident_31e9e23d4b2b",
      "anomaly_name": "latency_spike_recent",
      "fingerprint_action": "RESOLVE",
      "incident_action": "CLOSE",
      "final_severity": "high",
      "resolved_at": "2025-12-17T14:30:00",
      "total_occurrences": 8,
      "incident_duration_minutes": 45,
      "first_seen": "2025-12-17T13:45:00",
      "resolution_reason": "resolved"
    }
  ]
}
```

Database Cleanup

Closed incidents are cleaned up after a configurable period:

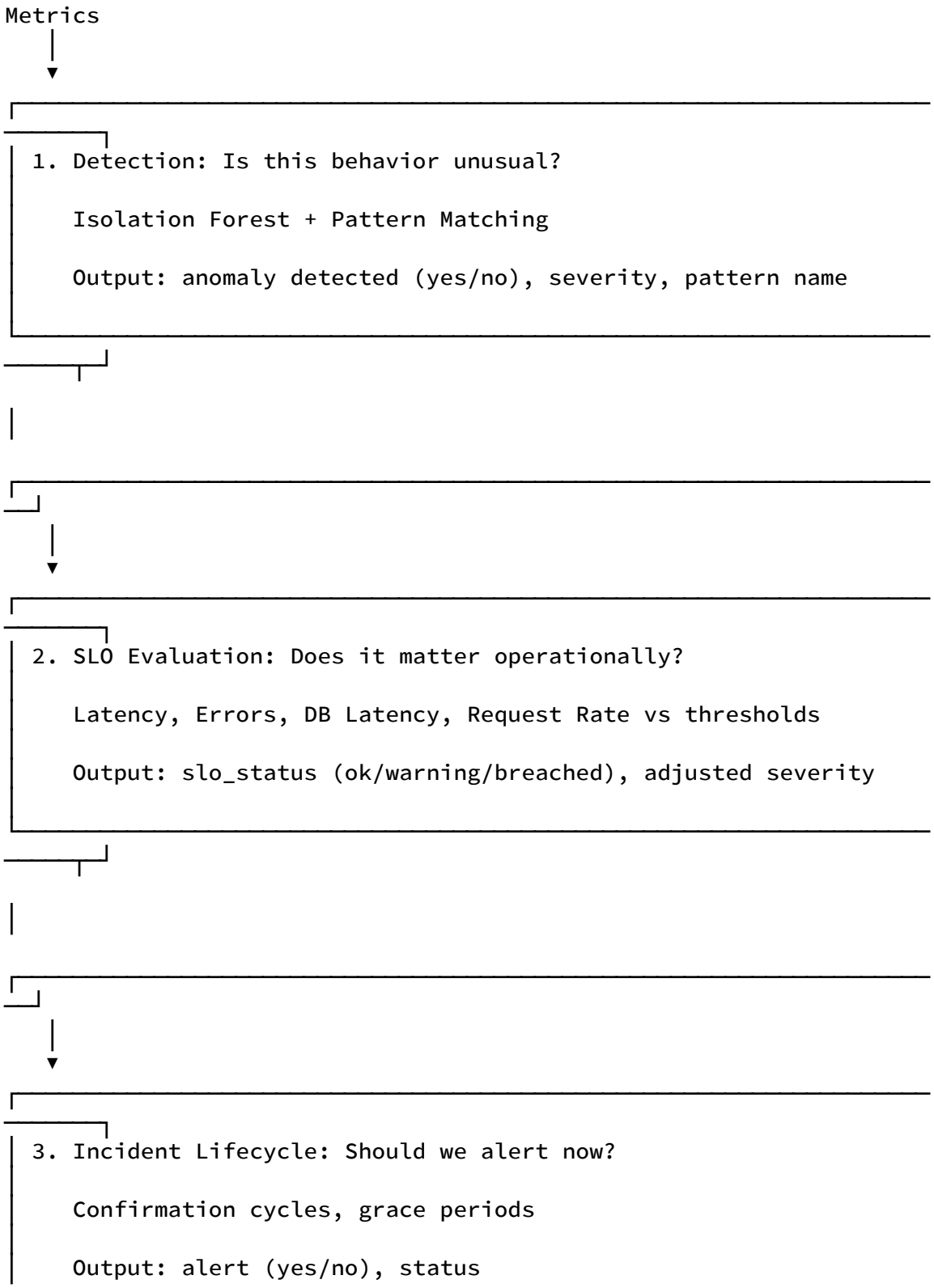
```
{
  "fingerprinting": {
    "cleanup_max_age_hours": 72
  }
}
```

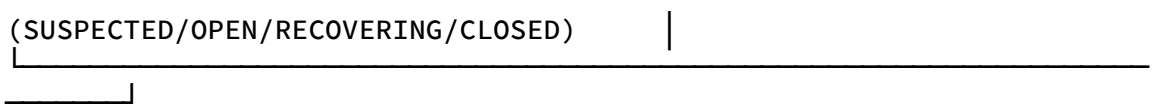
Incidents older than 72 hours (closed status) are removed to prevent unbounded database growth.

Decision Matrix

Quick reference for how different conditions map to alert decisions.

End-to-End Decision Flow





Detection Decision Matrix

Traffic	Latency	Errors	Pattern	Se
High	Normal	Normal	`traffic_surge_healthy`	lo
High	High	Normal	`traffic_surge_degrading`	hi
High	High	High	`traffic_surge_failing`	cr
Very Low	Any	Any	`traffic_cliff`	cr
Normal	High	Normal	`latency_spike_recent`	hi
Normal	High	Normal (deps healthy)	`internal_latency_issue`	hi
Normal	Normal	High	`error_rate_elevated`	hi
Normal	Normal	Very High	`error_rate_critical`	cr
Normal	Low	High	`fast_failure`	cr
Normal	Very Low	Very High	`fast_rejection`	cr
Normal	High (DB dominant)	Normal	`database_bottleneck`	hi
Normal	Normal (DB high)	Normal	`database_degradation`	m

SLO Severity Adjustment Matrix

ML Severity	Latency SLO	Error SLO	DB SLO	Final Severity
critical	ok	ok	ok	**low**
critical	ok	ok	warning	**low**
critical	warning	ok	ok	high
critical	ok	warning	ok	high
critical	breached	ok	ok	**critical**
critical	ok	breached	ok	**critical**
high	ok	ok	ok	**low**
high	warning	ok	ok	high
medium	ok	ok	ok	**low**
any	breached	breached	any	**critical**

Key Rule: SLO status `ok` → Final severity `low` (regardless of ML severity)

Incident State Decision Matrix

Current State	Anomaly Detected?	Consecutive	Action
None	Yes	1	CREATE (SUSPECTED)
SUSPECTED	Yes	< threshold	Stay SUSPECTED
SUSPECTED	Yes	≥ threshold	→ OPEN (alert)
SUSPECTED	No	< grace	Stay SUSPECTED
SUSPECTED	No	≥ grace	→ CLOSED (silent)
OPEN	Yes	any	Continue OPEN

Current State	Anomaly Detected?	Consecutive	Action
OPEN	No	1	→ RECOVERING
RECOVERING	Yes	any	→ OPEN (resume)
RECOVERING	No	< grace	Stay RECOVERING
RECOVERING	No	≥ grace	→ CLOSED (resolve)

Alert Decision Summary

Condition	Alert Sent?	Reason
First detection	No	SUSPECTED, waiting for confirmation
Second consecutive	Yes	Confirmed (OPEN)
Continuing OPEN	No	Already alerting
First non-detection	No	Grace period (RECOVERING)
Resolved	Resolution	CLOSED after grace period
Stale gap (>30 min)	New alert	Old closed, new SUSPECTED

Complete Example Scenarios

Scenario 1: Transient Spike (No Alert)

```
10:00 - Latency 450ms detected
        → ML: latency_spike_recent (high)
        → SLO: 450ms < 500ms acceptable → status: ok
        → Adjusted severity: low
        → Status: SUSPECTED (first detection)
        → Alert: ❌ NO

10:03 - Latency 120ms normal
        → No anomaly detected
        → Status: SUSPECTED (missed: 1)

10:06 - Latency 115ms normal
        → No anomaly detected
        → Status: SUSPECTED (missed: 2)

10:09 - Latency 118ms normal
        → Status: CLOSED (suspected_expired)
        → Alert: ❌ NO (never confirmed)
```

Scenario 2: Real Incident (Alert)

```
10:00 - Latency 850ms detected
        → ML: latency_spike_recent (critical)
        → SLO: 850ms > 800ms warning → status: warning
        → Adjusted severity: high
        → Status: SUSPECTED
        → Alert: ❌ NO (not confirmed)

10:03 - Latency 900ms detected
        → Status: OPEN (confirmed!)
        → Alert: ✅ YES

10:06 - Latency 920ms detected
        → Status: OPEN (continue)
        → Alert: Already sent

...

10:30 - Latency 200ms normal
        → Status: RECOVERING (grace: 1)

10:33 - Latency 180ms normal
        → Status: RECOVERING (grace: 2)

10:36 - Latency 175ms normal
        → Status: CLOSED (resolved)
        → Resolution: ✅ YES
```

Scenario 3: SLO Suppression (Low Priority)

```
10:00 - Latency 280ms detected
        → ML: latency_spike_recent (high)
        → SLO: 280ms < 300ms acceptable → status: ok
        → Adjusted severity: low (!!!)
        → Status: SUSPECTED
        → Alert: ❌ NO

10:03 - Latency 285ms detected
        → Status: OPEN (confirmed)
        → Severity: low
        → Alert: ✅ YES (low priority)
```

Quick Reference: When to Alert

Must be True	Description
Anomaly detected	ML flagged unusual behavior
Pattern matched	Known operational scenario
2+ consecutive	Confirmed, not transient
SLO evaluated	Operational impact assessed

Final Severity	Action
critical	PagerDuty, immediate action
high	Alert, investigate soon
low	Log only, informational
none	No action

Configuration Reference

Complete configuration reference for `config.json`.

Configuration File Location

The system searches for configuration in this order:

1. `CONFIG_FILE` environment variable
2. `./config.json` (current directory)
3. `./config/config.json`
4. `~/.smartbox/config.json`
5. `/etc/smartbox/config.json`

Core Sections

VictoriaMetrics

```
{
  "victoria_metrics": {
    "endpoint": "https://otel-metrics.production.smartbox.com",
    "timeout_seconds": 10,
    "max_retries": 3,
    "pool_connections": 20,
    "circuit_breaker_threshold": 5,
    "circuit_breaker_timeout_seconds": 300
  }
}
```

Field	Default	Description
`endpoint`	required	VictoriaMetrics server URL

Field	Default	Description
`timeout_seconds`	10	Request timeout
`max_retries`	3	Retry attempts
`circuit_breaker_threshold`	5	Failures before circuit opens

SLO Configuration

```
{
  "slos": {
    "enabled": true,
    "allow_downgrade_to_informational": true,
    "require_slo_breach_for_critical": true,
    "defaults": {
      "latency_acceptable_ms": 500,
      "latency_warning_ms": 800,
      "latency_critical_ms": 1000,
      "error_rate_acceptable": 0.005,
      "error_rate_warning": 0.01,
      "error_rate_critical": 0.02,
      "error_rate_floor": 0,
      "database_latency_floor_ms": 5.0,
      "database_latency_ratios": {
        "info": 1.5,
        "warning": 2.0,
        "high": 3.0,
        "critical": 5.0
      },
      "busy_period_factor": 1.5
    },
    "services": {
      "booking": {
        "latency_acceptable_ms": 300,
        "latency_critical_ms": 500,
        "error_rate_acceptable": 0.002,
        "error_rate_floor": 0.002
      }
    },
    "busy_periods": [
      {
        "start": "2024-12-20T00:00:00",
        "end": "2025-01-05T23:59:59"
      }
    ]
  }
}
```

Field	Default	Description
`enabled`	true	Enable SLO evaluation

Field	Default	Description
`allow_downgrade_to_informational`	true	Allow severity reduction when SLO ok
`require_slo_breach_for_critical`	true	Only critical if SLO breached
`latency_acceptable_ms`	500	Latency SLO acceptable threshold
`latency_critical_ms`	1000	Latency SLO critical threshold
`error_rate_acceptable`	0.005	Error rate acceptable (0.5%)
`error_rate_floor`	0	Error rate suppression floor
`database_latency_floor_ms`	5.0	DB latency noise floor

Fingerprinting

```
{
  "fingerprinting": {
    "db_path": "./anomaly_state.db",
    "cleanup_max_age_hours": 72,
    "incident_separation_minutes": 30,
    "confirmation_cycles": 2,
    "resolution_grace_cycles": 3
  }
}
```

Field	Default	Description
`db_path`	./anomaly_state.db	SQLite database path

Field	Default	Description
`cleanup_max_age_hours`	72	Hours before cleanup
`incident_separation_minutes`	30	Gap triggering new incident
`confirmation_cycles`	2	Cycles to confirm (send alert)
`resolution_grace_cycles`	3	Cycles before closing

Services

```
{
  "services": {
    "critical": ["booking", "search", "mobile-api"],
    "standard": ["friday", "gambit", "titan"],
    "micro": ["fa5"],
    "admin": ["m2-fr-adm", "m2-it-adm"],
    "core": ["m2-bb", "m2-fr"]
  }
}
```

Service categories affect default contamination rates:

Category	Contamination	Description
critical	0.03	Revenue-critical services
standard	0.05	Normal production
core	0.04	Platform infrastructure
admin	0.06	Administrative tools
micro	0.08	Low-traffic services

Dependencies

```
{
  "dependencies": {
    "graph": {
      "booking": ["search", "vms", "r2d2"],
      "vms": ["titan"],
      "search": ["catalog", "r2d2"]
    },
    "cascade_detection": {
      "enabled": true,
      "max_depth": 5
    }
  }
}
```

Model Configuration

```
{
  "model": {
    "models_directory": "./smartbox_models/",
    "min_training_samples": 500,
    "min_multivariate_samples": 1000,
    "default_contamination": 0.05,
    "default_n_estimators": 200,
    "contamination_by_service": {
      "booking": 0.02,
      "search": 0.04
    }
  }
}
```

Time Periods

```
{
  "time_periods": {
    "business_hours": {"start": 8, "end": 18, "weekdays_only":
true},
    "evening_hours": {"start": 18, "end": 22, "weekdays_only":
true},
    "night_hours": {"start": 22, "end": 6, "weekdays_only": true},
    "weekend_day": {"start": 8, "end": 22, "weekends_only": true},
    "weekend_night": {"start": 22, "end": 8, "weekends_only": true}
  }
}
```

Inference

```
{
  "inference": {
    "alerts_directory": "./alerts/",
    "max_workers": 3,
    "inter_service_delay_seconds": 0.2,
    "check_drift": false
  }
}
```

Environment Variables

Variable	Config Path	Description
`CONFIG_FILE`	-	Path to config file
`VM_ENDPOINT`	victoria_metrics.endpoint	VictoriaMetrics URL
`FINGERPRINT_DB`	fingerprinting.db_path	SQLite path
`OBSERVABILITY_URL`	observability_api.base_url	API server URL
`OBSERVABILITY_ENABLED`	observability_api.enabled	Enable API

Docker Configuration

```
environment:
  - TZ=UTC
  - CONFIG_PATH=/app/config.json
  - TRAIN_SCHEDULE=0 2 * * *
  - INFERENCE_SCHEDULE=*/10 * * * *
volumes:
  - ./smartbox_models:/app/smartbox_models
  - ./data:/app/data
  - ./config.json:/app/config.json
```

Adding New Services

1. Add to appropriate category in `services` section
2. Optionally add per-service SLO thresholds
3. Optionally add contamination override
4. Run training: `docker compose run --rm yaga train`
5. Verify: `docker compose run --rm yaga inference --verbose`

API Payload Reference

Reference for the JSON payload structure sent by the inference engine.

Alert Types

Type	Description
`anomaly_detected`	Anomalies detected for service
`no_anomaly`	Service is healthy
`metrics_unavailable`	Metrics collection failed

Top-Level Structure

```
{
  "alert_type": "anomaly_detected",
  "service_name": "booking",
  "timestamp": "2024-01-15T10:30:00",
  "time_period": "business_hours",
  "model_name": "business_hours",
  "model_type": "time_aware_5period",

  "anomaly_count": 1,
  "overall_severity": "high",

  "anomalies": { ... },
  "current_metrics": { ... },
  "slo_evaluation": { ... },
  "exception_context": { ... },
  "service_graph_context": { ... },
  "fingerprinting": { ... }
}
```

Current Metrics


```
{
  "current_metrics": {
    "request_rate": 52.7,
    "application_latency": 110.5,
    "client_latency": 1.4,
    "database_latency": 0.8,
    "error_rate": 0.0001
  }
}
```


Metric	Unit	Description
request_rate	req/s	Requests per second
application_latency	ms	Server processing time
client_latency	ms	External dependency latency
database_latency	ms	Database query time
error_rate	ratio	Error rate (0.05 = 5%)

Anomaly Object


```
{
  "latency_spike_recent": {
    "type": "consolidated",
    "root_metric": "application_latency",
    "severity": "high",
    "confidence": 0.80,
    "score": -0.5,
    "signal_count": 2,

    "description": "Latency degradation: 636ms (92nd percentile)",
    "interpretation": "Latency recently increased...",
    "pattern_name": "latency_spike_recent",

    "detection_signals": [ ... ],
    "recommended_actions": [ ... ],
    "comparison_data": { ... },

    "fingerprint_id": "anomaly_d18f6ae2bf62",
    "incident_id": "incident_31e9e23d4b2b",
    "status": "OPEN",
    "occurrence_count": 5,
    "is_confirmed": true
  }
}
```

Detection Signals


```
{
  "detection_signals": [
    {
      "method": "isolation_forest",
      "type": "ml_isolation",
      "severity": "low",
      "score": -0.01,
      "direction": "high",
      "percentile": 91.7
    },
    {
      "method": "named_pattern_matching",
      "type": "multivariate_pattern",
      "severity": "high",
      "score": -0.5,
      "pattern": "latency_spike_recent"
    }
  ]
}
```

SLO Evaluation


```
{
  "slo_evaluation": {
    "original_severity": "critical",
    "adjusted_severity": "low",
    "severity_changed": true,
    "slo_status": "ok",
    "slo_proximity": 0.56,
    "operational_impact": "informational",

    "latency_evaluation": {
      "status": "ok",
      "value": 280.0,
      "threshold_acceptable": 300,
      "proximity": 0.93
    },
    "error_rate_evaluation": {
      "status": "ok",
      "value": 0.001,
      "value_percent": "0.10%",
      "within_acceptable": true
    },
    "database_latency_evaluation": {
      "status": "warning",
      "value_ms": 25.0,
      "baseline_mean_ms": 10.0,
      "ratio": 2.5
    },
    "request_rate_evaluation": {
      "status": "ok",
      "type": "normal",
      "value_rps": 52.7,
      "baseline_mean_rps": 50.0,
      "ratio": 1.05
    },

    "explanation": "Severity adjusted from critical to low..."
  }
}
```

Cascade Analysis


```
{
  "cascade_analysis": {
    "is_cascade": true,
    "root_cause_service": "titan",
    "affected_chain": ["titan", "vms"],
    "cascade_type": "upstream_cascade",
    "confidence": 0.85,
    "propagation_path": [
      {"service": "titan", "has_anomaly": true, "anomaly_type":
"database_bottleneck"},
      {"service": "vms", "has_anomaly": true, "anomaly_type":
"downstream_cascade"}
    ]
  }
}
```

Fingerprinting


```
{
  "fingerprinting": {
    "service_name": "booking",
    "model_name": "business_hours",
    "timestamp": "2025-12-17T13:56:06",
    "overall_action": "UPDATE",
    "total_active_incidents": 1,
    "total_alerting_incidents": 1,

    "action_summary": {
      "incident_creates": 0,
      "incident_continues": 1,
      "incident_closes": 0,
      "newly_confirmed": 0
    },
    "status_summary": {
      "suspected": 0,
      "confirmed": 1,
      "recovering": 0
    },
    "resolved_incidents": [],
    "newly_confirmed_incidents": []
  }
}
```


Exception Context

Present when error SLO breached:

```
{
  "exception_context": {
    "service_name": "search",
    "timestamp": "2024-01-15T10:30:00",
    "total_exception_rate": 0.35,
    "exception_count": 3,
    "top_exceptions": [
      {
        "type": "Smartbox\\Search\\R2D2\\Exception\\R2D2Exception",
        "short_name": "R2D2Exception",
        "rate": 0.217,
        "percentage": 62.0
      }
    ],
    "query_successful": true
  }
}
```

Service Graph Context

Present when client latency SLO breached:

```
{
  "service_graph_context": {
    "service_name": "cmhub",
    "total_request_rate": 2.1,
    "routes": [
      {
        "server": "r2d2",
        "route": "roomavailabilitylistener",
        "request_rate": 0.117,
        "avg_latency_ms": 29.0,
        "percentage": 5.6
      }
    ],
    "top_route": { ... },
    "slowest_route": { ... },
    "summary": "Service graph for cmhub..."
  }
}
```

Resolution Payload


```
{
  "resolved_incidents": [
    {
      "fingerprint_id": "anomaly_061598e9ca91",
      "incident_id": "incident_abc123def456",
      "anomaly_name": "database_degradation",
      "fingerprint_action": "RESOLVE",
      "incident_action": "CLOSE",
      "final_severity": "medium",
      "resolved_at": "2025-12-17T14:30:00",
      "total_occurrences": 5,
      "incident_duration_minutes": 45,
      "first_seen": "2025-12-17T13:45:00",
      "resolution_reason": "resolved"
    }
  ]
}
```


Severity Values

Severity	Priority	Description
critical	1	Immediate action required
high	2	Investigate promptly
medium	3	Monitor closely
low	4	Informational
none	5	No anomaly

Named Patterns

Pattern	Severity	Trigger Condition
`traffic_surge_healthy`	low	High traffic, normal latency/errors

Pattern	Severity	Trigger Condition
`traffic_surge_degrading`	high	High traffic, high latency
`traffic_surge_failing`	critical	High traffic, high latency, high errors
`traffic_cliff`	critical	Very low traffic
`latency_spike_recent`	high	Normal traffic, high latency
`internal_latency_issue`	high	High latency, healthy deps
`error_rate_elevated`	high	Elevated error rate
`error_rate_critical`	critical	Very high error rate
`fast_failure`	critical	Low latency, high errors
`fast_rejection`	critical	Very low latency, very high errors
`database_bottleneck`	high	High DB latency, DB dominant
`database_degradation`	medium	High DB latency, compensating
`upstream_cascade`	high	High latency + upstream anomaly

Troubleshooting

Common issues and solutions for the anomaly detection system.

Detection Issues

No Anomalies Detected (When Expected)

Symptoms: Real incidents but no alerts.

Checks:

```
# Verify models exist
ls -la smartbox_models/<service-name>/

# Check inference logs
docker compose exec yaga tail -20 /app/logs/inference.log

# Run verbose inference
docker compose run --rm yaga inference --verbose
```

Common Causes:

Cause	Solution
No trained model	Run `docker compose run --rm yaga train`
Service not in config	Add to `services` section
Contamination too high	Lower contamination for service
Model stale	Retrain with fresh data

Too Many False Positives

Symptoms: Alerts for normal behavior.

Solutions:

1. **Increase contamination**:

```
{
  "model": {
    "contamination_by_service": {
      "noisy-service": 0.08
    }
  }
}
```

2. **Adjust SLO thresholds**:

```
{
  "slos": {
    "services": {
      "noisy-service": {
        "latency_acceptable_ms": 600,
        "error_rate_floor": 0.005
      }
    }
  }
}
```

3. **Retrain models** after configuration change.

Alerts Always Low Severity

Symptom: All alerts show `severity: low`.

Cause: SLO status is `ok` for all anomalies.

Check: Review SLO thresholds - they may be too lenient.

```
{
  "slos": {
    "services": {
      "booking": {
        "latency_acceptable_ms": 200,  // Was 500
        "latency_critical_ms": 400     // Was 1000
      }
    }
  }
}
```

Incident Lifecycle Issues

Alerts Not Being Sent

Symptom: Anomalies detected but no alerts reach web API.

Possible Causes:

1. **Still in SUSPECTED state** (not confirmed):
   - Wait for 2 consecutive detections
   - Check `is_confirmed: false` in output
2. **Web API unreachable**:

   ```
   docker compose exec yaga curl http://observability-
   api:8000/health
   ```

3. **API disabled in config**:

   ```
   {
     "observability_api": {
       "enabled": true  // Check this
     }
   }
   ```

Orphaned Incidents in Web API

Symptom: OPEN incidents that never resolve.

Cause: SUSPECTED alerts were sent before v1.3.2.

Solution: Clean up orphaned incidents manually or wait for next detection cycle.

Incidents Restarting Unexpectedly

Symptom: Same anomaly creates new incident ID.

Cause: Time gap exceeded `incident_separation_minutes`.

Check: Look for `resolution_reason: "auto_stale"` in logs.

Adjust:

```
{
  "fingerprinting": {
    "incident_separation_minutes": 60  // Was 30
  }
}
```

Metrics Issues

metrics_unavailable Alerts

Symptom: `alert_type: "metrics_unavailable"` instead of detection.

Cause: VictoriaMetrics unreachable.

Checks:

```
# Test connectivity
docker compose exec yaga curl -s
"http://vm:8428/api/v1/status/buildinfo"

# Check circuit breaker
docker compose exec yaga tail -50 /app/logs/inference.log | grep -i
"circuit"
```

Solutions:

- Fix VM connectivity
- Wait for circuit breaker timeout (5 min default)
- Increase timeout in config

Validation Warnings

Symptom: `validation_warnings` in output.

Example:

```
{
  "validation_warnings": [
    "error_rate: value 1.5 > 1.0, capping at 1.0"
  ]
}
```

Cause: Upstream data quality issue.

Action: Investigate metric source. Detection still runs with sanitized values.

SLO Issues

SLO Evaluation Failed

Symptom: Warning in logs about SLO evaluation.

Check: Ensure SLOEvaluator has correct method calls:

```
# Correct
slo_evaluator.evaluate_metrics(current_metrics, service_name)

# Incorrect
slo_evaluator.evaluate(...)  # Method doesn't exist
```

Database Latency Always OK

Symptom: DB latency anomalies show `status: "ok"`.

Cause: Values below noise floor.

Check: Current value vs floor:

```
{
  "database_latency_evaluation": {
    "value_ms": 2.0,
    "floor_ms": 5.0,
    "below_floor": true  // ← This is why
  }
}
```

Adjust: Lower floor if needed:

```
{
  "slos": {
    "services": {
      "fast-db-service": {
        "database_latency_floor_ms": 1.0
      }
    }
  }
}
```

Training Issues

Training Fails

Symptoms: Models not updating.

Checks:

```
# Check training logs
docker compose exec yaga cat /app/logs/train.log

# Check VM connectivity
docker compose exec yaga curl -s
"http://vm:8428/api/v1/status/buildinfo"

# Verify disk space
df -h
```

Common Causes:

Cause	Solution
VM unreachable	Check network
Not enough data	Need 30 days minimum
Disk full	Clean old models/logs

Model Drift Detected

Symptom: `drift_warning` in output.

Check drift score:

- Score < 3: Normal
- Score 3-5: Monitor, consider retraining
- Score > 5: Retrain soon

```
# Retrain
docker compose run --rm yaga train
```

Container Issues

Container Keeps Restarting

Check exit code:

```
docker compose ps -a
docker compose logs --tail 50
```

Common Causes:

Cause	Solution
Config missing	Ensure config.json exists
Invalid JSON	Validate config syntax
OOM killed	Increase memory limit

Database Locked

Symptom: `database is locked` error.

Cause: Concurrent access to SQLite.

Solution: Only run one inference at a time, or use proper locking.

Debugging

Enable Verbose Logging

```
# Run with verbose
docker compose run --rm yaga inference --verbose

# Or set in config


{
  "logging": {
    "level": "DEBUG"
  }
}
```

Inspect Model Details

```
# Check model metadata
docker compose exec yaga cat
/app/smartbox_models/<service>/business_hours/metadata.json | python
-m json.tool
```

Check Fingerprint Database

```
# List active incidents
docker compose exec yaga sqlite3 /app/data/anomaly_state.db \
  "SELECT fingerprint_id, status, occurrence_count FROM
anomaly_incidents WHERE status != 'CLOSED';"
```

Getting Help

1. Check logs: `docker compose logs --tail 100`

2. Run verbose: `docker compose run --rm yaga inference --verbose`
3. Review configuration: Ensure all paths and URLs are correct
4. Check documentation: See detailed docs in `docs/` directory