Splunk ITSI

ITSI Architecture & KV Store

---

1. ITSI Deployment Model

What is ITSI in Splunk?

Splunk IT Service Intelligence (ITSI) is an **application that runs on top of Splunk Enterprise**.
It does **not ingest data by itself** — it **consumes data already indexed** in Splunk (metrics, events, logs).

Core Components of ITSI Architecture

1. Search Head (SH)

- Hosts:
  - ITSI UI
  - Services
  - KPIs
  - Correlation searches
  - Glass Tables
- Executes:
  - KPI base searches
  - Correlation searches
- Heavy dependency on:
  - Scheduler
  - KV Store

🔴 If Search Head is unhealthy → ITSI becomes unusable

---

2. Indexers

- Store:
  - Metrics (_metrics)
  - Events & logs
  - KPI results

- Execute:
    - Distributed searches initiated by Search Head

---

3. KV Store (MongoDB-based)

- Stores **ITSI configuration & runtime state**
- Acts as ITSI's **metadata database**

(Deep dive covered later)

---

4. Scheduler

- Executes:
    - KPI base searches
    - Correlation searches
- Controls:
    - Search frequency
    - Resource allocation

Poor scheduler performance = delayed KPIs

Common ITSI Deployment Models

1. Single Instance (POC / Demo)

- All components on one server
- Not recommended for production

Use case:

- Learning
- Proof of concept

---

2. Distributed Deployment (Production)

- Dedicated:
    - Search Head
    - Indexers
- KV Store runs on Search Head

Most common **enterprise ITSI setup**

---

3. Search Head Cluster + Indexer Cluster

- Used in:
    - Large enterprises
    - High availability environments
- KV Store replicated across SHC members

⚠️ KV Store consistency is critical in SHC

---

2. Dependencies (ITSI Perspective)

What Are Dependencies in ITSI?

Dependencies represent **relationships between services or components** that affect **health propagation**.

Example:

Database → Application → Web

Types of Dependencies

1. Service-to-Service Dependencies

- Parent service health depends on child service
- Used for:
    - Impact analysis
    - Root cause identification

---

2. Entity-Level Dependencies

- Entity health impacts KPIs
- Example:
    - Host entity impacts CPU KPI

Dependency Direction

Direction	Meaning
Upstream	Root cause source

Direction	Meaning
Downstream	Impacted service

Why Dependencies Matter

- Enables:
    - Health roll-up
    - Failure propagation
    - Root cause analysis
- Without dependencies:
    - ITSI becomes a KPI dashboard only

Dependency Best Practices

- Keep dependency depth minimal
- Avoid circular dependencies
- Model **technical → business flow**

---

3. KV Store Role (Very Critical Topic)

What Is KV Store?

- Embedded **NoSQL database** (MongoDB)
- Runs inside Splunk (splunkd)
- Used by apps like:
    - ITSI
    - ES
    - SOAR (some components)

What KV Store Stores for ITSI

Stored in KV Store	Not Stored
Services	Raw metrics
KPIs	Logs
Thresholds	Events

Stored in KV Store	Not Stored
Entities	Indexed data
Episodes	

👉 **KV Store stores "what ITSI knows" — not "what Splunk collects"**

Why KV Store Is Critical for ITSI

- All ITSI objects live here
- Real-time KPI state stored here
- Episode tracking depends on KV Store

🔴 If KV Store goes down:

- Services disappear
- KPIs stop updating
- Glass tables break

KV Store Architecture (Conceptual)

- Based on MongoDB
- Each ITSI object stored as a document
- Collections used for:
    - itsi_services
    - itsi_kpis
    - itsi_entities
    - itsi_episodes

KV Store in Search Head Clustering

- KV Store replicated across SHC members
- One node acts as:
    - Primary
- Others:
    - Secondary replicas

Failure scenarios:

- KV Store sync lag → ITSI inconsistency
- Split brain → corrupted ITSI objects

Common KV Store Issues in ITSI

- Disk I/O latency
- Corrupted collections
- Memory pressure
- Large episode history

KV Store Best Practices

- Use SSD storage
- Monitor:
    - KV Store status
    - MongoDB processes
- Take regular backups
- Limit episode retention

Entities & Entity Types (ITSI)

---

1. What Is an Entity in ITSI?

An **Entity** represents a **real-world, monitorable component** in your IT environment.

Examples:

- A **server** (Linux / Windows)
- A **database**
- An **application**
- A **network device**
- A **Kubernetes pod**
- A **cloud resource**

👉 **Key idea:**

Entities are the bridge between raw data and services.

Without entities:

- KPIs become static
- Services cannot scale
- Dependency modeling becomes difficult

---

2. Entity Types

What Is an Entity Type?

An **Entity Type** is a **classification** that defines:

- What kind of object the entity is
- What attributes it has
- What metrics apply to it

Think of it like a **template or schema** for entities.

Common Built-in Entity Types

Entity Type	Example
Host	Linux / Windows server
Application	Payment Service
Database	Oracle / MySQL
Network Device	Router / Switch
Service	API Gateway

Custom Entity Types

You can create custom entity types for:

- Microservices
- Kubernetes workloads
- Cloud-native components
- Business-level objects

Example:

- payment_microservice
- aws_alb

- k8s_namespace

Why Entity Types Matter

- Define **metric applicability**
- Enable **consistent KPI reuse**
- Simplify large-scale onboarding

Same entity type = same KPIs = faster service modeling

---

3. Entity Creation

Entities can be created **manually or automatically** based on incoming data.

Methods of Entity Creation

1. Automatic Entity Creation

- Most common method
- Based on:
    - Incoming metrics
    - Field extraction rules
- ITSI automatically detects entities when data arrives

Example:

host=web01

→ ITSI creates entity **web01** of type **Host**

---

2. Manual Entity Creation

Used when:

- No data yet
- Pre-defining services
- Business entities

Steps:

- ITSI UI → Configuration → Entities
- Define entity name
- Assign entity type

- Add attributes

---

3. Lookup-Based Entity Creation

- Used for bulk onboarding
- CSV lookup maps fields → entities

Example lookup:

entity_name,entity_type,environment,app

web01,host,prod,payment

Entity Attributes

Attributes describe the entity and help in:

- Filtering
- Tagging
- Service assignment

Common attributes:

- Hostname
- IP address
- Environment (Prod / QA)
- Application name
- Region

---

4. Tagging

What Is Tagging?

Tagging allows you to **logically group entities** using labels.

Tags are:

- Key-value pairs
- Dynamic
- Reusable

Examples of Tags

env:prod

tier:frontend

app:payment

region:apac

Why Tagging Is Important

- Dynamic service composition
- Simplifies entity selection
- Enables scalable modeling

Example:

Instead of adding 50 hosts manually → add tag app:payment

Tagging Use Cases

- Environment separation
- Application grouping
- SLA-based services
- Team ownership

Tagging Best Practices

- Use consistent naming
- Avoid over-tagging
- Prefer business-meaningful tags

---

5. Metric Mapping

What Is Metric Mapping?

Metric Mapping defines **how incoming metrics are associated with entities**.

Without mapping:

- KPIs won't calculate correctly
- Health scores remain unknown

Metric Flow in ITSI

Metrics → Entity → KPI → Service Health

Metric Mapping Components

- Metric name
- Entity identifier (field)
- Entity type

Example:

cpu.utilization → host=web01 → Host entity

How Metric Mapping Works

- ITSI examines incoming metric fields
- Matches them to entity attributes
- Assigns metrics to the correct entity

Example: Host CPU Metric

Metric:

cpu.utilization

Fields:

host=web01

Mapping:

- Entity Type: Host
- Entity Identifier: host

Result:

- CPU KPI calculated per host entity

Metric Mapping Challenges

- Inconsistent field names
- Missing entity identifiers
- Duplicate entity names

Best Practices for Metric Mapping

- Standardize metric field names

- Use unique identifiers
- Validate mappings early
- Prefer metrics (mstats) over events

Key Relationship Summary

Component	Purpose
Entity Type	Classification
Entity	Real-world object
Tags	Logical grouping
Metric Mapping	Data association

Service & Template Modeling (ITSI)

---

1. What Is a Service in ITSI?

In ITSI, a **Service** represents **what the business actually cares about**, not just infrastructure.

Examples:

- Online Banking
- Payment Gateway
- E-Commerce Checkout
- Core Banking System

👉 **Key idea:**

Services translate technical metrics into business health.

---

2. Types of Services

1. Technical Services

- Infrastructure-level
- Example:

    - Web Server Service
    - Database Service

Used by:

- NOC
- SRE teams

---

2. Business Services

- End-to-end workflows
- Built on top of technical services

Example:

E-Commerce Checkout

├── Web Tier

├── App Tier

└── Database Tier

Used by:

- Management
- Business stakeholders

---

3. Building Business Services

Step-by-Step Service Creation

Step 1: Define Service Scope

- What does the service deliver?
- What is the business outcome?

Example:

“Process online payments successfully”

Step 2: Add Entities

Entities represent:

- Hosts
- Applications
- Databases

- Containers

Selection methods:

- Manual selection
- Tag-based selection (recommended)

Step 3: Add KPIs

KPIs measure:

- Performance
- Availability
- Errors
- Capacity

Examples:

- Response Time
- Error Rate
- CPU Utilization
- Throughput

Step 4: KPI Aggregation

How multiple KPIs combine:

- Average
- Weighted average
- Worst case

Step 5: Assign Weights

- Assign importance to KPIs
- Business KPIs usually have higher weight

Example:

KPI	Weight
Error Rate	40%
Response Time	30%

KPI	Weight
Availability	30%

Step 6: Define Dependencies

Dependencies define **health propagation** between services.

---

4. Templates

What Is a Service Template?

A **template** is a **reusable blueprint** for creating services.

It includes:

- KPIs
- Thresholds
- Dependencies
- Aggregation logic

Why Templates Are Important

- Faster onboarding
- Standardization
- Reduced configuration errors
- Easy scaling

Example:

One “Web Tier Template” → 50 web services

Types of Templates

Template Type	Usage
Technical	Web, DB, App tiers
Business	End-to-end services
Entity-based	Host/Application templates

Template Inheritance

- Child services inherit:
    - KPIs
    - Thresholds
- Can override specific values

---

5. Dependencies (Very Important Concept)

What Are Dependencies in ITSI?

Dependencies define **how service health impacts other services**.

Example:

Database → Application → Web → Business Service

Types of Dependencies

1. Parent–Child Dependencies

- Parent service health depends on child service
- Used for root cause analysis

---

2. Peer Dependencies

- Services depend on each other equally
- Used rarely

Dependency Direction

Direction	Meaning
Upstream	Root cause
Downstream	Impacted services

Health Propagation

- If a child service turns red:
    - Parent service degrades automatically
- Weight defines impact severity

---

6. Dependency Best Practices

- Model **real technical flow**
- Avoid circular dependencies
- Keep depth manageable
- Test failure scenarios

---

7. Common Modeling Mistakes

- Treating infra as business services
- Too many KPIs in one service
- No dependency modeling
- Equal weights everywhere

Key Relationship Summary

Entities → KPIs → Services → Dependencies → Business Health

KPI Base Searches & Thresholding (ITSI)

---

1. What Is a KPI in ITSI?

A **Key Performance Indicator (KPI)** is a **measurable signal** that represents the **health of an entity or service**.

Examples:

- Response Time
- Error Rate
- CPU Utilization
- Availability
- Throughput

👉 **Key concept:**

KPIs convert raw data into health scores.

---

2. KPI Architecture Flow

Raw Data → Base Search → KPI Value → Threshold → Health Score → Service Health

If the base search is wrong, **everything above it is wrong**.

---

3. Base Search Design

What Is a Base Search?

A **base search** is an SPL query that:

- Retrieves raw data (metrics or events)
- Produces a **numeric time-series value**
- Feeds one or more KPIs

Base Search Design Principles

1. Efficiency First

- Runs continuously
- Consumes scheduler resources
- Must be optimized

Best practice:

- Use mstats for metrics
- Avoid heavy joins and subsearches

---

2. One Base Search → Multiple KPIs

- Base search fetches raw data once
- KPIs derive values from it

Example:

```
| mstats avg(cpu.utilization) as cpu_avg
  where host=* span=1m
```

KPIs:

- CPU Average
- CPU Peak
- CPU Variance

---

3. Time Span Selection

- Typical span: 1m or 5m
- Smaller span = higher resolution + more load
- Larger span = lower accuracy

---

4. Metrics vs Events

Type	Recommendation
Metrics	Preferred
Logs	Use only if metrics unavailable

Reason:

- Metrics are faster
- Lower storage
- Designed for time-series KPIs

Base Search Output Requirements

A valid base search must output:

- _time
- Entity identifier (host, service, app)
- Numeric value

If any are missing → KPI fails.

Common Base Search Mistakes

- Returning multiple values per entity
- Missing _time
- Using non-numeric fields
- Over-filtering data

---

4. Thresholding

Thresholds convert **KPI values → health states**.

Health states:

- Green (Healthy)

- Yellow (Warning)
- Red (Critical)

Static Thresholds

What Are Static Thresholds?

Fixed threshold values defined manually.

Example:

Value	Health
< 70%	Green
70–85%	Yellow
> 85%	Red

When to Use Static Thresholds

- Well-known limits
- Infrastructure metrics
- Compliance metrics

Examples:

- Disk usage
- CPU utilization
- Memory usage

Advantages

- Simple
- Predictable
- Easy to explain

Disadvantages

- Not adaptive
- Can generate false positives
- Doesn’t handle seasonality

Adaptive Thresholds

What Are Adaptive Thresholds?

Dynamic thresholds calculated using **historical data** and **machine learning**.

ITSI uses:

- Past behavior
- Time-of-day patterns
- Day-of-week trends

When to Use Adaptive Thresholds

- Transaction volumes
- Response times
- Error rates
- Business KPIs

Advantages

- Fewer false alerts
- Handles spikes & seasonality
- Self-adjusting

Disadvantages

- Requires sufficient historical data
- Less predictable initially
- Harder to explain to business

Static vs Adaptive Thresholds

Aspect	Static	Adaptive
Setup	Simple	Moderate
Flexibility	Low	High
False Positives	Higher	Lower

Aspect	Static	Adaptive
Business KPIs	❌	✅

---

5. KPI Aggregation & Health Calculation

- KPI values evaluated per entity
- Health scores aggregated at service level
- Weighted average used by default

Example:

Response Time (Red) + Error Rate (Yellow)

→ Service Health = Red

---

6. Best Practices

- Start with static → move to adaptive
- Validate base search before KPI creation
- Use consistent time spans
- Monitor scheduler load

Correlation Searches & Episode Review (ITSI)

---

1. Why Correlation & Episodes Matter

In real environments:

- Hundreds of alerts fire for **one incident**
- Operators waste time chasing symptoms

👉 **ITSI's job here**:

Group related events, identify the real issue, and show impact.

This is done using:

- **Correlation Searches**
- **Event Aggregation**
- **Episodes**
- **Root Cause Analysis (RCA)**

---

2. Correlation Searches

What Is a Correlation Search?

A **correlation search** is an SPL-based search that:

- Looks at **events, KPI alerts, or notable events**
- Detects patterns across time, entities, or services
- Generates **notable events** or feeds **episodes**

What Correlation Searches Can Use

- KPI threshold breaches
- Log events
- Metric anomalies
- External alerts (APM, cloud, security)

Correlation Search Use Cases

- High CPU + High error rate
- Multiple hosts failing in same service
- Repeated KPI flapping
- SLA breach patterns

Correlation Search Output

- Notable event
- Episode contribution
- Impacted service/entity
- Severity & time window

Key Characteristics

- Scheduled (runs periodically)
- Rule-based or pattern-based
- Designed to **reduce alert noise**

Good vs Bad Correlation

Bad	Good
One alert per KPI	One episode per incident
No context	Business impact visible
Symptom-based	Cause-based

---

3. Event Aggregation

What Is Event Aggregation?

Event aggregation groups **multiple related events** into **one logical incident**, called an **Episode**.

Why Aggregation Is Needed

Without aggregation:

- 50 alerts = 50 tickets
  With aggregation:
- 50 alerts = **1 episode**

Aggregation Criteria

Events are grouped based on:

- Time window
- Entity
- Service
- Severity
- KPI type

Example:

CPU spike (Host A)

Memory spike (Host A)

Disk latency (Host A)

→ 1 Episode

Aggregation Methods

- Time-based grouping
- Entity-based grouping
- Service-based grouping
- Rule-based grouping

Aggregation Window

Defines how long events are grouped together.

Example:

- 5 minutes
- 15 minutes
- 30 minutes

👉 Too small → too many episodes
👉 Too large → unrelated events grouped

---

4. Episodes

What Is an Episode?

An **Episode** represents:

- A **single incident**
- With **multiple contributing events or KPIs**
- Across entities and services

What an Episode Contains

- Start time & duration
- Severity
- Contributing KPIs
- Impacted services
- Suspected root cause

Episode Lifecycle

1. Episode opens
2. Events added

3. Root cause suggested
4. Episode closes (manual or auto)

Episode Benefits

- Single pane of glass
- Faster triage
- Reduced MTTR
- Clear business impact

---

5. Root Cause Analysis (RCA)

What Is RCA in ITSI?

RCA is ITSI's ability to:

- Identify **where the problem started**
- Separate **cause vs symptom**
- Show **impact propagation**

How ITSI Performs RCA

ITSI uses:

- Service dependencies
- KPI health timelines
- Event order
- Entity relationships

RCA Example

Database latency ↑

→ App response time ↑

→ Web errors ↑

→ Checkout service ↓

Root Cause:
➞ **Database latency**

RCA Visualization

- Dependency graphs
- Health degradation timeline
- Episode contributor ranking

Probable Root Cause

ITSI assigns:

- Confidence score
- Based on earliest degradation
- Dependency position (upstream)

---

6. Relationship Between Concepts

KPI Alerts

↓

Correlation Searches

↓

Event Aggregation

↓

Episode

↓

Root Cause Analysis

---

7. Best Practices

Correlation Searches

- Keep rules simple
- Correlate only meaningful KPIs
- Avoid duplicate rules

Event Aggregation

- Tune time windows carefully
- Group by entity/service

- Avoid over-aggregation

RCA

- Model dependencies correctly
- Ensure KPI timestamps are accurate
- Avoid circular dependencies

---

8. Common Mistakes

- No dependency modeling → bad RCA
- Correlating everything → noise
- Too many correlation searches
- Ignoring episode tuning

Service Analyzer & Glass Tables (ITSI)

---

1. Service Analyzer

What Is Service Analyzer?

The **Service Analyzer** is ITSI's **primary operational view** that shows:

- Overall service health
- KPI health breakdown
- Dependency-driven impact
- Active episodes

👉 Think of it as the **mission control** for IT operations.

Purpose of Service Analyzer

- Provide **single-pane visibility**
- Help NOC/SRE teams:
    - Detect issues fast
    - Prioritize incidents
    - Understand business impact

What Service Analyzer Displays

- Service name
- Health score (0–100)
- Health color:
    - Green (Healthy)
    - Yellow (Degraded)
    - Red (Critical)
- Child services & dependencies
- Linked episodes

Service Hierarchy View

Services are displayed in a **tree structure**:

Business Service

├— Application Service

| ├— Web Tier

| └— App Tier

└— Database Service

👉 Health **rolls up** from bottom → top.

Drilldowns in Service Analyzer

From a single click, users can:

- Open KPI details
- View entity-level health
- See contributing episodes
- Jump to Glass Tables

---

2. Health Scores

What Is a Health Score?

A **numeric value (0–100)** representing service or KPI health.

Score Range	Meaning
90–100	Healthy
70–89	Warning
< 70	Critical

How Health Scores Are Calculated

Health score is based on:

- KPI values
- Threshold breaches
- KPI weights
- Dependency impact

Example:

Error Rate (Red, weight 40%)

Response Time (Yellow, weight 30%)

Availability (Green, weight 30%)

→ Overall Health = Red

Why Health Scores Matter

- Abstracts raw metrics
- Enables **quick decisions**
- Makes IT data understandable for business users

Business leaders don't want CPU charts — they want **"Is my service OK?"**

---

3. Business Visuals

What Are Business Visuals?

Visual representations that map **technical health → business process flow**.

Used in:

- Executive reviews
- War rooms

- Incident calls
- NOC displays

Business Visual Examples

- Order flow
- Payment lifecycle
- Login → Checkout journey
- Transaction pipeline

---

4. Glass Tables

What Is a Glass Table?

A **custom, interactive visual dashboard** that overlays:

- Services
- KPIs
- Health scores
- Dependencies

On top of:

- Images
- Diagrams
- Architecture layouts

Why Glass Tables Are Powerful

- Visual storytelling
- Real-time health
- Business + IT alignment
- Extremely intuitive

👉 This is the **most demo-worthy ITSI feature**.

Glass Table Components

1. Shapes

- Represent services or entities

- Change color based on health

---

2. KPIs

- Embedded directly on visuals
- Show live values and trends

---

3. Icons & Images

- Application icons
- Architecture diagrams
- Data flow illustrations

---

4. Interactions

- Click to drill down
- Hover for KPI details
- Navigate to Service Analyzer

Glass Table Health Behavior

- Colors auto-update in real time
- Health propagation follows dependencies
- Episodes can be highlighted visually

Example:

Database icon turns Red

→ App turns Yellow

→ Business Service turns Red

---

5. Real-Time Dashboards

What Makes Them "Real-Time"?

- Auto-refresh
- KPI base search updates
- Live episode integration

Real-Time Dashboard Use Cases

- NOC wallboards
- 24×7 monitoring
- Incident bridges
- Executive status screens

Dashboard Characteristics

- Minimal clutter
- Focus on health, not raw logs
- Visual-first, data-second

Best Practices for Service Analyzer

- Limit KPI count per service
- Use meaningful weights
- Keep hierarchy simple
- Regularly review stale services

Best Practices for Glass Tables

- Design for non-technical users
- Avoid too many KPIs on one screen
- Use consistent color semantics
- Align visuals with real architecture

Common Mistakes

- Treating Glass Tables like dashboards
- Overloading visuals
- No drilldowns
- Poor dependency modeling

Concept Relationship Summary

KPIs → Health Scores

Health Scores → Services

Services → Service Analyzer

Services → Glass Tables

Episodes → Visual Impact