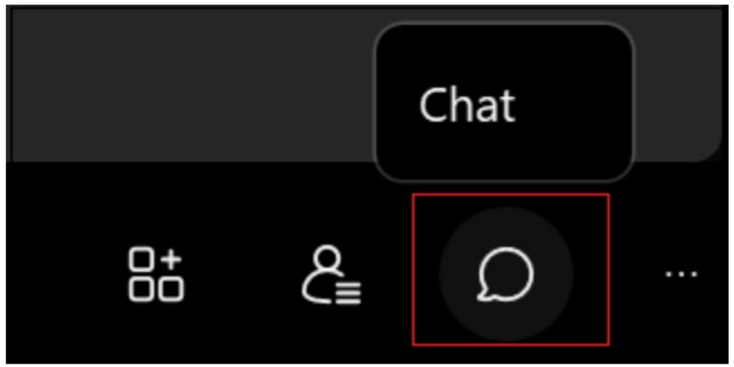
IBM Sterling Order Management

Holiday Readiness 2023

Recommendations
Best Practices

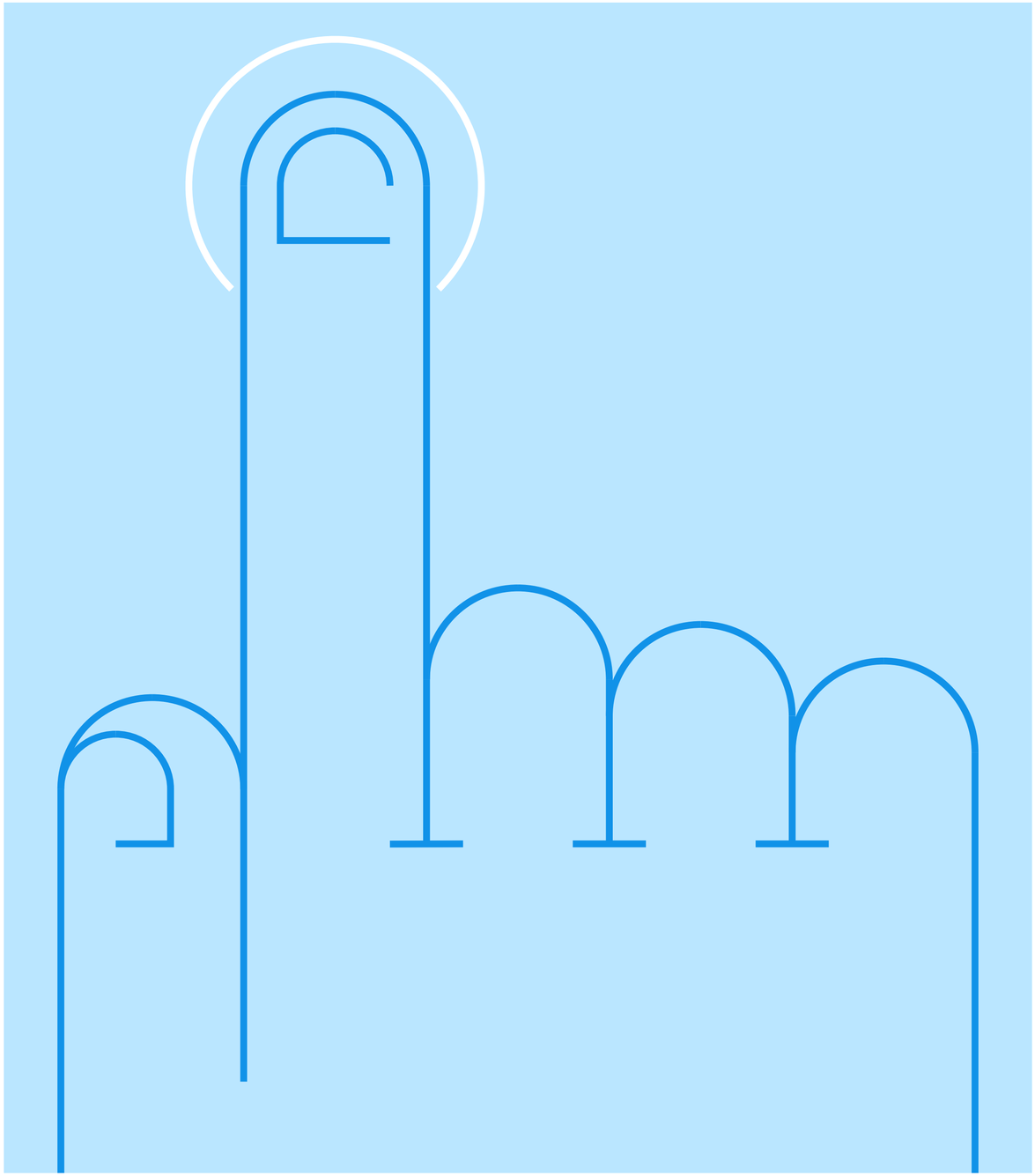
Have a Question(s)?

1. Open the Chat panel from the link in the lower right of the meeting window:



2. In the **To** drop-down list, select the recipient of the message.

3. Enter your message in the chat text box, then press **Enter** on your keyboard.



Your Holiday Readiness Team
… and today's speakers

Chris Burgess
Manager –
WW Support Experience Team

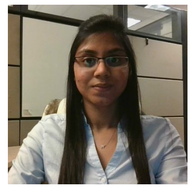
Mike Callaghan
Program Director –
WW Supply Chain Support

Shoeb Bihari
Technical Lead / SRE Advisor –
Order Management Support

Senthil Ponnusamy
Technical Lead / SRE Advisor –
Order Management Support

Jitendra Buge
Technical Support Engineer
Order Management Support

Paresh Vinaykya
Executive Technical Account
Manager – Expertise Connect



Damini Tacouri
Technical Support Analyst
Order Management Support

Abdul Shad
Technical Lead –
Order Management Support

Agenda

Our Journey to peak success

Recommended Configuration

- ✓ Application
- ✓ Integrations
- ✓ Database
- ✓ Sterling Intelligent Promising
  - Inventory Visibility
- ✓ Payments

Performance Testing & Optimization

Self Service & Monitoring Overview

Proactive Engagement

What Not to Do

Plan

- ☒ Aggressive rollout plans
- ☒ Outdated product, stack
- ☒ Narrow test coverage
- ☒ Unclear peak workloads

Prepare

- ☒ Poor DB hygiene
- ☒ Major DG changes
- ☒ Defer Q3 push
- ☒ Last-minute fix, deploy
- ☒ Open recommendations

Execute

- ☒ Oversell detected
- ☒ BOPIS orders delayed
- ☒ Major Escalation
- ☒ Chaotic War-room
- ☒ Tedious triage
- ☒ Tune on the fly

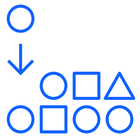
MC

IBM OMS Holiday Readiness

Our Mission Statement

Stable Platform

Continuous improvement of platform and monitoring, with focus on performance, stability, reliability

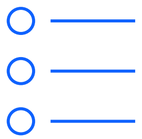


Best Practices

Establish, expand and apply a robust collection of proven self-help best practices focused on peak season success

Proactive Engagement

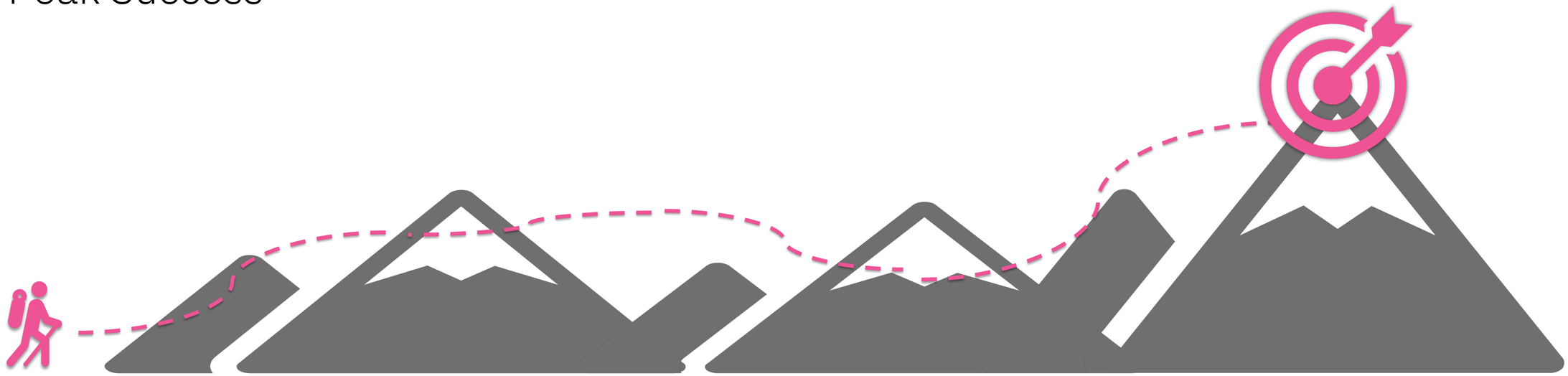
Early and regular identification, communication, and mitigation of potential risks



Prescriptive Guidance

Deeper partnership with specific clients in need of direct analysis and prescriptive guidance via our Enhanced Event Readiness offering

Journey to Peak Success



Plan

- ❑ Retrospective
- ❑ Align Business and IT
- ❑ Platform Enhancements
- ❑ Align Schedules
- ❑ Know the Best practices
- ❑ Identify Risks

Prepare

- ❑ Implement Best Practices
- ❑ Ongoing housekeeping
- ❑ Performance test, tune
- ❑ Know, test Breaking points
- ❑ Failover, DR scenarios
- ❑ Monitoring & Alerting

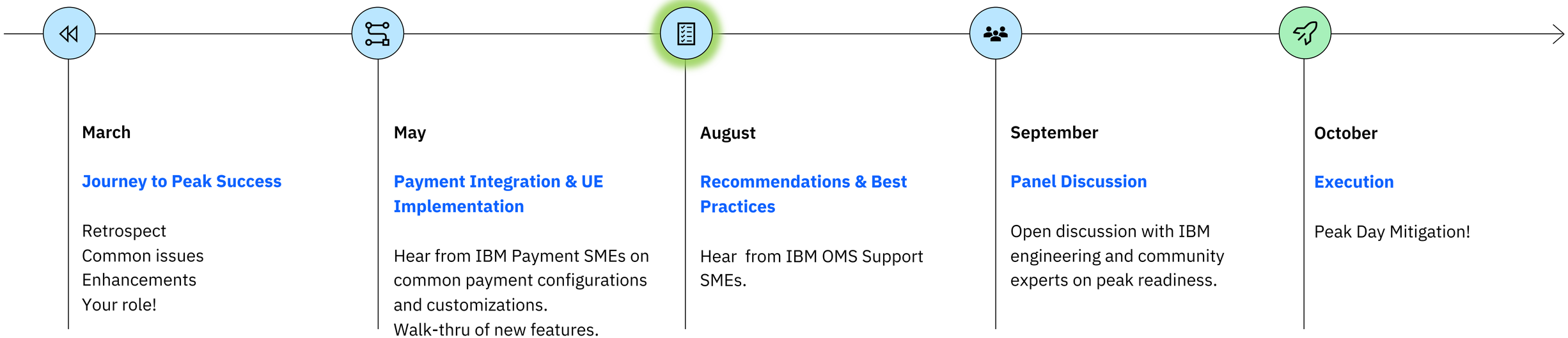
Execute

- ❑ Roles & responsibilities
- ❑ Escalation paths
- ❑ Critical Workloads
- ❑ Triage Runbooks
- ❑ Mitigation Techniques
- ❑ Communication Plan

The Path Ahead

Journey to Peak Success

The IBM OMS Support team are continuously expanding our technical best practices based on the observations and learnings over our supported launches and peak events!



March

Journey to Peak Success

Retrospect
Common issues
Enhancements
Your role!

May

Payment Integration & UE Implementation

Hear from IBM Payment SMEs on common payment configurations and customizations.
Walk-thru of new features.

August

Recommendations & Best Practices

Hear from IBM OMS Support SMEs.

September

Panel Discussion

Open discussion with IBM engineering and community experts on peak readiness.

October

Execution

Peak Day Mitigation!

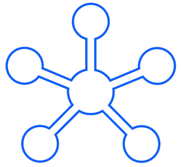
Best Practices

Apply Recommended Configuration

The IBM OMS Support team are continuously expanding our technical best practices based on the observations and learnings over our supported launches and peak events!

Application

- ✓ Use Optimal API and optimize output template
- ✓ Fine tune entity cache, disable redundant.
- ✓ Use HOTSku & OLA configuration
- ✓ Enable capacity cache
- ✓ Set API isolation level based on the use case
- ✓ Use pagination for getter APIs



Integration

- ✓ Use JMS session pooling
- ✓ Avoid message selectors
- ✓ Enable service retry (transaction reprocessing)
- ✓ Have timeout’s for up/down stream synchronous calls
- ✓ Inventory Visibility (SIP)

Database

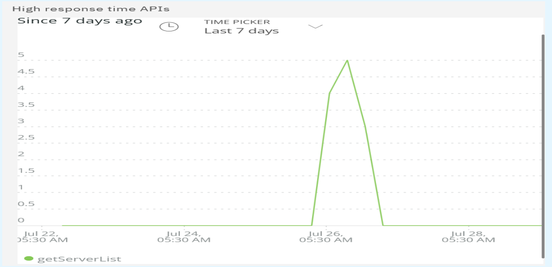
- ✓ Maintain transactional tables, Purge (set appropriate retentions)
- ✓ Serviceability & Monitoring
- ✓ Minimize contention, maximize concurrency
- ✓ Optimize long-running or expensive queries
- ✓ Enable *stmt_conc* (LITERALS)*

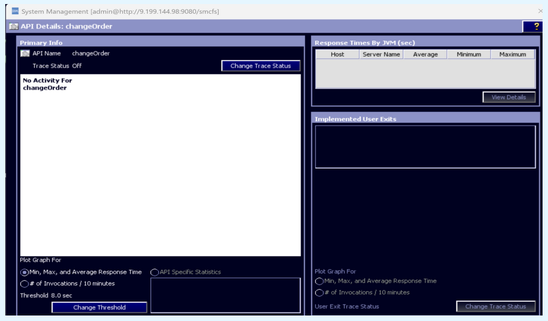
API Performance

These best practices can help you achieve required API response times.

Common issues:

1. getAvailableInventory call taking more than 30 secs
2. getOrderDetails and getOrderList API call has high response time
3. OMS services response time is high





API Template Optimization

- Keep only required attributes
- Strive not to use the `TotalNumberOfRecords` attribute all the time
- Limit the number of records returned
  - `MaximumRecords`
- API template sequence Read more →

Long running queries

- Very large data to retrieve
- Poor indexing
- DB statistics very old
- Reorg not done timely
- lock-wait
- Purges not running
- Use 'Database metrics' SST dashboard
- Use Pagination for get APIs

Overhead of excessive logging

- Increases response time of all transactions
- Logging can cause high Memory/GC utilization
- Use appropriate logging levels
- `TraceTTL` enhancement

Blank open-ended queries

- `SELECT YFS_ORDER_HEADER.* FROM YFS_ORDER_HEADER YFS_ORDER_HEADER WHERE ( ( ? = ? ) ) ORDER BY ORDER_HEADER_KEY`
- In custom code, ensure to search with some filters to avoid whole table scans

–Select wait method

- `WAIT` or `NO_WAIT` mode
- Query timeout properties:
  - `yfs.agentserver.queryTimeout`
  - `yfs.ui.queryTimeout`

Read more →

Timeouts for external calls

- Ensure the processing time of the user exits and events are small.
- Ensure that call-out (requests) to external systems can scale beyond anticipated peak processing rates
- Ensure that you do not hold critical record locks during the call out.

Monitor API Response Time

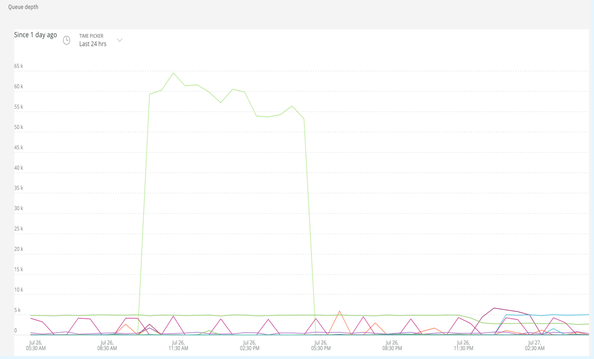
- SST OMS API Performance dashboard
- `YFS_STATISTICS_DETAIL Table`
- SMC
  - Slow response time for a user exit
  - Response Time By JVM panel

JMS performance tuning and Best Practices

These best practices can help you achieve your NFRs.

Common issues

1. Message PUT slowness
2. Agent (GetJobs) is slow
3. Consumer is slow



Performance Properties

- Use JMS Session Pooling
  - `yfs.yfs.jms.session.disable.pooling=N`
- Use anonymous reuse (requires JMS Session pooling to be enabled)
  - `yfs.jms.sender.anonymous.reuse=true`
- Enable multi-threaded PUT's
  - `yfs.yfs.jms.sender.multiThreaded=Y`
- Bulk sender properties
  - `yfs.agent.bulk.sender.enabled=Y`
  - `yfs.agent.bulk.sender.batch.size=5000`

Read More

Message Persistence

- For agents, define your queues as non-persistent which can improve agent performance
- All integration queues used for external communications must be defined as persistent.

Measuring the performance

- Review `MessageBufferPutTime` relative to `ExecuteMessageCreated` statistic from
- `YFS_STATISTICS_DETAIL` table for any slowness
- Ensure JMS performance properties are in place

Enable reconnect from external applications

- In case of connection errors/ MQ server restarts, ensure that your application is either configured for auto reconnect or have the ability to retry the connection.

Enable retries for JMS Sender

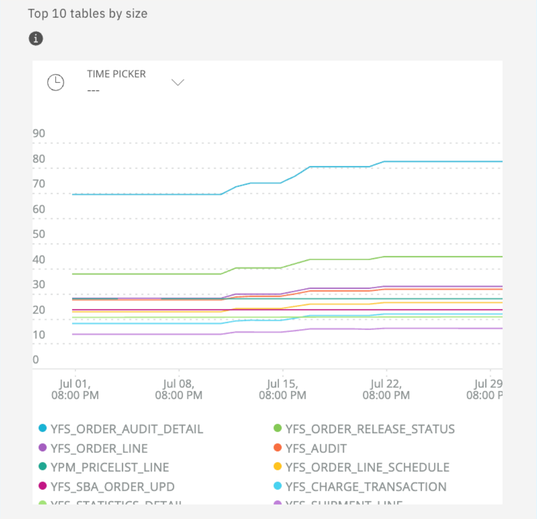
- While using JMS Sender, ensure you are configuring the number of retries and retry interval.
- Configure minimum three reties with the interval of 100 milliseconds

Other Recommendations

- Validate MQ settings
- Avoid long-running transactions
- Avoid using message Selector, instead have dedicated queues.
- Ensure necessary queue-depth alerts are configured

Database performance

Maintaining a healthy database can prevent unexpected performance issues during peak days.



Database Size

- Ensure purges are running for all tables and records are becoming eligible for purge.
- Enable compression on CLOB columns to reduce its footprint. Read more →
- Disable unwanted audits & debugging message flowing into `YFS_EXPORT` table

Query Optimization

- Suboptimal query tuning, there could be a cases where custom queries aren't using correct filtering columns or pulling out large data set
- Blank queries, one of the common use case when non optimal API input is passed in. Ensure APIs are invoked with key filtering input.

Proactive monitoring

- Use SST to proactively monitor long running and lock wait queries during your performance testing and ongoing basis.
- Keep an eye on table size growth and address it regularly.

Cache Management

- Identify the optimal cache sizing through your performance testing
- Monitor the frequently invalidated table caches and disable them if needed
- Ensure SQLs aren't formed with unique values at runtime, it impacts the cache reusability.
- Enable *`stmt_conc (LITERALS)*`*

DB CPU / IO / Transaction log space

- Heavy query execution from DBQuery client tool or like EOD report through REST calls
- Blank queries without correct filtering condition (API template issue)
- Waiting on external system without releasing the DB lock.
- Timeboxing non critical (like purge) workloads during off hours.

Lock-Wait / Long running Queries

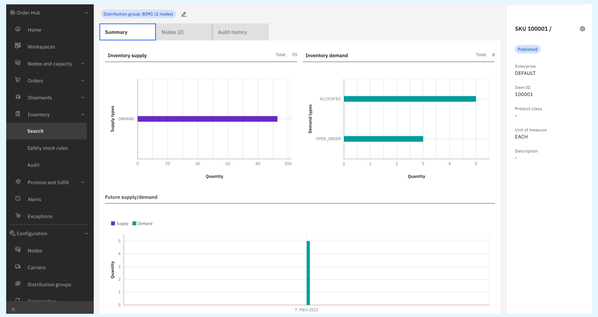
- Long running queries
  - Non optimized SQL
  - Pulling out large amount of data
  - Not using correct filtering clause
  - Indexing
  - DB maintenance
- Lock-Wait Condition
  - Along with the above details,
  - HOTSku
  - Capacity caching
  - API input (select method)
  - Workload separation etc.,

Sterling Inventory Visibility (IV)

These best practices can help you achieve your NFRs.

Common issues - 2022

1. Incorrect availability during internal event-based audit.
2. Slow API response due to excessive token generation.
3. Synchronize inventory for entire catalog



Token Management

- Reuse IV token as much as possible
- Generating token instead of reusing can negatively affects performance.
- Automatically regenerate a token before the expiration time or as part of error handling on 401 Unauthorized or 403 Forbidden response. Read more →

Use Optimal Payload

- Adhere to best practice when invoking IV APIs. Read more →

- IBM recommends no more than 100 items per payload when invoking IV APIs with multiple lines.
  - Availability lookup
  - Supply adjustment / sync
  - Reservations

Event Management

- Do not miss failed events; implement a process to retrieve failed events.
  Read more →

- Supply/Demand audit looks correct; however, cached inventory picture does not match availability in IV.

Avoid redundant Snapshot calls

- Only publishes the current inventory picture

- Space out the sync supply and snapshot calls

- If possible, sync only items for which inventory picture changed

Avoid redundant Network availability recomputes

- Recompute Network Availability API recomputes availability for existing DG.

- Update DG API will recompute availability for newly created or modified DGs but not for existing DG.

Use enhanced APIs

- Do not use depreciated APIs, instead use updated APIs.

- Item FO/Threshold APIs
- Safety stock APIs
- Event Threshold APIs
- Ship node and DG management APIs are moved under Sterling Intelligent Promising.

Read more →

Payment Integration

- Prior Webcast:

- Payments Deep Dive session

- Details on new serviceability enhancements and walk-through of the new **Payment Audit** feature.

- Example happy path scenario with the **Dynamic Charge Transaction Request Distribution** feature.

- Recommendations, best practices, common payment configurations, Do's & Don'ts based on lessons learned from common issues seen during prior peak seasons.

Automatic Hold

- Implement the Automatic order hold so that if there is a looping condition detected due to payment mismatch, order can be put on hold via change order. Read more →

  - `yfs.payment.infiniteLoop.paymentHoldType`
  - `yfs.payment.infiniteLoop.allowViewingOfOrder`

APIs

- Review *`javadocs`* before implementing `processOrderPayments`, and use `RequestCollection, ExecuteCollection, RequestCollection`.

- Do not call `processOrderPayments` as part of long transaction boundary.

- This API is intended for In-person scenarios e.g., carry lines.

UserExit

- Tax related UE output should include all necessary taxes to avoid wiping out previous existing taxes.

- Correct authorization IDs should be stamped along with corresponding expiration dates.

- Handle all the exceptions from the collection UE. Otherwise, charge and authorization transactions will get stuck in the 'invoked' user exit status.

Excessive Charge Transaction Records

- Review orders having many charge transaction records.

- Having excessive YCT records shows underlying issue.

- Place orders having excessive YCT on hold, to prevent further processing.

```
- SELECT ORDER_HEADER_KEY, COUNT(*) FROM
OMDB.YFS_CHARGE_TRANSACTION GROUP BY
ORDER_HEADER_KEY HAVING COUNT(*) > 100
ORDER BY ORDER_HEADER_KEY DESC WITH UR;
```

Monitor Backlog

- Query `YFS_ORDER_HEADER` table to get payment collection backlog, refer to getJobs query.

- Query `YFS_CHARGE_TRANSACTION` table to get payment execution backlog, refer to getJobs query

- Queries indicate how many orders are eligible to be picked and processed by the agents.

- Redundant processing of problematic orders can lead to bottlenecks.

Payment Collection Failure

Ensure the following parameter is set to ensure `PAYMENT_COLLECTION` agent does not fail with `java.lang.IllegalArgumentException: Comparison method violates its general contract!`

Read more →

Performance Testing & Optimization

Performance is a non-functional requirement, impacting the quality of the **user experience**

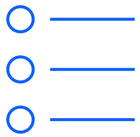
A strategy for a good performance test is to use a mixture of concurrent scenarios that involve read and write operations.

Plan

Establish and quantify goals and constraints

What business wants?

Identify KPI’s & NFR’s



Prepare

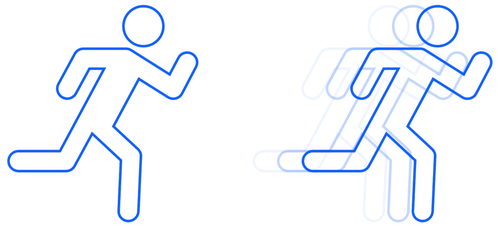
Populate the application with realistic data

Use a phased ‘stair-case’ model

Execute

Simulate workload

Tune!



Scale

Until it meets your NFRs!

Performance Testing Guidance

Performance testing is an art, but a mandatory one! It is imperative to vet out issues in advance on pre-production load testing, rather than wait for it to surface as a business-critical production issue!

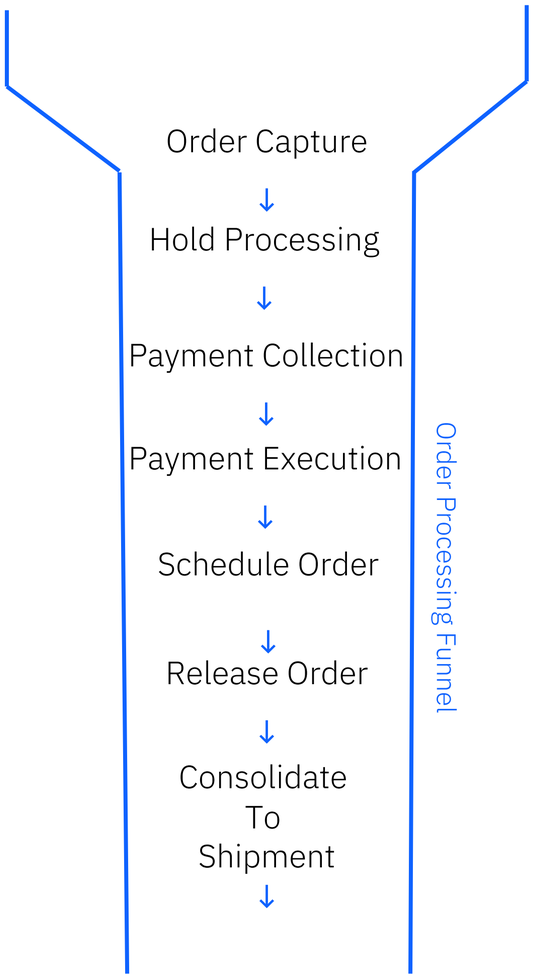
1. **Projected peak volumes** – Ensure business and IT are in sync on expected peak loads to ensure planned tests are accurate.
2. **Representative Combination Tests** – Assemble components to reflect real time DATA, scenarios and run in parallel to ensure adherence with NFR; Stage data for various components and run them under full load (ie. Create + Schedule + Release+ Create Shipment + Confirm Shipment + Inventory Snapshot (IV))
3. **Agent and integration servers –** ensure asynchronous batch processing components are tested in isolation and in combination with broader workload; ensure to tune agents (processes, threads, profile) to meet expected peak SLAs/NFRs on throughput

Metric	2022 Peak	2023 Projected Peak	2023 Load Test Peak
Orders / hour (max)	?	?	?
Orderlines / hour (max)	?	?	?
Get Inventory Availability	?	?	?
Reserve Inventory	?	?	?
Inventory Adjustment trickle	?	?	?
Inventory Adjustment burst	?	?	?
Concurrent Store/CC users	?	?	?

4. **Test Failure Scenarios** – validate resiliency of overall system and operations, ensuring graceful recovery if front-end channel (web, mobile, Call Center, Store, EDI, JMS), backend OMS, or external integration endpoints fail. Include ‘kill switches’ in any components that can be disabled to avoid magnifying an isolated issue into system wide one, especially for any synchronous calls.
5. **Confirm Peak days and Hours -** Share any specific key dates or max burst times with IBM Support, including code freezes, flash sales.
6. **Coordinate with IBM -** Inform IBM (CSM/Support) in advance when load tests are planned if any data or diagnostics (such as against Database) are needing to be captured; IBM can also then review internal metrics and response in parallel. ***➔ Inform IBM in advance of major configuration changes (sourcing rule: Increase in Ship from Store orders).***

Refer to Knowledge Center for detailed Tuning and performance guidance.

Server Profile



Next-generation

Select optimal performance profile

Select optimal server profile and thread configuration for agent processes and integration service to ensure service can scale w/ custom logic and configuration.

Example: Target to achieve 30k TPH for createOrder w/ 2.5 average lines

Approach:

Configure create order integration server in OMS

↓ Initial Threads =1, Performance Profile = Balanced
Default # of JVM Instances = 1

Execute and **monitor** the server performance via **Self Service Tool**

↓ Monitor KPI's: API Response time (ms), Invocations (rpm), Container CPU Utilizations, GC CPU Utilizations, JVM Heap Utilization, and Order Lines Throughput

Observe and **Adjust** the configuration until throughput is achieved

Increase the # of threads
Switch Performance Profile
Increase the # of JVM instance

NOTE: Below numbers represent OOB createOrder with some customization

Server Type	Integration		Performance Profile	Balanced			
Server Name	CreateOrderServer						
JVM Instances	No. of Threads Per JVM Instance	API Response (ms)	Invocations (rmp)	Container CPU %	GC CPU %	Heap Utilization	Order/Hour (2.5 average lines)
1	1	198	305	51.5	3	52.7	18760
1	2	285	420	85	7	57.8	25200
1	3	410	432	96	12	62.4	25920
2	4	580	804	99	19	70	47398

Server Type	Integration		Performance Profile	Compute			
Server Name	CreateOrderServer						
JVM Instances	No. of Threads Per JVM Instance	API Response (ms)	Invocations (rmp)	Container CPU %	GC CPU %	Heap Utilization	Order/Hour (2.5 average lines)
1	1	182	324	25.6	1.4	29.1	19440
1	2	196	612	47	3	35.5	36720
1	3	219	810	61	5.87	44.2	48600
1	4	230	1041	75	6.47	51.7	62100

Optimal Solution:
Threads = 3
Performance Profile = Compute
JVM Instances = 1

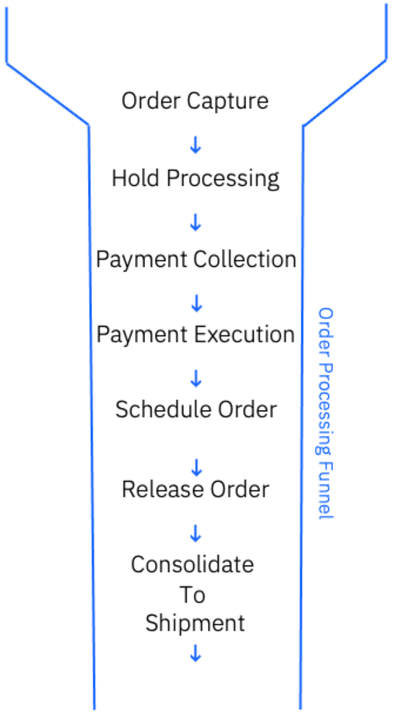
Recommendations:

- Spawning additional (untuned) instances of agent to try and improve throughput let to **exhaustion of resource** allocation available
- Review KC Guidelines to select performance profile | Review community article on Sterling OMS Performance Profiles

Order Processing Funnel

01

Order Flow



Best Practices

- Apply recommended JMS performance properties Read more →
- Review order and shipment monitors for redundancy, review and remove obsolete monitor rules.
  - Avoid reprocessing of order once condition evaluates to false. Read more →
    `yfs.yfs.monitor.stopprocessing.ifcondition.eval.false=Y`
- Tune next task queue interval of "Process order hold type" agent from 15 minutes to the customized value `yfs.omp.holdtype.reprocess.interval.delayminutes`
- Have dedicated schedule order server to process backorders using OrderFilter= N|B agent criteria parameter.
- Use workload separation to isolate the transaction by order or release attributes Read more →
- Apply and Tune OMoC default HOTSku and OLA configuration Read more →
- Enable Capacity cache and tune node locking properties based on business use case.
- Apply sourcing optimization (reduce DG size, region-based sourcing)
- When using `YFSGetAvailabilityCorrectionsForItemListUE`, make sure output of the UE excludes the items with ZERO supply quantity before passing the result to OOB API.
- Apply solver/sourcing interrupt properties to prevent runaway transactions
- If capacity is enabled, then make sure to check the calendar setting (store hours, etc.) for peak.
- Disable capacity instead of setting it very high value.
- Control/Throttle use of createInventoryActivityList API when using capacity filled event.
- Run Inventory purge

Plan and take necessary action to position y(our) solution for success, it is critical to *TAKE ACTION NOW!*

↓

01
User Exit

- Make sure correct authorization IDs are stamped along with corresponding expiration dates.
- Records having the same authorization IDs should have the same authorization expiration date.
- Handle all the exceptions from the collection UE. Otherwise, charge and authorization transactions will get stuck in the 'invoked' user exit status.
- `RecalculateLineTaxUE` and `RecalculateHeaderTaxUE` output should include all necessary taxes to avoid wiping out previous existing taxes.

02
External Calls

- Periodically review the response time of the external calls to payment system.
- Implement both connect and socket read time to ensure external call does not wait in socket-read indefinitely.
- Long running transaction can lead to DB contention, and resource problem on JMS (MQ Server).
- Unforeseen performance issues and impact to other components.

Do's

- ❑ If Dynamic CTR feature is enabled, use `manageChargeTransaction` API and create separate authorizations for each release/shipment. If single line has multiple releases, then one CTR should be created for each release.
- ❑ Review the javadocs before implementing `processOrderPayments`, and use `RequestCollection, ExecuteCollection, RequestCollection.`
- ❑ Avoid redundant processing of orders by the payment agents. Use the `getJobs` query to verify eligible orders.

Dont's

- ❑ Do not take authorization as part of `createOrder` when Dynamic CTR Distribution is enabled.
- ❑ Do not call `processOrderPayments` as part of long transaction boundary. This API is intended for In-person scenarios e.g., carry lines.

  Note: This API cannot be used with any of the order modification APIs or any APIs that modify orders - either through events, `multiApi` calls or services.

  The `requestCollection()` API will be invoked in a new transaction boundary and with a special condition - each Charge and Authorization request created will have `UserExitStatus` set to `"ONLINE"`. When `requestCollection()` is complete, it will return to `processOrderPayments()` and execute a commit in the new transaction boundary then close it. Thus, even if an error is thrown after this point, the database will not rollback the changes made by `requestCollection()`. Javadoc →
- ❑ Do not `useUnlimitedCharges` on the payment method.

Plan and take necessary action to position y(our) solution for peak success, it is critical to *TAKE ACTION NOW!*

↓

01
Database Hygiene

- Mmaintaining healthy database can prevent disruption in production.
- Reduce the IBM Sterling Order Management database size with entity level compression and enhanced purges.

More details →

02
Slow Transactions

- Long running transaction can lead to DB contention, and resource problem on JMS (MQ Server).
- Limits your ability to (auto) scale based on KPIs.
- Achieve scalability with smaller lightweight transactions boundaries.

Y(our) actions

- ❑ Ensure all necessary **purges** are running to maintain healthy & lightweight database, which in-turn minimizes performance issues.
- ❑ **Disable** unnecessary transaction **audits** (Order Audits, General Audits, etc.)
- ❑ Implement **entity level database compression** for custom and OOB CLOB column types.
- ❑ Leverage Self-Service database dashboards; continuously review **top tables optimization opportunities.**
- ❑ Review and **consolidate agent and integration workload** to optimize resource allocation.
- ❑ Select **correct** JVM profile **(*OMoC NextGen)** based on analysis from verbose GC logs or your -Xmx/-Xms parameters (Legacy/On-Premise)
- ❑ Review and optimize long running transactions; average async transaction response time should be below 1 seconds.
- ❑ Review common configuration (RTAM, HotSku, JMS), based on the prior recommendations.
- ❑ Reduce message payload by optimizing API, event templates, pull only required data.
  - ❑ Restrict output by setting the `MaximumRecords` in the inputs to any list API calls; use pagination (link)
- ❑ Review reference data cache; catch redundancy by analyzing application logs for frequent cache drops (i.e., ‘Clearing cache’). Frequent refreshes of MCF reference data cache can lead to performance issues. (link)
- ❑ Review errors and ensure errors are addressed to avoid noise, if not address it could mislead during crunch time, also it could cost performance during elevated load, impacts our ability to monitor the system effectively.

Position for Peak Success

To best position for success on the OMS platform, it is important to understand how your application handles various scenarios known to challenges performance or stability. Testing in pre-production with data/workloads representative of production enables ability identify and address issues without impact to production business and operations.

1. Resource/Hardware **sizing** based on segment profile, but is validated as OUTCOME of performance testing, not a replacement for it
2. **Database** is common bottleneck, not due to capacity, but untuned queries, missing indexes, competing processes, unqualified end-user searches
3. Underlying config **data** has significant impact on performance, including database query execution plans, inventory sourcing rule evaluation
4. Accumulation of **transactional data** over long periods of time (and failure to purge as possible), may degraded query performance
5. **Item distribution** and commonality must reflect realistic peak load; high-demand / hot items (free-gift) may significantly impact concurrent processes
6. Composition of a custom service (**Service definition framework**) can lead to inefficient execution or potential lock contention, reducing throughput
7. Understanding **queueing/**de-queueing rates to align with business SLA / expectation (ie. create order, confirm shipment); SI needs to know when there is an issue to intervene / troubleshoot (ie. particular queue depth)
8. Agent/integration server throughput must be sufficient, but remain below **max resource allocation**; varies on number of instances, server profile
9. Important to understand / validate impact to upstream application (eComm) if specific synchronous calls into OMS slow or become unavailable

Real Scenarios *(Real impact…)*

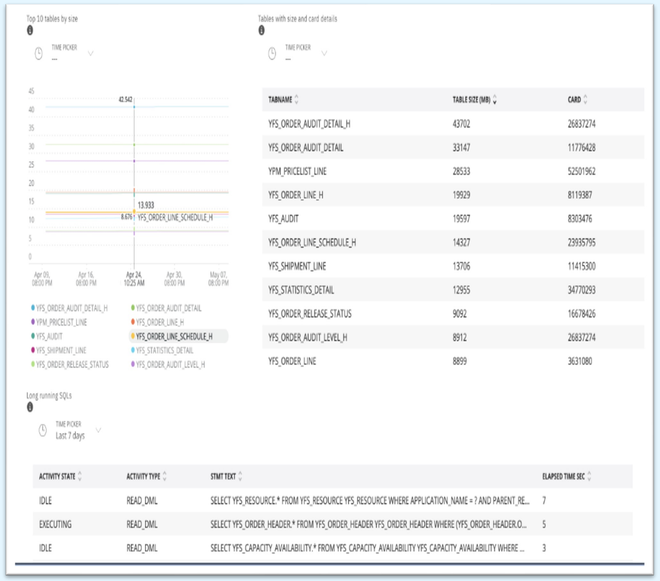
- In recent years more stores enabled for BOPIS/SFS which made DG significantly larger, which led to more time for synchronous inventory availability calls; similar scenarios where client had to split nodes in DG to improve throughput
- Rapid ramp up of **in-store associates** led to several unqualified searches in Store and Call Center apps which caused significant overall degradation
- multiAPI made 8 successive API calls led to poor response, needed to be refactored to **use asynchronous** requests (via MQ to drop message on queue)
- **Custom service call** to getOrderList API was missing OHK in input, each invocation caused fetch of 5K records which led to a crash, had to limit records
- Needed to **throttle down** instances/threads of agent to reduce concurrency contention issues (Create/Schedule/Release) and optimize throughput
- Gradual **memory leak** led to out-of-memory condition after a couple days; similarly, untuned heap led to excessive GC overhead, high CPU, slowness
- Daily manual processing of orders via java client against single JVM bypassed load balancer and overwhelmed JVM to OOM/crash
- Upstream eComm site was unable to gracefully handle a short period of unavailability from backend OMS and took hours to recover
- Unintentionally carrying capacity for high volume node during the peak. (Example: Popup /Temporary fulfilment warehouse)
- Avoid changes to DG in IV during peak time

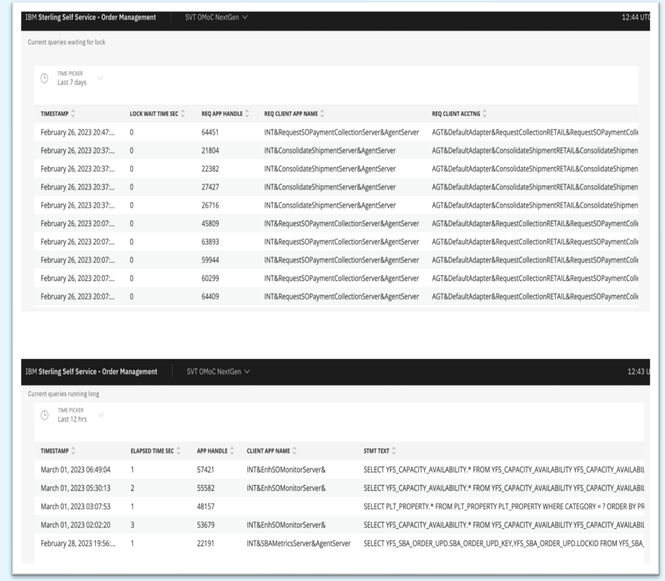
Self Service Tool Dashboards

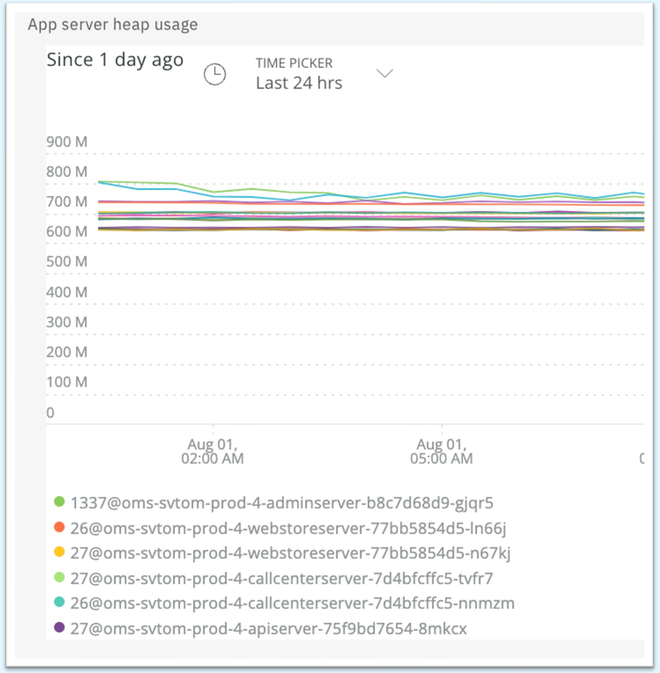
Leverage Self Service tool dashboards to proactively monitor the environments and take ongoing house keeping actions to keep it in a healthy state

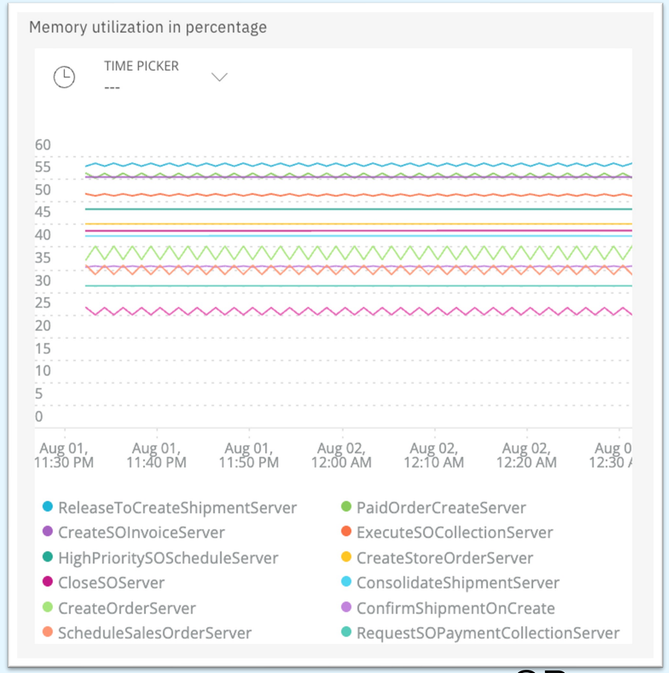
The following types of dashboards are available Read more → :

- Service Resource Utilization
- Agent and Integration Server Performance
- Database metrics
- Application Server Performance
- API Performance
- Service Performance
- Business Performance
- JMS metrics
- JVM metrics





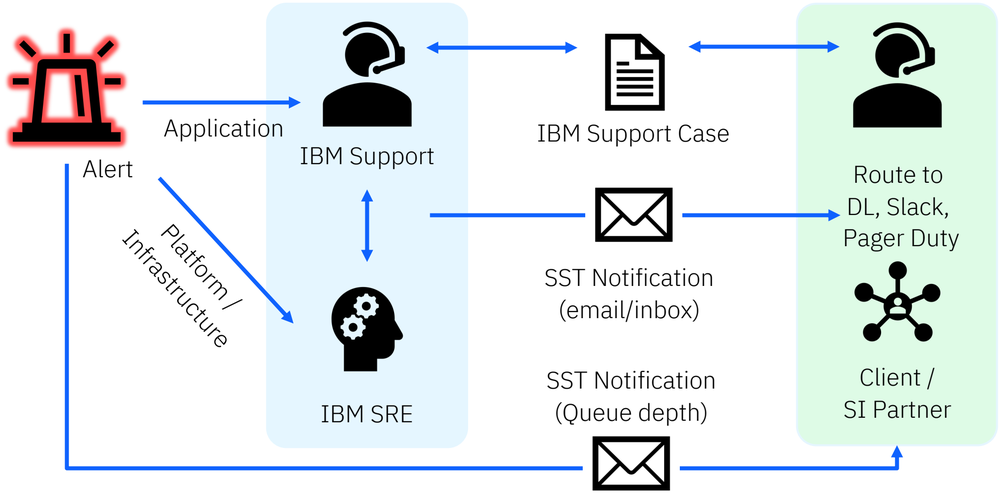




Robust Monitoring & Runbooks

The OMS SaaS Proactive Support & Notification Model aims to quickly detect and mitigate issues before they become impactful

IBM performs proactive monitoring 24x7 to assess the health of your production infrastructure and application



If a potentially impactful condition is detected, IBM Support will proactively notify you, and inform if your action is required to mitigate.

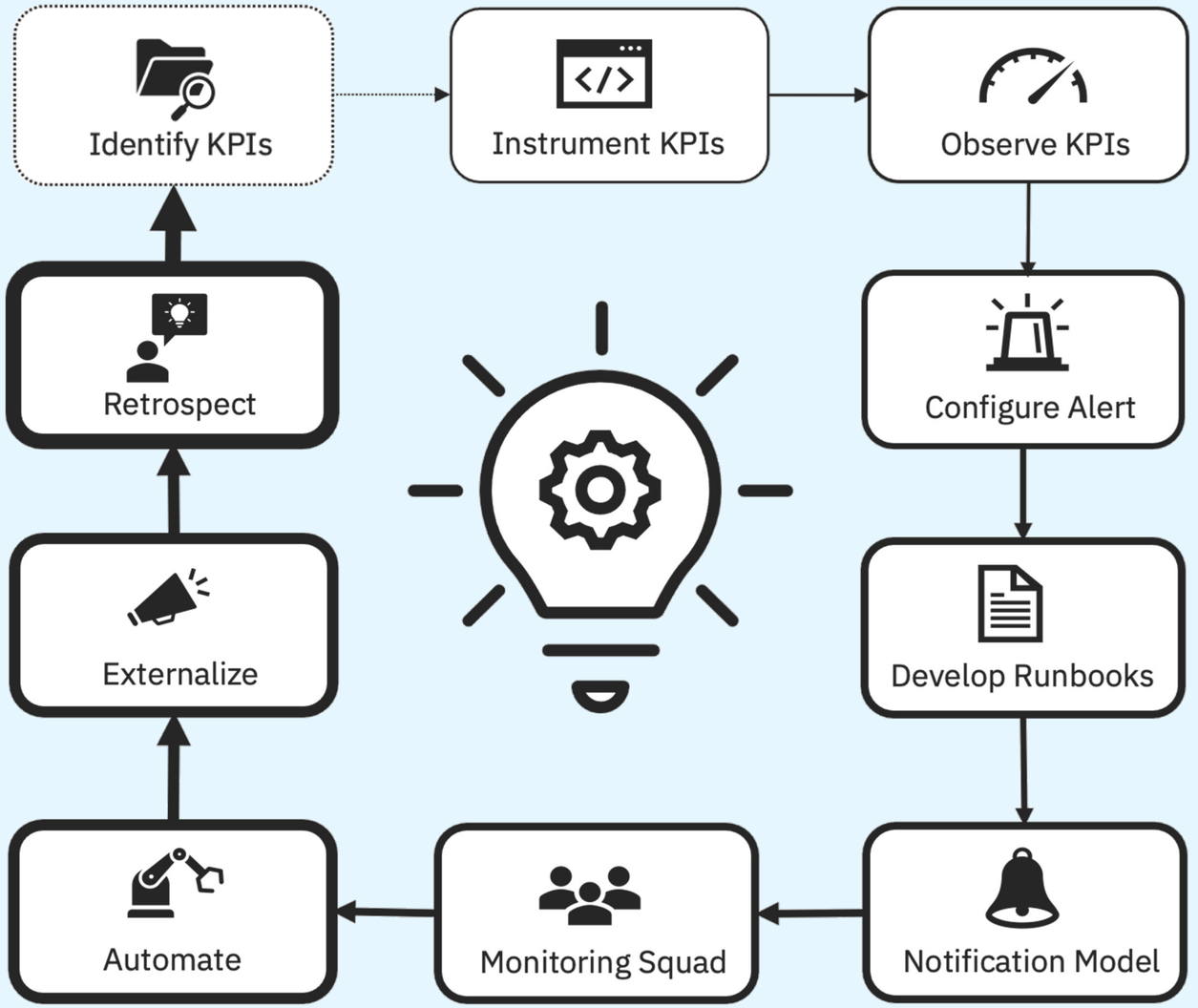
- EMAIL (✉) for issue with **Multi-tenant shared infrastructure**
- SUPPORT CASE (📄) via IBM Support Portal for **client-specific component**

Important Reminders!

- Self-manage your distribution lists via **Self-Serve Tool ,** add any necessary distribution lists or programmatic IDs (ie. PagerDuty address) as Stakeholder role
- Maintain Support case access and visibility via **IBM Support Portal!**
- Respond and take action on any **Proactive Case** opened by IBM Support!

SaaS

Continuous Improvement



IBM Support and SRE teams are focused on **driving continuous improvement** of alerts and internal runbooks based on lessons learned across infrastructure, database, app servers, agent/integration servers, error conditions

SP

OMoC Production Alert Handling

IBM are continuously improving monitoring, alerting and runbooks to allow quick handling of production issues:

1. **Proactive case** will be opened by IBM Support to inform client/SI of triggered alert, and that investigation is underway
2. IBM capture **diagnostics**, review, determine source of alert
3. IBM act to, **mitigate,** if possible, inform / get consent from client/SI as needed (such as restarting an agent/integration server)
4. In event client/partner need to **take action**, the proactive case will be used to convey this information

SaaS

MQ Connectivity issue

(JMS Metrics Dashboard)

(Max connections, JMS Transaction Failures)

- ✓ Critical MQ connection issue
- ✓ Excessive MQ Connection Reset
- ✓ MQ - Invalid Message → 2 in 10 min.

Database Connectivity Issue

(Database Metrics Dashboard)

- ✓ Excessive Database Query Timeouts (YFC0006)
- ✓ Critical DB connection issue (YFC0003 - DB Error)
- ✓ Max connections, DB Transaction Failures)

Network/Connectivity Failures

- ✓ Connectivity Issue (ConnectException, SocketException)
- ✓ Data Extract Failure - YFS: SFTP server not reachable

Application Server

(App Server Performance & JVM Metrics Dashboard)

(Real Time Failure / Sync Calls)

- ✓ GC (Global) Overhead (High) → 5% for 10 min.
- ✓ Heap memory usage (High) → 80% for 15 min.
- ✓ Server Hung/Unresponsive → 90% threads used for 5 min.
- ✓ Excessive Errors by JVM - critical
- ✓ Server Startup Failure (YIC10004) – Cache Initialization
- ✓ Excessive REST:HTTP 401
- ✓ Process DOWN (Health Check , Missing POD)
- ✓ Response time alerts for web requests

MQ Server

(JMS Metrics Dashboard)

- ✓ MQ Listener Down: No listener running for (OM_QMGR)
- ✓ MQ Server Down
- ✓ Generic Queue depth alert 50%
- ✓ MQ failover alert

Stale Agent/Integration Server

(Agent, Integration Server Performance & JVM Metrics Dashboard)

(Stale Agents)

- ✓ Heap memory usage (High) → 80% for 60 min.
- ✓ GC (Global) Overhead (High) → 5% for 10 min.
- ✓ Custom Queue Depth Alert
- ✓ Agent & Integration Process DOWN

Stale/Stuck Database Query

(Database Metrics Dashboard)

- ✓ DB: Lock-Wait
- ✓ DB: Long Running Query
- ✓ DB: Not enough storage is available, SQLCODE=-973 DB: Too many open statements, SQLCODE=-805

OOB Integration

(Application Server Performance Dashboard)

- ✓ IV Integration Failures (Connectivity issue & Error Response)
- ✓ SIM Integration Failures (Connectivity issue & Error Response)

Low Severity alerts

(Error count widget – App, Agent & Int Performance Dashboard)

(Based on Exceptions)

- ✓ Data Extract Failures
- ✓ JMS: Configuration errors
- ✓ DB: Inserted Column Data > Column Size, Error: YDB92_001 (10+)
- ✓ DB: Failed Update due to concurrent modification, Error: YFC0009 (100+)
- ✓ Search index size alerts

Database Server

(Database Metrics Dashboard)

- ✓ Database CPU, Disk Utilization
- ✓ Host is not responding for 5 minutes.
- ✓ Transaction logs size
- ✓ HADR/TSA connection
- ✓ DB Read/Write/Disk Utilization

Other VM Host

(Server Resource Utilization)

- ✓ Local , NFS Disk Utilization
- ✓ VM (host) is not responding
- ✓ CPU, Memory , Disk Usage
- ✓ CPU Steal
- ✓ Cluster Health

Availability Check

- ✓ Order Management Components
- ✓ Sterling Intelligent Promising APIs
- ✓ Self-Serve Tooling
- ✓ Order Hub
- ✓ Store Inventory Management APIs

Infrastructure

Synthetics

SP

Self Service Features

Several key features within the Self-Service tool help keep you informed on critical information to help ensure smooth operation

The **Event Calendar** helps you plan and manage major upgrade and maintenance events for your environments

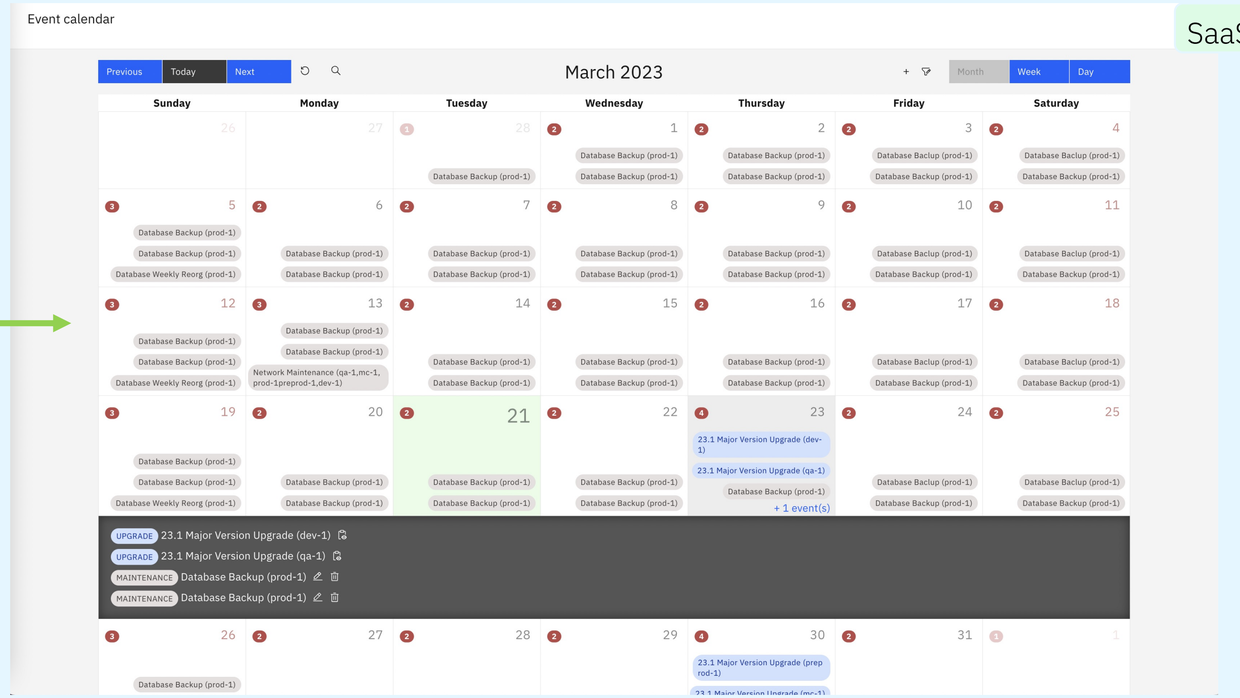
> **Important:**
> - You must raise the reschedule request minimum three days in advance. IBM provides you a window of 14 days to pick the reschedule date from. Ensure that you pick a date minimum three days from the current day. Remember that you cannot choose a date that is close to end of the window, because the lead time which is of three days will fall outside of the window.
> - Confirmation of your rescheduling request is subjected to acceptance and approval by IBM Site Reliability Engineering (SRE).

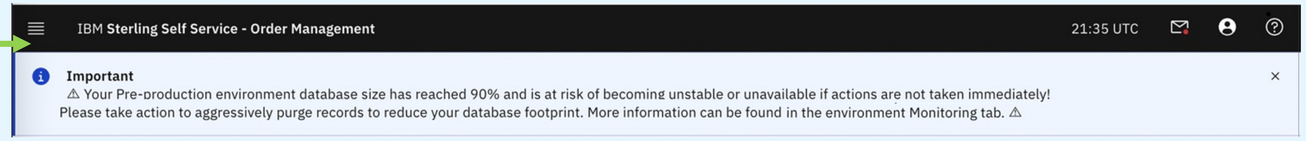
A **Banner** of one or more notifications help the System Implementer (SI) when they log into SST showing important updates or messages.

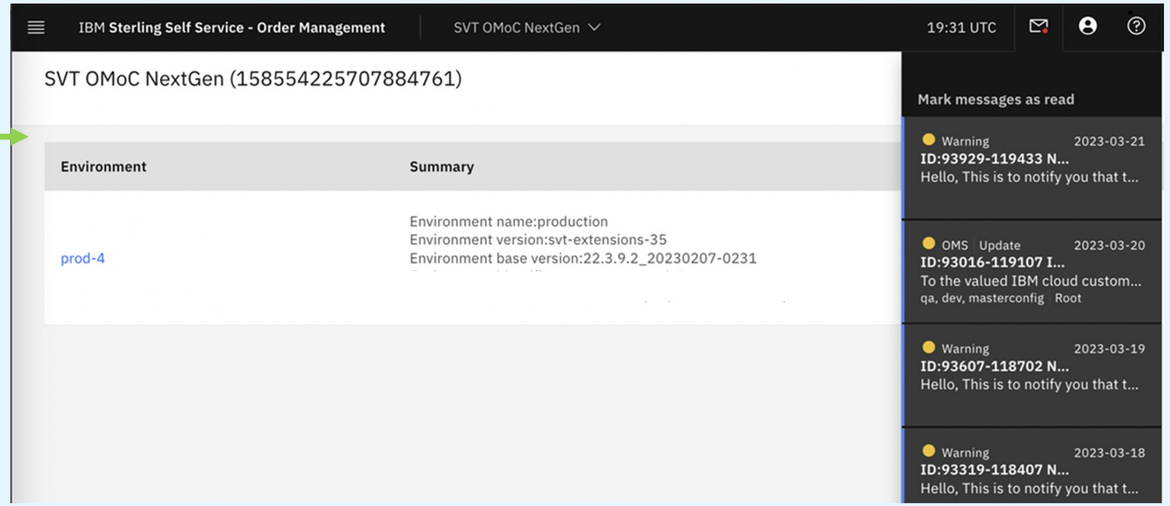
Users with the Organization Administrator and Developer roles can view **messages** and use the **Inbox notification** feature to preview messages, about application alerts, proactive cases opened, upcoming maintenance, and upgrade events of their environments. For any type of event update, you will receive both an email and Inbox message.

The preview panel displays following information:

- **Severity**: The level of importance of the message - High (🔴), Medium (🟡), and Low (🔵)
- **Product**: The name of the product the message is about, such as Order Management Software (OMS), Inventory Visibility (IV), Sterling Inteligent Promising (SIP), and more.
- **Message Type**: The type of the message such as General, Service Issue, Maintenance, and Update.
- **Subject**: The subject of the message.
- **Summary**: A brief of the message.







SP

How to Succeed

Plan

- ☑ Retrospective
- ☑ Latest product levels
- ☑ Detailed projections
- ☑ Catch prior webcasts
- ☑ Engage help as needed

Prepare

- ☑ Align to IBM schedule
- ☑ Representative testing
- ☑ Proactive housekeeping
- ☑ Clean up the noise
- ☑ Track risks

Execute

- ☑ Clear runbooks, RACI
- ☑ Quickly detect issues
- ☑ Throttle as necessary
- ☑ Quick mitigation

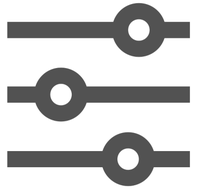
MC

Enhanced Event Readiness Offering

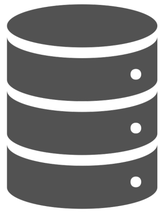
A proactive engagement leveraging a methodical approach to provide targeted, prescriptive guidance toward stability and success on IBM Order Management

Support Backlog Reviews, Prioritization

Best Practice Enablement, Consultation



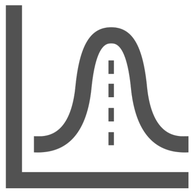
Application Configuration Audit



Database workload review

Application workload review

Production performance review



Peak Projection and Capacity validation

Peak Day Readiness Checklist

SWAT Peak Day Standby

IBM Event Readiness Team

OMS Performance Experts apply years of proactive preparation and support of worldwide clients for successful go-lives and peak events

- ✓ Identify, mitigate potential risks
- ✓ Align to proven best practices
- ✓ Peak day mitigation techniques

Support Experience Team prioritize Support workload, augment communication and escalation to help avoid blockers

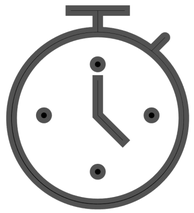
Expert Labs (optional) available to perform comprehensive reviews and health checks

Event Readiness *is modelled as 80-120 hour engagement over 4 months – partnering as you prepare, test, and execute go-live or peak event;* ***For November peak, our Engagement must begin no later than September 1, ensuring ample time to proactively review, implement, and validate recommendations***

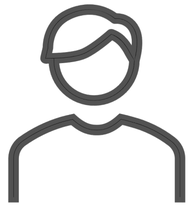
IBM Advanced Support **Offering**

An enhanced support experience on top of your active IBM support subscription, providing prioritized case handling and shorter response time objectives

NEW in 2Q23!



Enhanced Initial Response SLOs including <30 min for Severity 1



Named IBM Advanced Support Focal (ASF) within business hours

Priority access to Senior Technical Support for accelerated issue resolution

Monitor, manage, escalate critical cases and provide period case status reports and trends

Cases handled with higher ongoing prioritization within Severity level

7x24 coverage for mutually agreed Sev-2 requiring urgent attention

Are *you* ready?

Technical Best Practices

Payment Collection Agent GetJobs Query - Reference

```
select yfs_order_header.order_header_key, yfs_order_header.lockid from omdb.yfs_order_header yfs_order_header
where (yfs_order_header.payment_status in
('AWAIT_PAY_INFO','AWAIT_AUTH','REQUESTED_AUTH','REQUEST_CHARGE','AUTHORIZED','INVOICED','PAID',
'RELEASE_HOLD', 'FAILED_AUTH', 'FAILED_CHARGE', 'VERFIFY', 'FAILED')) and
yfs_order_header.authorization_expiration_date <=sysdate and yfs_order_header.draft_order_flag='N' and
Yfs_order_header.enterprise_key IN (select distinct enterprise_key from omdb.YFS_ORDER_HEADER) and
yfs_order_header.document_type='0001' and not exists (Select 1 from omdb.yfs_order_hold_type
yfs_order_hold_type where yfs_order_hold_type.order_header_key= yfs_order_header.order_header_key and
(yfs_order_hold_type.hold_type in ( SELECT  DISTINCT HOLD_TYPE FROM omdb.YFS_HOLD_TYPE  WHERE  DOCUMENT_TYPE =
'0001' AND  ORGANIZATION_CODE = 'DEFAULT'  AND  BASE_PROCESS_TYPE_KEY =
'ORDER_FULFILLMENT'  AND  (  HOLD_TYPE_KEY IN ( SELECT  HOLD_TYPE_KEY FROM omdb.YFS_HOLD_TYPE_TRAN
WHERE  TRANSACTION_ID = 'PAYMENT_COLLECTION'  AND  PURPOSE = 'PREVENT')  ))  and
yfs_order_hold_type.status<'1300')) with ur;
```

Payment Execution Agent Query - Reference

```
SELECT ORDER_HEADER_KEY, count(*) AS COUNT from omdb.YFS_CHARGE_TRANSACTION WHERE ORDER_HEADER_KEY in ( SELECT
DISTINCT YFS_CHARGE_TRANSACTION.ORDER_HEADER_KEY FROM omdb.YFS_CHARGE_TRANSACTION YFS_CHARGE_TRANSACTION ,
omdb.YFS_ORDER_HEADER YFS_ORDER_HEADER  WHERE  YFS_CHARGE_TRANSACTION.STATUS =
'OPEN'  AND  (  YFS_CHARGE_TRANSACTION.CHARGE_TYPE IN ( 'AUTHORIZATION' ,  'CHARGE' )  )  AND
YFS_ORDER_HEADER.ORDER_HEADER_KEY =
YFS_CHARGE_TRANSACTION.ORDER_HEADER_KEY  AND  YFS_ORDER_HEADER.PAYMENT_STATUS <>
'HOLD'  AND  YFS_ORDER_HEADER.DOCUMENT_TYPE = '0001'  AND  YFS_ORDER_HEADER.DRAFT_ORDER_FLAG = 'N'
AND  YFS_ORDER_HEADER.ENTERPRISE_KEY = 'DEFAULT' AND  NOT EXISTS ( SELECT '1' FROM omdb.YFS_ORDER_HOLD_TYPE
yfs_order_hold_type WHERE yfs_order_hold_type.order_header_key=
yfs_order_header.order_header_key  AND  (  YFS_ORDER_HOLD_TYPE.HOLD_TYPE IN ( SELECT  HOLD_TYPE FROM
omdb.YFS_HOLD_TYPE  WHERE  DOCUMENT_TYPE = '0001' AND  ORGANIZATION_CODE =
'DEFAULT'  AND  BASE_PROCESS_TYPE_KEY = 'ORDER_FULFILLMENT'  AND  (  HOLD_TYPE_KEY IN ( SELECT  HOLD_TYPE_KEY
FROM omdb.YFS_HOLD_TYPE_TRAN WHERE  TRANSACTION_ID = 'PAYMENT_EXECUTION'  AND  PURPOSE =
'PREVENT')  ))  )  AND  YFS_ORDER_HOLD_TYPE.STATUS <
'1300'  )  AND  (  (  YFS_CHARGE_TRANSACTION.USER_EXIT_STATUS <> 'ONLINE'  ) OR
(  YFS_CHARGE_TRANSACTION.CREATETS <= sysdate  )))  GROUP BY ORDER_HEADER_KEY having count(*) > 2 ORDER BY
COUNT DESC with ur;
```