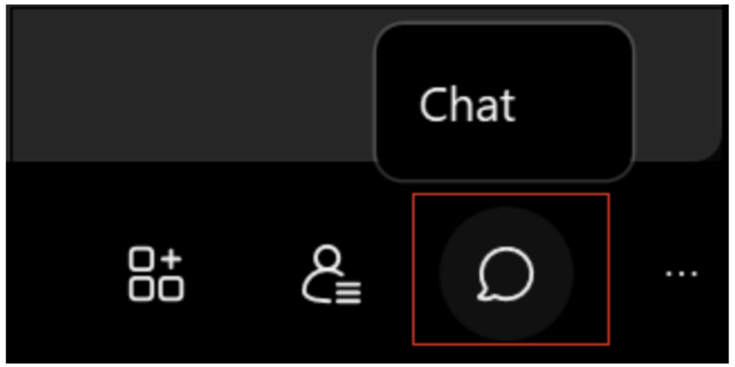
IBM Sterling Order Management

Event Readiness 2024

Best Practices

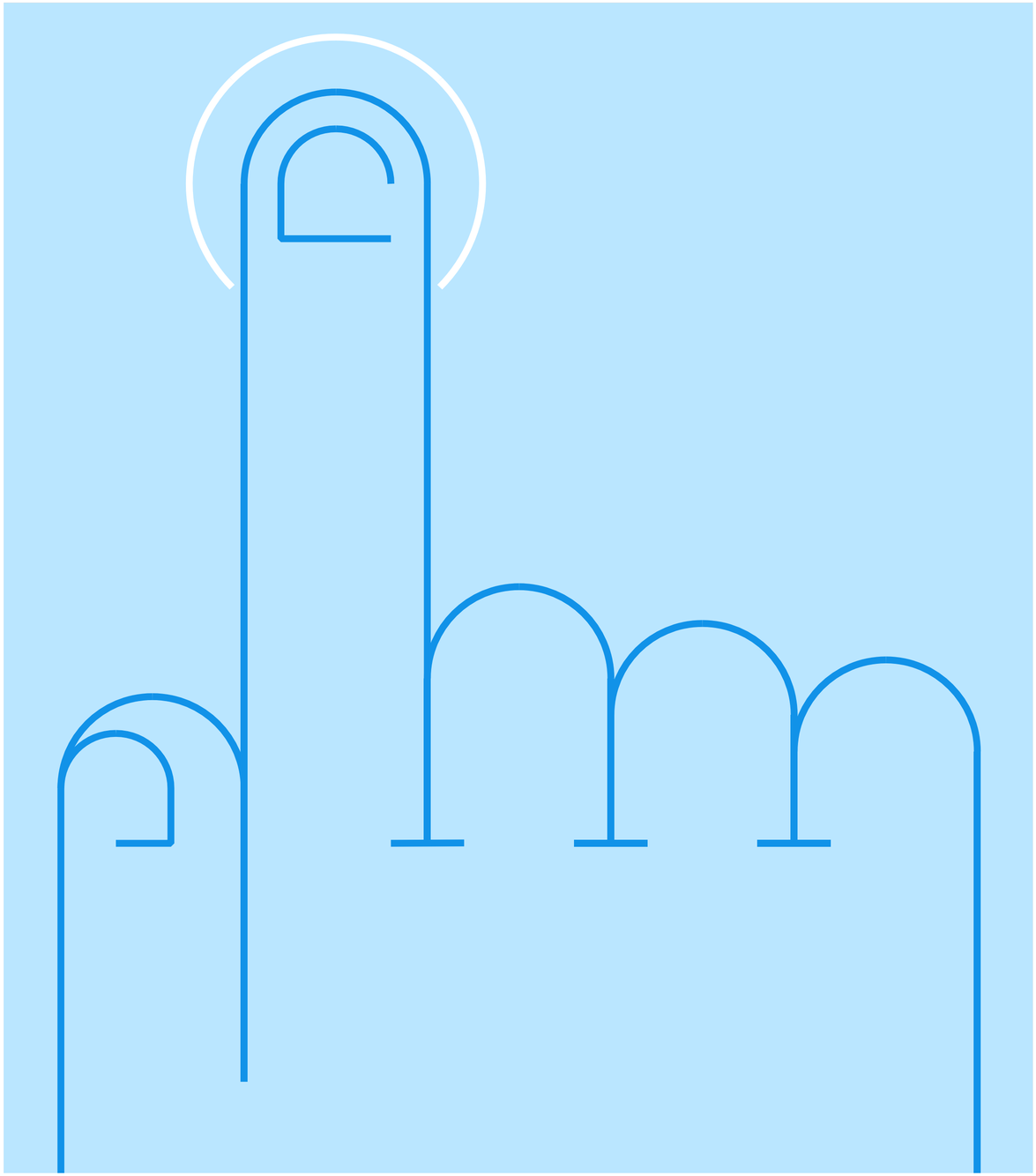
Have a Question(s)?

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 Event Readiness Team
... and today's speakers

Chris Burgess
Manager,
WW Support Experience Team

Mike Callaghan
Program Director,
WW Supply Chain Support

Shoeb Bihari
Executive IT Specialist | SRE Architect | SWAT,
Order Management Support

Senthil Ponnusamy
Technical Lead / SRE Advisor,
Order Management Support

Vishal Arora
Technical lead, Development,
Order Management Solutions

Jitendra Buge
Technical lead,
Order Management Support

Paresh Vinaykya
Executive Technical Account Manager, Expertise Connect, IBM Expert Labs

Mohamed Jawahar
Senior Solution Architect,
IBM Software Support

Damini Tacouri
Advisory Support Engineer,
Order Management Support

Madhavi Pilaka
Technical Support Engineer,
Order Management Support

Agenda

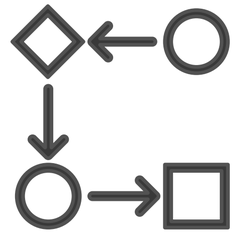
- ✓ Our Journey to peak success
- ✓ CDT best practices
- ✓ Application and Order flow performance best practices
- ✓ Customization best practices
- ✓ Large Order handling in OMS and SIP
- ✓ Store/Call Center UI performance
- ✓ SIP best practices
- ✓ Payments best practices
- ✓ JMS and Database Best practices

What are *your* plans in 2024 for IBM Order Management?

First Go-live

Stack upgrades

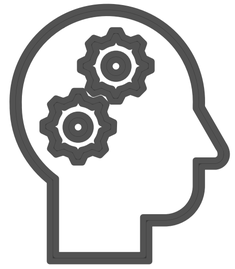
Container deployment



OMoC migration

On-Prem to Cloud

Expansion



Sterling Intelligent Promising

Call Center, Store, Order Hub

Holiday Peak Season

IBM OMS Holiday Readiness

Our Mission Statement

✓2023-03 | Journey to Peak Success

✓2023-05 | Payment Integration

✓2023-08 | Best Practices

✓2023-09 | Panel Discussion

✓2023-10 | Peak Day Preparedness

✓2024-04| 2024 Kickoff Retrospect

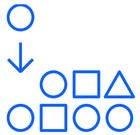
✓2024-06 | Self Service Monitoring Dashboards Demo

In case you missed it…

ibm.biz/IBM-OMS-HolidayReadiness

Stable Platform

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

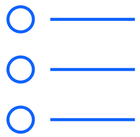


Best Practices

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



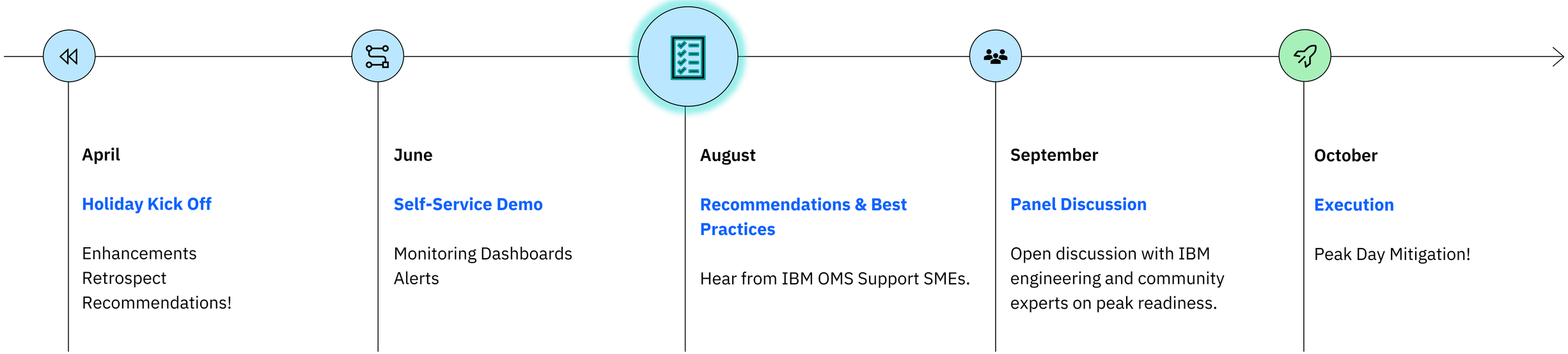
Prescriptive Guidance

Deeper partnership with specific clients in need of direct analysis and prescriptive guidance via Advanced Support and Expertise Connect

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!



April

Holiday Kick Off

Enhancements
Retrospect
Recommendations!

June

Self-Service Demo

Monitoring Dashboards
Alerts

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!

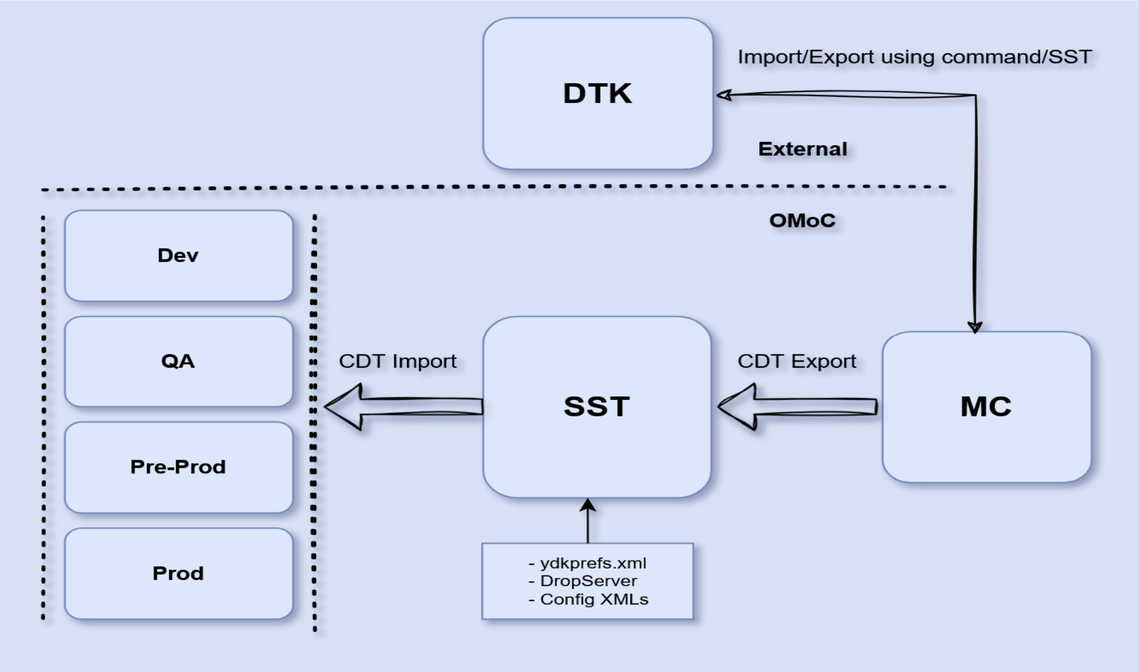
Configuration Deployment Tool (CDT)

These best practices will help you utilize CDT feature efficiently and effectively without any failures.

Supported capabilities:

–Export from databases to XML files or Import from XML files to databases
–Ignore or append certain tables
–Transform certain data
–Ignore selective records

Supported Flow (OMoC):



Recommendations & Best Practices

– `AppendOnly` and **Ignore mode** usage with extra caution
– Hierarchy and relationship among different tables for CDT
– Handling `YFS_PERSON_INFO` records
– CDT is primarily designed for exporting and importing configuration tables only
– To export transactional data use Data Extract process
– `PLT_PROPERTY` and `PLT_PROPERTY_METADATA` must not be added to `AppendOnly` in the `ydkprefs.xml` file.
– Source and target versions must be same, or the source version is less than the target runtime version.
– By default, the following tables are ignored from CDT:
  ✓ `YFS_ZIP_CODE`
  ✓ `YFS_REGION`
  ✓ `YFS_REGION_DETAIL`
  `-> Use yfs.cdt.ignoretable.<table name in caps>=N` to include the table
– Export `SourceDatabase` as `SYSTEMDB` and `TargetDatabase` as `DEFAULTXMLDB`
– Import `SourceDatabase` as `DEFAULTXMLDB` and `TargetDatabase` as `SYSTEMDB`
– Explore CDT as an alternative option for unavoidable manual DB updates with caution
– Viewing logs for CDT:
  ✓ Log displays details for all stages of the CDT process.
  ✓ View these logs in Graylog
– CDT Log analysis guidance

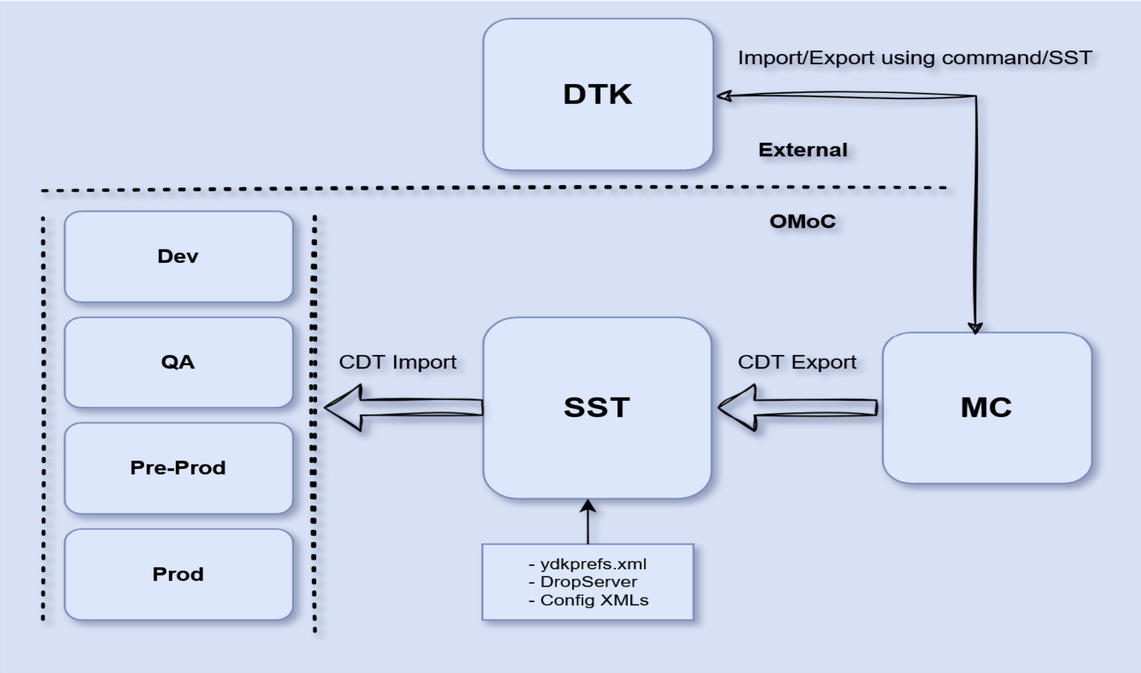
Configuration Deployment Tool (CDT)

These best practices will help you utilize CDT feature efficiently and effectively without any failures.

Supported capabilities:

–Export from databases to XML files or Import from XML files to databases
–Ignore or append certain tables
–Transform certain data
–Ignore selective records

Supported Flow (OMoC):



Common Issues

– OutOfMemoryError Exception

- ✓ Increate Heap Size (Xmx)
- ✓ Exclude / Ignore Large Tables
- ✓ Split the list of tables

– Slow running process

- ✓ Amount of data loaded through CDT
- ✓ Sync issue with SST backend system
- ✓ Restart backend process is stuck in 2.0

– Tables are not ignored

- ✓ Configuration issue in ydkprefs.xml
- ✓ Child tables are ignored if the parent table is in the ignored list

– CDT Failure due to version miss-match
– `YFS_PERSON_INFO` records require special handling in CDT
– CDT process fails with filename too long error

Best Practices

Together we achieve Y(our) objective of stability and success on our IBM solutions during most critical business events – through readiness of application, platform, and operations

Follow the Guide

Proven Best Practices:

Apply Recommended Configurations

Application

Use Optimal API and optimize output template

Fine tune entity cache, disable redundant.

Use HOTSKU & OLA, Capacity Optimization Settings

Set API isolation level based on the use case

Use pagination for getter APIs

…

Integration

Use OOB Integration as much as possible.

Have adequate retry

Use Connection Pool/Caching

Have timeout's for up/down stream synchronous calls

Apply Sterling Intelligent Promising (IV) Best Practices

…

JMS

Use JMS session pooling

Optimize message size

Use non-persistent queues for only agent workload

Enable multi-threaded PUT's

Avoid message selectors

Enable service retry (transaction reprocessing)

Enable queue depth / message delay alerts.

…

Database

Maintain transactional tables, Purge (set appropriate retentions)

Execute non-essential purges during off-peak hours

Apply indices for custom entity extension

Minimize contention, maximize concurrency

Optimize long-running or expensive queries

Enable `stmt_conc` (LITERALS)* for better performance on DB2

…

Platform

OMoC Legacy

OMoC NextGen

Certified Containers

Performance Testing

1 Plan
Establish and quantify goals and constraints
What business wants?
Identify KPI's & NFR's

2 Prepare
Populate the application with realistic data. Use a phased 'stair-case' model

3 Execute
Simulate workload

4 Tune
Scale workload to meet NFRs

Application Performance

API Performance

- Network / Client Logic
- API Input
- Session Management
- Debug/Logging Statements
- Business Logic
- Database Operations
- Call to External System
  - Time to Establish Connection
  - Payload
  - Connect/Socket Timeout
- Change/On-Success Events
- JMS Operations
  - Message Size
- Output Template

Best Practices

- Validate API input before calling API, ensure required or filtrable attributes are passed (avoid open ended SQL).
- Limit the input size
- Tune `servlet.token.absolute.timeout` properties to prevent YFS_USER_ ACTIVITY locking under heavy load.
- Select appropriate `SelectMethod` method for getter API's
  - Possible values are `NO_LOCK, NO_WAIT, WAIT`, etc
  - `QueryTimeout="3" TimeoutLockedUpdates="Y"`
- Keep transaction boundary small when using update/modify API, this will ensure DB object (row) is locked for minimum duration.
- Use appropriate connect and read timeout for external calls, preferably less than 5 seconds, and make use of connection pool (cached/persistent connection, keep-alive).
- Use optimal API output template to limit unnecessary data reads.
- Remove always on `DEBUG` or `SystemOut` statements
- Eliminate frequent `SELECT` by enabling entity cache.
- Disable redundant cache (always miss or frequently evicted)
- Use pagination (`getPage`) for getter APIs.
- Use Timeout properties for DB calls, `yfs.agentserver.queryTimeout, yfs.ui.queryTimeout`

Application Performance

API Performance

- Business Logic (promising)
  - Sourcing Optimization
  - Inventory Update (HOTSku)
  - Capacity Update
  - User Exits / Customization

Best Practices

- Enable HOTSku, and OLA configuration.
- Use Capacity Cache
  - Enable node locking / threshold properties based on the business use case.
  - Reduce the lock contention in the YFS_RES_POOL_CAPCTY_CONSMPTN table by enabling yfs.capacity.useMassAdjustCapacityDriver & yfs.persitCapacityAdjustments properties.
- Aggerate reservation calls to IV, this improves performance of reserveAvailableInventory API `-yfs.UseAggregatedReservationsForIV property to “Y”`
- Cache configuration data using entity caching
  - Example: `YFS_REGION, YFS_REGION_DETAIL, YFS_ATTR_ALLOWED_VALUES, YFS_ATTR_ALLOWED_VAL_LOCALE`
  - Avoid using current timestamp value as part of query predicate (this makes caching redundant due to unique value)
- Remove unwanted attribute/items from the output
  - Example: Let’s say if you are correcting inventory using `YFSGetAvailabilityCorrectionsForItemListUE`, then make sure output of the UE excludes the items with ZERO supply quantity before passing the result to OOB API.
- Enable API interrupt properties to avoid runaway transactions
- Run Inventory purge

Sourcing & Scheduling Considerations

Sourcing is the process of determining from which node or supplier a product should be shipped. A sourcing rule can be created by specifying one or more of the following key parameters:

Item Classifications or Item ID

Geographical region of the ship-to location or ship-to node

Minimum available capacity

Fulfillment type

Seller organization

Sourcing criteria

Use IBM Order Management for complex sourcing →

For each sourcing rule, you can then specify a sequence of node or distribution group to be used for sourcing the product, and performance of optimization logic depends on the complexity this setup.

What are the optimal configuration to maximize the performance?

- ❑ Number of nodes qualifying from sourcing should be as minimized by region, proximity, etc.
- ❑ If all nodes are qualifying, Capacity availability constraint should be avoided, or capacity cache should be used
- ❑ Use of Externally defined sourcing Vs Sourcing Correction
- ❑ If reservation node can be considered as final ship node, then it should be passed on order line. This avoids schedule order to consider sourcing again.
- ❑ Order profile for pre-prod should be very similar to what is expected in production with regards to DC Ship, SFS, Pick up.

Distribution Group

- Optimal Size
- Sourcing rule region hierarchy
- Priority of nodes when using multiple sequences
- Using sequence of sourcing and consideration around high availability.

Scheduling Rules

- Priority of nodes
- Distance of nodes from the ship-to location
- Date when the delivery can be made
- Number of resultant shipments
- Cost-based scheduling

Lead Times

- Whenever we review customer's scheduling rules, we typically see that lead times are set to default (30/60 days).
- How does “lead time” impact availability lookup APIs? What should be the optimal value?
- Can availability lookup be kept simple as scheduling process will handle the optimization?

Guard Rails

- Solver Optimization: There is quite a bit of optimization that happens within promising API (Cost), as such there might be certain level of performance implications.
- How to keep optimization light-weight?
- Termination properties

Hot SKU
OLA (Optimistic Lock Avoidance)

Properties to consider:

- Hot SKU properties with or without Optimistic Lock avoidance flag.
  - Initially availability assumed to be high, and lock is avoided until availability becomes low. (`yfs.hotsku.lockOnlyOnLowAvailability=Y`)
- Old Hot SKU properties
  - Availability lookup locks YFS_INVENTORY_ITEM table based on velocity and inventory availability.

Objectives:

- To process all orders in near real time and release to fulfillment node without any delay.
- To process the complete batch within a specified time window.

IBM Sterling Order Management System - Hot SKU properties tuning →

INV_INVENTORY_ITEM_LOCK

- Rolling average calculation determines if availability low
- Entries for item-node-demand type combination
- If availability becomes high, then INV_INVENTORY_ITEM_LOCK record is removed.

Tuneable Properties

- yfs.Hotsku.useAvailabilityAcrossNodes
- yfs.hotsku.useGranularLockingForItem
- yfs.hotsku.updateInventoryAfterAPIOutput
- yfs.hotsku.useGranularLockingForItem.mode

Locking PURPOSE

- The purpose for the lock record. Valid values:
  - 10 (Lock as Availability is now low)
  - 11 (Use previous Hot SKU functionality).
  - 20 Low availability when granular locking is enabled.
  - **21 to tracking 0 availability with granular locking.**

Lock during Inventory Change

- Avoid locks to YFS_INVENTORY_ITEM
  - `yfs.hotsku.lockItemOnInventoryChanges=N`
- Lock contention moved from item level down to the item-supply/item-demand level
- The `adjustInventory` API input must pass `UseHotSKUFeature=Y`

Capacity

Objectives:

- Node has known individual capacity limit based on the delivery method [Ship and Pick]
- Node cannot handle any extra orders above the allocated capacity
- Reduce the lock contention in the YFS_RES_POOL_CAPCTY_CONSMPTN table.

Usage:

API / Transaction	Capacity Availability	Capacity Updates
reserveAvailableInventory	Yes	Yes
reserveItemInventory	No	Yes
createOrder	No	No
createOrder (w/ RESERVATION)	No	Yes
scheduleOrder	Yes	Yes
releaseOrder	No	Yes
changeOrderStatus/orderSchedule	No	Yes
createShipment/changeShipment	No	Yes
confirmShipment	No	No
manageCapacityReservation	Yes	Yes

Capacity Cache

- Time-triggered agent pre calculates capacity and prepopulates YFS_CAPACITY_AVAILABILITY table
- Over allocation can be prevented using node locking properties.
  - `yfs.nodecapacity.lock`
  - `yfs.nodecapacity.threshold`

Disable Capacity

- Do not adjust capacity to infinite; use **changeResourcePool** API* to disable capacity.
- DISABLE_NODE_CAPACITY_FOR_ENT
- If capacity is set to 0, the capacity consumption record is deleted from the YFS_RES_POOL_CAPCTY_CONSMPTN table.
  - Use capacity purge to delete the 0 capacity records

Optimization Properties

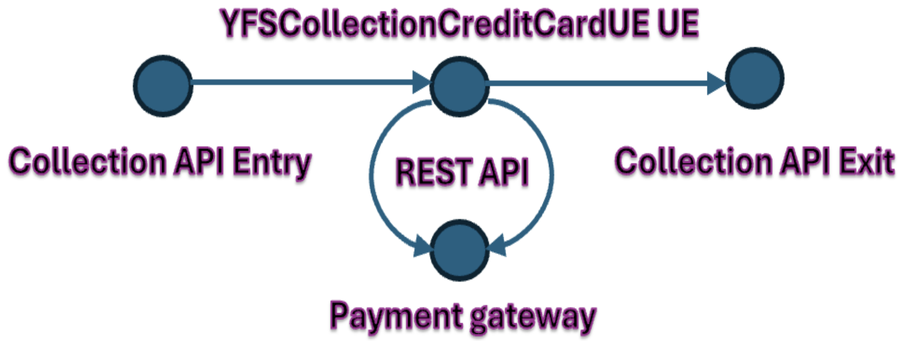
- `yfs.nodecapacity.ignoreCacheForLowAvailability`
- `yfs.capacityAvailablity.ignoreCacheForUpdateMode`
- `yfs.capacity.IgnoreCacheBelowThreshold`
- `yfs.nodecapacity.lock`
  - `yfs.nodecapacity.threshold`
- `yfs.useNodeLocaleTimeForCapacityCheck`
- `yfs.persitCapacityAdjustments`
  - `yfs.capacity.useMassAdjustCapacityDriver`

Read more →

Application Customization

User Exists (UE)

- User exits are created to enable business logic extensions to the Sterling Order Management System transactions.
- Within the Sterling Order Management System transactions, code exists to invoke user exits so that you may plugin custom logic.
- Commonly used User-Exits:
  - YFSBeforeCreateOrderUE
  - OMPGetExternalCostForOptionsUE
  - OMPProcessRoutingExternallyUE
  - YFSGetDistanceUE
  - YFSGetDeliveryLeadTimeUE
  - YFSCollectionCreditCardUE
- UE input can be either XML or Java Structs.



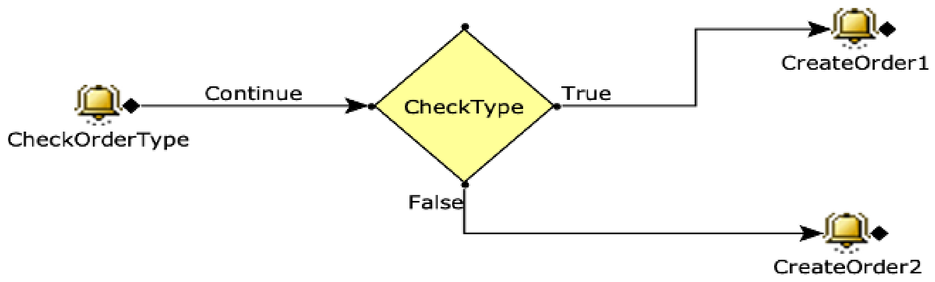
Design Considerations & Best Practices

- Data translation and transformation. (DOM parsing, Jackson parsing, etc)
- Synchronous webs/rest services calls to internal and external calls. Example: Rest service call made to payment gateway as part of execute collection API call.
- Number of sub-API calls.
  - **Example:** Calling `getOrderDetails` API of the parent order in the `YFSBeforeCreateOrderUE` of the Transfer.
- Template used for the UE input.
- Number of UE calls per API trigger.
  - **Example:** `OMPGetExternalCostForOptionsUE` UE is called once per sourcing rule sequence. The more the sequence and the more the rate of assignment rejection, the higher will be the number of calls.
- Avoid unnecessary possibility of deadlock causing web containers to be hung.

Application Customization

Events, Actions and Services

- An event is a specific occurrence in the business process; often a status change or generated alert.
- An action is a process or program that is triggered by an event. These processes and programs send alert notifications, publish data, or initiate custom services.
- Services define the business process flow between Sterling Order Management System and external systems.
- Typical implementation: Event is configured for a transaction. Event executes and action. Action triggers one or more services.



Design Considerations & Best Practices

- Number of linked actions triggered for an event.
- Number of services and type of services triggered by an action.
- Synchronous vs asynchronous calls.
- Data publication and third-party connectivity overhead.

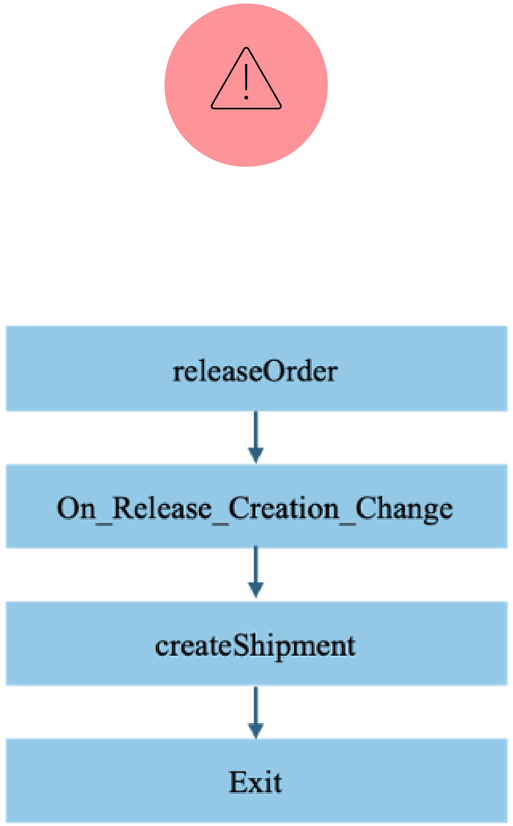
  » **Example:** Publishing to a Kafka topic on release of an order.

- Time-complexity of actions and services.

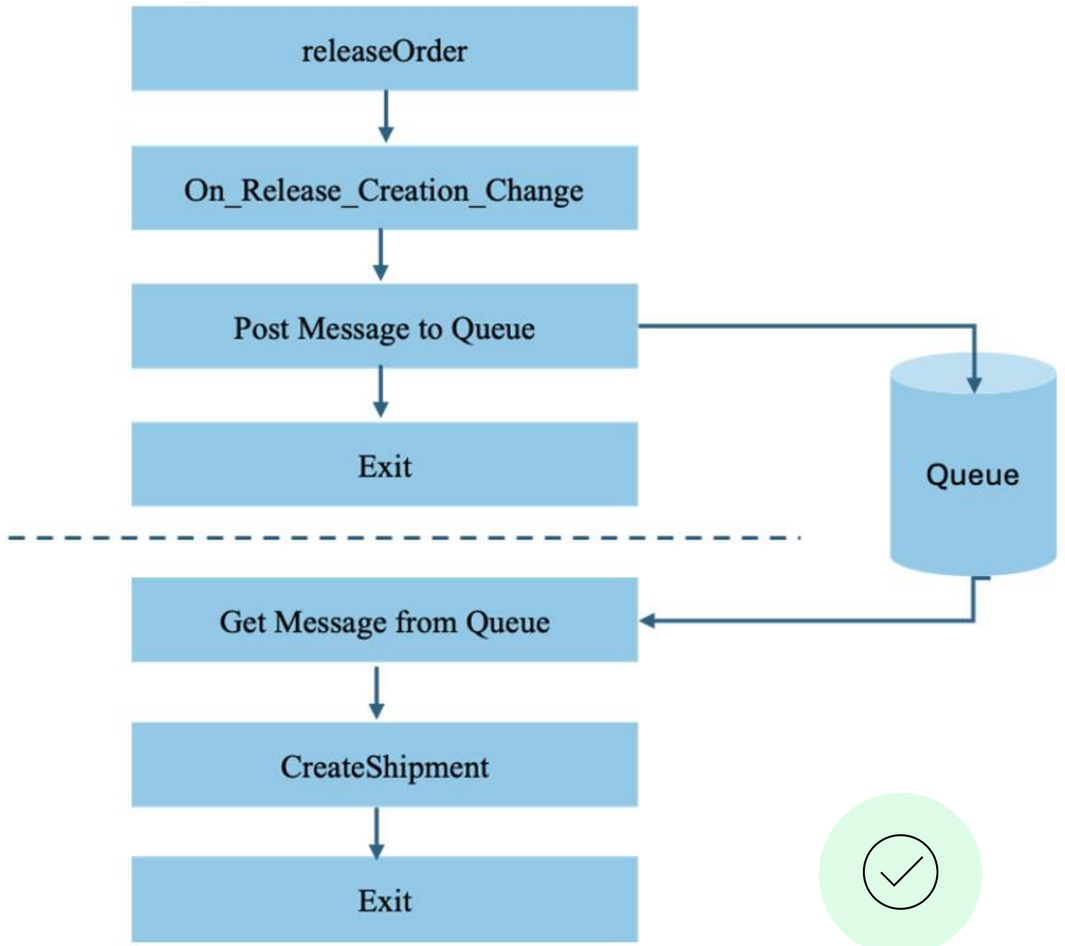
Application Customization

Synchronous vs Asynchronous Events → Sample Use Cases

- `On_Release_Creation_Change` is triggered for each successful release.
- For large orders with multiple release the number of events triggered will be high for a single `releaseOrder` API call.
- `createShipment` is called for each event.
- The `releaseOrder` is waited for successful completion of all shipments



Synchronous event

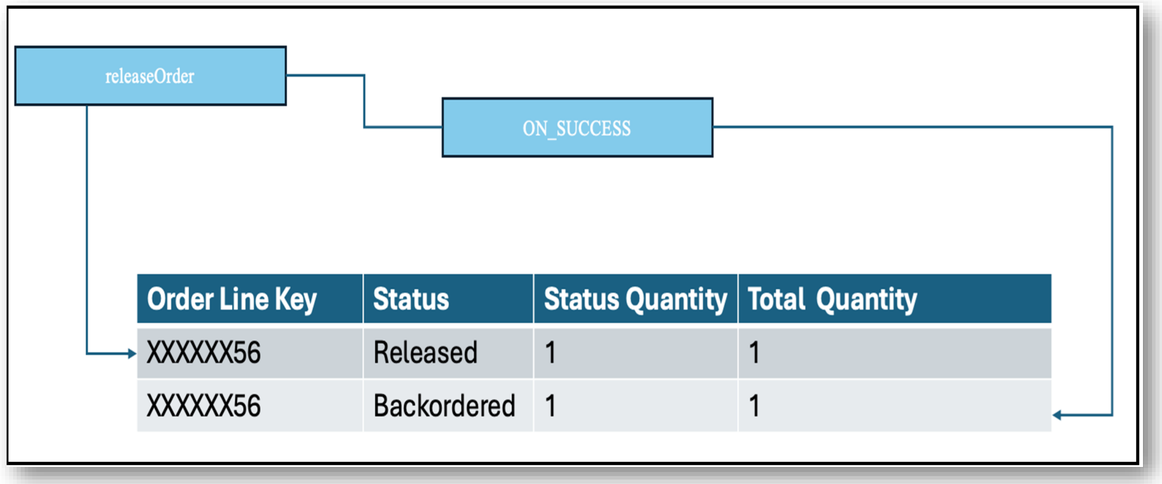


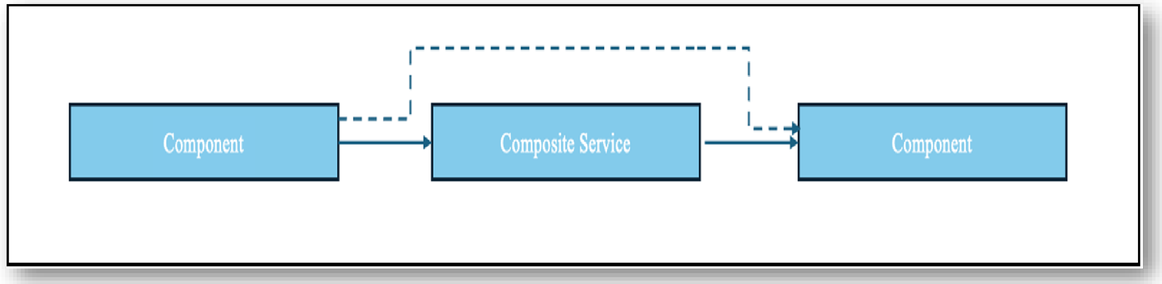
Asynchronous event

Application Customization

Coding Practices

- Avoid multiple API calls within single transaction boundary where there is potential for the APIs to run into a deadlock.
- Calling APIs within transaction boundaries where the second API updates/inserts same record as the parent API call. Example, calling `changeReleaseStatus` to change status of a released order on the success event of releaseOrder API.
- Be aware of how API wrappers are built. For example, a custom service that acts as a wrapper for `createOrder`. The wrapper parses the input data before triggering `createOrder` API.
- How exceptions are handled in the custom code. If the exception is not propagated to the parent function and the exception is suppressed, then there will be challenge in identifying the root cause of an abnormal API output.
- Exception handling in composite services





Application Customization

Entity Extension & Coding Practices (Database)

- Adding column to default table, index, non-unique index, hang-off table.
- Design table extensions and ensure right indices are in place to avoid performance impact in the event of table querying.
- Sorting information for a specific order is applicable to situations where you need to grab multiple inventory item locks within a single transaction boundary. However, you do not need to sort if you call the APIs to process single items per transaction commit.
- Purging of data from hang-off tables.

Logging / Debugs Log4j customization

- From Fix Pack 30 onwards, log4j2 is a default logging API used by the Sterling Order Management System Software.
- To make configuration changes update `log4j2_extra.xml`
- To disable programmatic configuration, add the following property to customer overrides.

  » `yfs.log4j2.omsConfig.disable=true`

- Timer logging

Wrong:

```
logger.beginTimer("Begin of findInventoryInput:::");
// Invoke findInventory()
logger.endTimer("End of findInventoryInput:::");
```

Right:

```
logger.beginTimer("findInventoryInput:::");
// Invoke findInventory()
logger.endTimer("findInventoryInput:::");
```

Application Performance

Order Flow

- Create Order
- Hold Processing
- Order Monitor
- Payment Server
- Schedule Order
- Release Order
- Create/Consolidate Shipment
- RTAM
- Purge

Best Practices

- Apply recommended JVM performance properties
- Review order and shipment monitors for redundancy, review and remove obsolete monitor rules.
  - Avoid reprocessing of order once condition evaluates to false.
    `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 flag.
- Separate out the processing of orders by one of the attributes (ex. Large order, etc) with workload separation feature.
- Apply and Tune OMoC default HOTSku and OLA configuration
- Enable Capacity cache and tune node locking properties based on business use case.
- Apply sourcing optimization (reduce DG size)
- 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 double 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

Application Performance

Large Order → GIV

Solution:

- API Templates
- Event Templates
- Asynchronous Inventory Updates

Best practices for improving order flow performance in OMS-GIV

- Extend out-of-box templates to use the minimum required elements and attributes.
  - » API(s), Event(s), UE(s), HOLD Conditions.

- If an event is not time sensitive, configure the event to be asynchronous.
- To reduce the level of inventory locking, enable Hot SKU.
- Orders that exceed 30 order lines: Update inventory asynchronously
- Inventory updates on order creation for large number of lines may lead to slowness as for each item, demand must be created/updated. This may also cause lock waits.
  - » Product recommends to use OOTB feature “Asynchronous inventory update” which helps in order creation without waiting on inventory update in same transaction boundary.

Considerations with HOTSku enabled (i.e., with below properties)

- » `yfs.yfs.hotsku.useHotSKUFeature=Y`
- » `yfs.yfs.hotsku.lockItemOnInventoryChanges=N`

- Asynchronous inventory update may be kept high as 200-500 Lines as with the above set of properties, though demand is updated in same transaction boundary but “Lock For Update” for demand updates.
  - » `LARGE_ORDER_SIZE` rule

Application Performance

Large Order → SIV Integrated Environment

Requirements:

– To support Order with large number of Order Lines in an OMS Sterling Inventory Visibility Integrated Environment.

Solution:

– The OMS IV adapter automatically break up inventory call to IV into small batches to achieve quicker response time via request parallelism

Handling orders with large number of order lines →

Use Case

– B2B customers have an enterprise order with large number of order lines

– OMS can leverage real time inventory look up via Inventory Visibility resulting in faster scheduling performance and accurate inventory picture.

» To leverage real time inventory-lookup, Inventory visibility mandates 100 lines per payload

Large Order Tenant Activation Properties

`iv_integration.largeOrder.batchMode.enable`	Enabling the feature
`iv_integration.largeOrder.nodeAvailability.batchSize`	Batch size for Node Availability
`iv_integration.largeOrder.asyncInventoryUpdateBatchSize`	Batch size for Async Demand and Supply updates
`iv_integration.largeOrder.reservation.batchSize`	Batch size for Reservation
`iv_integration.largeOrder.batchFailure.considerUnavailable`	In event of a batch failure consider that batch as unavailable or fail the API

Available Statistics

SERVICE_NAME	SERVICE_TYPE	CONTEXT_NAME	STATISTIC_NAME	STATISTIC_VALUE
scheduleOrder	AsyncReadsForNodeAvailability	/v2/availability/node/availableSupplies/	BatchLines	2,000
scheduleOrder	AsyncReadsForNodeAvailability	/v2/availability/node/availableSupplies/	Batches	20

SERVICE_NAME	SERVICE_TYPE	CONTEXT_NAME	STATISTIC_NAME	STATISTIC_VALUE
reserveAvailableInventory	AsyncReservations	/v2/reservations/	BatchLines	2,000
reserveAvailableInventory	AsyncReservations	/v2/reservations/	Batches	20

Sterling Intelligent Promising

Enhancements & New features:

- Inventory purge enhancement for SIP integrated environments.
- Supply snapshot and Demand snapshot
- `suppressZeroQty` flag to filter snapshot events.
- `considerEventThreshold` to filter availability snapshot events
- Utility to call SIP IV API in integrated environment.
- Optimizing response time.

General Best Practices →

Inventory Purge

- To selectively purge the specific entities which can be configured under agent criteria parameter `TableCode`.
- Purge data of `YFS_INVENTORY_SUPPLY`, `YFS_INVENTORY_DEMAND` & `YFS_INVENTORY_RESERVATION` irrespective of quantity
  - Criteria parameter: IsOrgMigratedToIV. Set it to Y if inventory org is migrated from GIV to SIP IV(Phase2)
  - Read more →

Demand/Supply snapshots

- Fulfillment managers can use `demandSnapshot` and `supplySnapshot` to get insights of supply/demand system for analysis and troubleshooting in advance.
- For more information:
  - Demand Snapshot →
  - Supply Snapshot →

SuppressZeroQty

- By default, Sterling Intelligent Promising sends availability events for all items with zero or nonzero quantity. To filter events with zero quantity, use the `suppressZeroQty` option in snapshot events.
  - `suppressZeroQty` Default value: false
  - When true, events with 0 quantity will not be considered when raising snapshot events.

considerEventThreshold

- `considerEventThreshold` can be used with availability snapshot events.
- When true, event threshold configurations will be queried to determine which SKUs will raise snapshot events, based on their availability.
- Default value: false
  - Additionally, if fulfillment is off either at node or item level availability change/snapshot events will not be published for that delivery method.

SIV Integration

- Use `CustomIVInvokeRestAPI` SDF service utility to make API calls to Sterling Intelligent Promising in developing customizations.
- This service uses the same properties for tenant credentials as that of OMS-SIV integration and takes care of authentication.
- This service automatically manages the access token, including generating a new token, renewing the token, and invalidating and resetting the token cache.

Read more →

API Input Limit

- In Sterling Intelligent Promising, a line limit has been imposed as of July 31st, 2024
- These action limits are as follows:
  - **Availability:** 100 actions per API invocation.
  - **Availability by date:** 100 actions per API invocation.
  - **Reservation:** 500 actions per API invocation.
- An action is measured by the number of unique SKU-Location combinations.

Read more →

Application Performance

UI Performance

- Web Store (wsc/isf)
- Call Center
- Order Hub

Best Practices

- Apply recommended configuration around API performance
- Optimize API inputs to avoid expensive or blank queries for order search scenarios.
- Ensure commonly used customer search attribute columns are part of the database index.
- Optimize API template to only pullout the data that's displayed on the UI.
- Run purges prior to peak to ensure transaction tables such as `YFS_ORDER_RELEASE_STATUS, YFS_ORDER_HEADER, YFS_ORDER_LINE, YFS_SHIPMENT`, etc are lightweight.
- Cache critical configuration data using entity cache: `YFS_REGION, YFS_REGION_DETAIL, YFS_ATTR_ALLOWED_VALUE, YFS_ATTR_ALLOWED_VAL_LOCALE`
- Add the following indices to enhance the performance of the Batch Pick
  - Index on the `STORE_BATCH_KEY` column of the YFS_SHIPMENT_LINE table.
  - Index on the `SHIPNODE_KEY` and `INCLUDED_IN_BATCH` columns of the `YFS_SHIPMENT` table.
- Set the property `yfs.applyChildContainerQueryOptimization=Y` in DB properties to optimize the query on the shipment container table while fetching child containers.
- `closeManifest` API must be called asynchronously
- Control/Set polling interval of Store/CC dashboard widgets
- Call `GetStoreBatchList` with optimum values for `MaxNumberOfShipments, NoOfShipmentLinesForNewBatch`
- Purge `YFS_INBOX` table, identify and address root cause of the exception, keep exception to minimal
- UI test cases to mimic the expected peak users and associated action.
- External calls from the UI to have an optimal timeout value configured.

Payment Integration

– Prior Webcast:

- Payments Deep Dive session

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

Common issues

1. Penny differences causing error infinite loop detected in dynamic distribution best match.
2. Orders having excessive charge transaction records.
3. Contention on YFS_ORDER_HEADER table.

General Best Practices →

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. 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.

User-Exit

– 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.

Handling Penny Differences

To avoid penny differences for returns, when a quantity is cancelled, use the return amount calculated by OMS instead of calculating this externally and then passing that amount to OMS.

OMS supports 2-digit decimals.

Consume latest fix packs, contains rounding enhancement

JMS Performance

- Message **PUT** slowness
- Agent (**GetJobs**) slowness
- Consumer is slow

Best Practices

- Review **MessageBufferPutTime** relative to **ExecuteMessageCreated** statistic from YFS_STATISTICS_DETAIL table for any slowness
- Use non-persistent queues for internal agent queues
- User persistent queues for external integration or integration server processes.
- Avoid using message Selector, instead have dedicated internal/external queues
- Avoid longer transaction to prevent **MQRC_BACKED_OUT** error message
- Optimize output template to prevent **MQRC_MSG_TOO_BIG_FOR_Q** error while posting a message
- Enable JMS Session Pool
  - **-yfs.yfs.jms.session.disable.pooling=N**
- Use anonymous reuse (requires JMS Session pooling to be enabled)
  - **-yfs.jms.sender.anonymous.reuse**
- Enable multi-threaded PUT's
  - **-yfs.yfs.jms.sender.multiThreaded=Y**
- Enable JMS connection retries
  - Retry Interval (milliseconds) 100 ms
  - Number of Retries – at least 3 retries.
- Enable agent bulk sender properties to POST message in batches.
  - **-yfs.agent.bulk.sender.enabled**
  - **-yfs.agent.bulk.sender.batch.size=5000** (increase as needed)

Database

Database Performance

- High DB response time
- Contention
- Long running queries
- High DB CPU/IO
- DB Transaction logs
- Backup takes time
  - DB is too BIG!

Best Practices

- Enabling property `yfs.yfs.app.identifyconnection=Y` to identify the source of DB query / connection.
- Control database size
  - Compressing the CLOB data using entity compression feature
  - Execute order audit and order release status purge
  - Disable unwanted audits.
  - Do not use YFS_EXPORT table for debugging purpose.
- Enable *`stmt_conc`* `(LITERALS)*`
- Avoid full table scan with `ConsiderOracleDateTimeAsTimeStamp` attribute when using Oracle database.
- 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.
- Blank queries, one of the common use case when non optimal API input is passed in. Ensure APIs are invoked with key filtering attributes.
- Enable performance features and properties required for concurrent workload to avoid DB contentions (HOTSku, Capacity, etc.)
- Disable resource intensive database maintenance during peak period (such as offline reorg on critical table)
- Tune / Avoid ad-hoc queries used for reporting purpose, if possible, use standby or replica instances to query.
- Monitor database for long queriers and queries in lock-wait, and transaction logs usage.

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!

Key Indicator

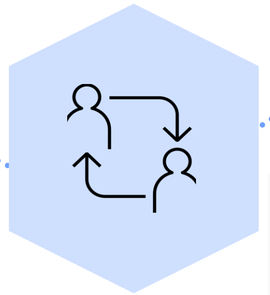
2023 Peak Days vs. 2024 Typical Volume vs. Performance Test

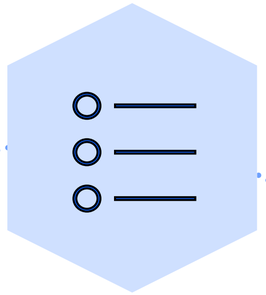
- Orders & Order Lines / hour
- Total Synchronous Calls (Inventory, Orders, Shipment, etc.)
- Full Sync's & Delta Updates for Inventory, Items, etc.
- Concurrent Store & Call Center Users

1

Projected peak volumes
Ensure business and IT are align on expected peak loads to ensure planned tests are accurate.

Plan





Prepare

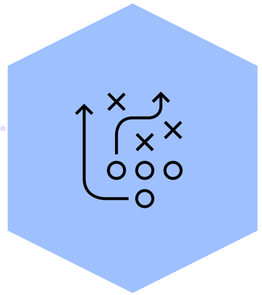
2

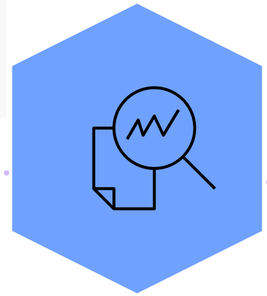
Representative Combination Tests
Assemble components to reflect real time DATA, scenarios and run in parallel to ensure adherence with NFR.
Order Profile, Inventory Picture, External Service Response Time and dataset everything must match production.

3

Execute
Run variety of test with all possible combinations.
Isolated Runs & Fullload (i.e. Create + Schedule + Release+ Shipment +Invoice + Store Ops + Inventory Snapshot (IV))

Execute





Tune

4

Tune
Analyze the KPIs and Tune processes, threads and profile to meet expected peak SLAs/NFRs on throughput.

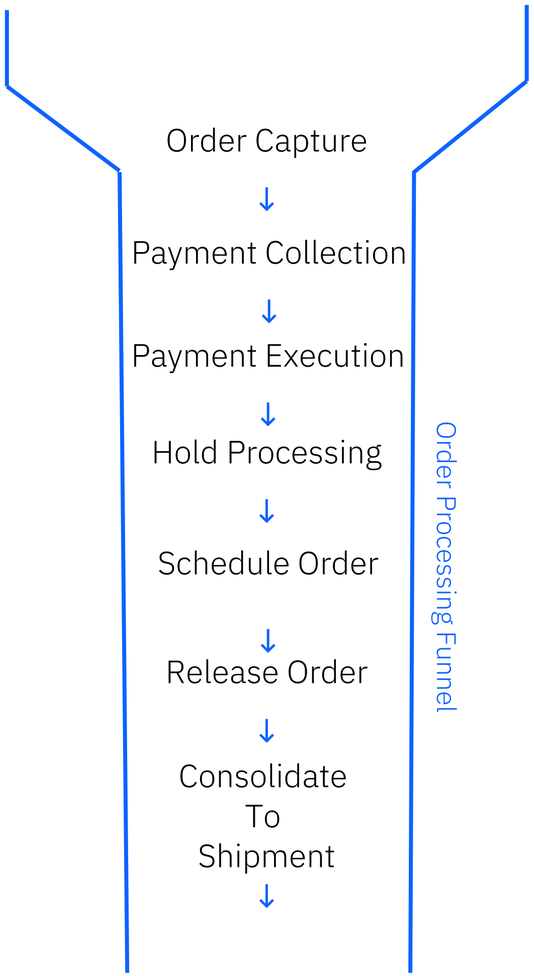
5

Scale
Achieve optimal setting for workload, and scale horizontally to meet performance NFRs and beyond.

Scale

Performance Testing & Optimization

Server Profile



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:

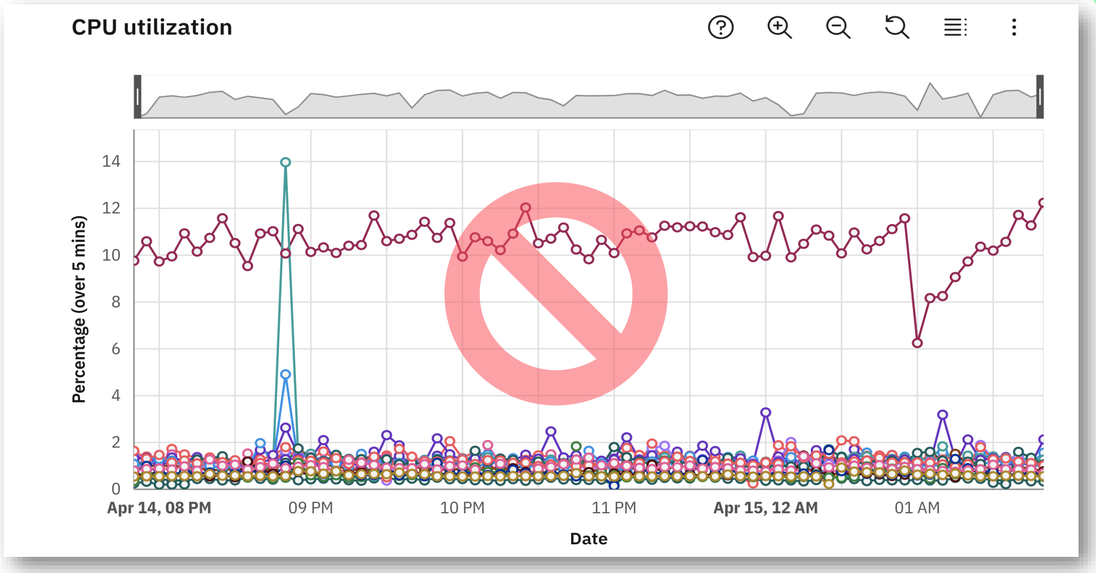
- Spawning additional (untuned) instances of agent to try and improve throughput let to **exhaustion of resource** allocation available
- Review KC Guidelines to select performance profile | Review community article on Sterling OMS Performance Profiles

Effective Server Consolidation

CPU and Memory usage for optimal consolidation.

CPU allocation 90.00%

Memory allocation 69.00%



SaaS - NextGen

1

Reduce # of Instances according to the workload; scale up using SST Server scheduling feature

Stop the servers that are not required or are obsolete.

Stop any failed servers; potentially misconfigured

2

Use Self-Service server scheduling feature

Schedule the processes such that it uses the CPU and memory resources optimally.

Choose correct performance profile; start with Balanced

3

Consolidate by grouping the server based on the functionality

Consolidate based on the scaling requirement

Consolidate based on the workload pattern;

4

Arbitrary groups with no more than 5 integration service or agent criteria; priority groups

OMS certified containers

Performance & Optimization

On Premise

Objectives

– Scale seamlessly for peak workload.

Best Practices

– Update to latest quarterly release prior to freeze
  – New fixes will be available on top latest quarterly release i.e., 10.0.2406.1
  – Deployment strategy (blue / green, canary, etc.)
– Avoid automatic operator updates for production.
– Review and be prepared to captured diagnostics as per the Mustgather.
  » Mustgather for IBM Order Management Software Certified Containers: Performance Issues →
– Check for SSL certificate validity for all internal and external communications.
– Tune readiness / Liveness probes
– Monitor NFS IOPS for shared mounts used to store certs, catalog index, etc.
– Reduce redundant RMI calls; verify host ulimits (Open File/Socket)

Discussion Points

– Review CPU and memory resource requests and limits, define optimal profiles, and agent & integration threads.

```
serverProfiles:
- name: ""
  resources:
    limits:
      cpu: millicores
      memory: bytes
    requests:
      cpu: millicores
      memory: bytes
```

```
- name: agents-huge
  profile: ProfileHuge
  property:
    customerOverrides: AgentProperties
    envVars: EnvironmentVariables
    jvmArgs: BaseJVMArgs
  replicaCount: 1
  agentServer:
    names:[ScheduleLargeServer,ReleaseLargeOrderServer]
```

  » Watch for CPU throttling.

– Adjust Default Executor Threads, Data source pool size according to `serverProfile` you have defined for the Application Server.
– Separate traffic using `appServer.ingress.contextRoots`
  » `[smcfs, sbc, sma, icc, isf, adminCenter]`
– Pod/Cluster Autoscalers.
– Network monitoring, latency to Database and JMS server; have network utility to validate the connections.
– Understand & Tune Ingress/Egress request/response limits

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

↓

01
Database Hygiene

- Maintaining 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.

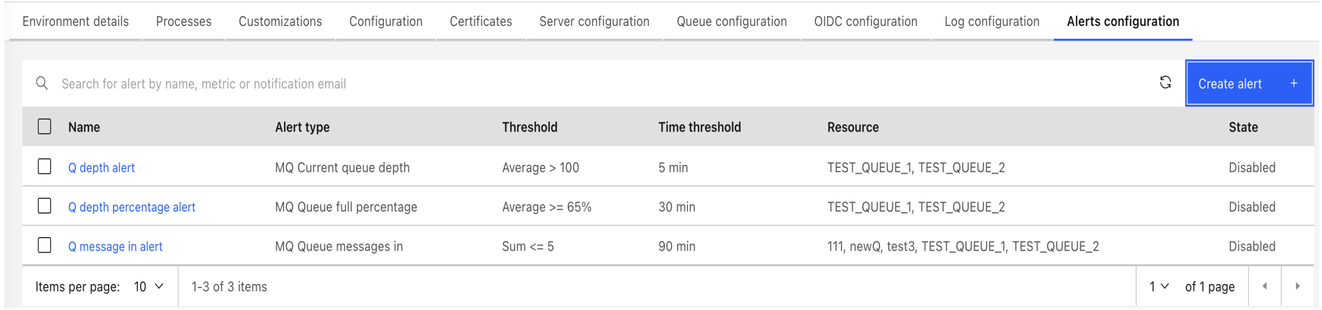
Peak Checklist

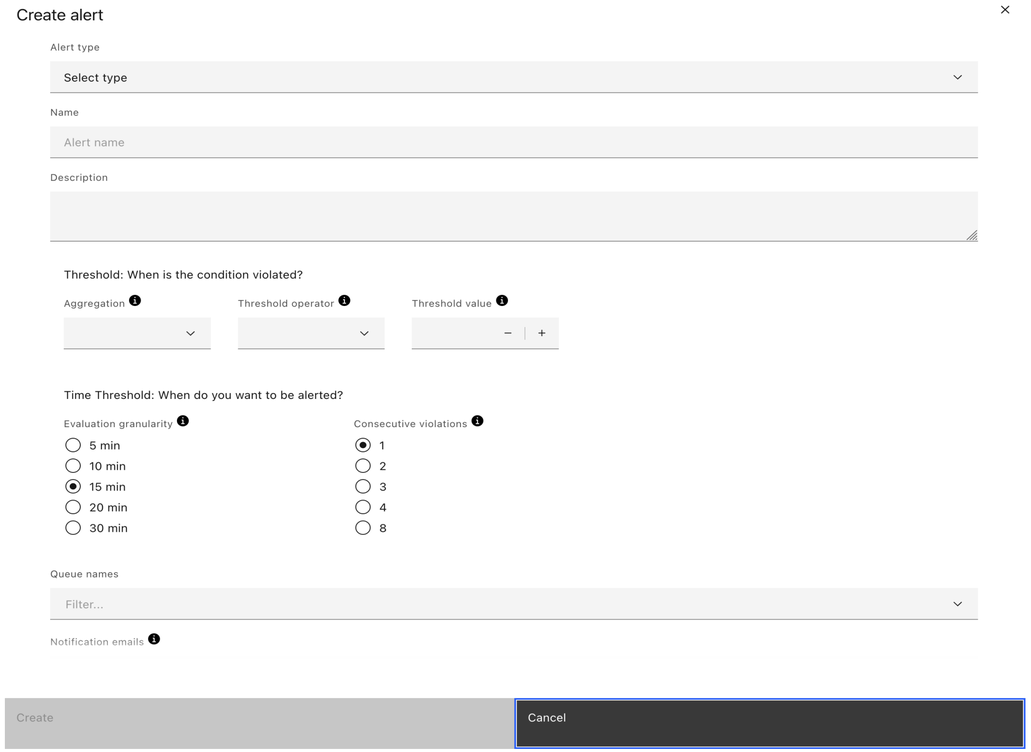
- ❑ Ensure all necessary **purges** are running to maintain healthy & lightweight database, which in-turn minimizes performance issues.
- ❑ **Disable** unnecessary transaction **audits** (Order Audits, General Audits, etc.)
- ❑ Implement **entity level database compression** for custom and OOB CLOB column 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.

IBM Sterling Self Service – Order Management

Alert Configuration*

1. The alert configuration feature will allow users to create and manage alerts.
2. Users can define standalone alert email recipients that’s separate from users in SST
3. Users can add custom label to each alerts that will be included in the notification.
4. Alerts can be duplicated to allow easy creation of similar alerts.
5. Robust search option to filter based on different alert attributes.
6. Ability to multiselect the alerts and manage them.
7. Auto merges similar type of alerts to avoid duplicates





Learn more about this here

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

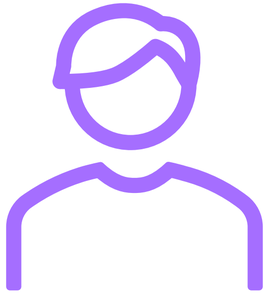
- ☑ Clear runbooks, RACI
- ☑ Quickly detect issues
- ☑ Throttle as necessary
- ☑ Quick mitigation

IBM Support Offering – **Advanced Support**

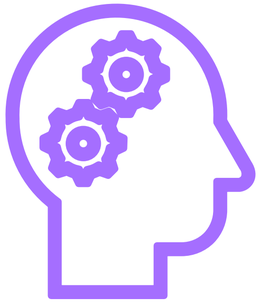
NEW in 2Q23!

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

www.ibm.com/support/pages/ibm-advanced-support-offering

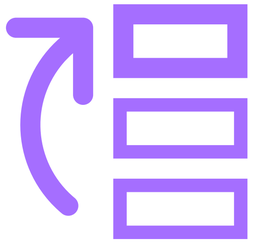


Named IBM Advanced Support Focal (ASF)

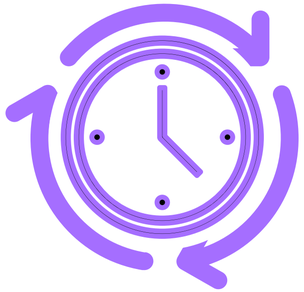


Priority access to Senior Technical Support Squad

Enhanced initial, ongoing response SLOs



Higher ongoing case prioritization



24x7 coverage for priority Sev-2

Manage, escalate backlog via cadence, reporting

MC

Next Steps

IBM Advanced Support

www.ibm.com/support/pages/ibm-advanced-support-offering

Contact your IBM Client Success Manager, Account representative, or Mike Callaghan(mcallagh@ca.ibm.com)

Sterling OMS Support 101

www.ibm.com/community/101/sterling/oms/

Technical Best Practices

Start with the new Performance Guide

Are *you* ready?

Follow the Guide

Thank you

This content was provided for informational purposes only. The opinions and insights discussed are those of the presenter and guests and do not necessarily represent those of the IBM Corporation.

Nothing contained in these materials, or the products discussed is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers, or altering the terms and conditions of any agreement you have with IBM.

The information presented is not intended to imply that any actions taken by you will result in any specific result or benefit and should not be relied on in making a purchasing decision. IBM does not warrant that any systems, products or services are immune from, or will make your enterprise immune from, the malicious or illegal contact of any party.

All product plans, directions and intent are subject to change or withdrawal without notice. References to IBM products, programs or services do not imply that they will be available in all countries in which IBM operates. IBM, the IBM logo, and other IBM products and services are trademarks of the International Business Machines Corporation, in the United States, other countries or both. Other company, product, or services names may be trademarks or services marks of others.