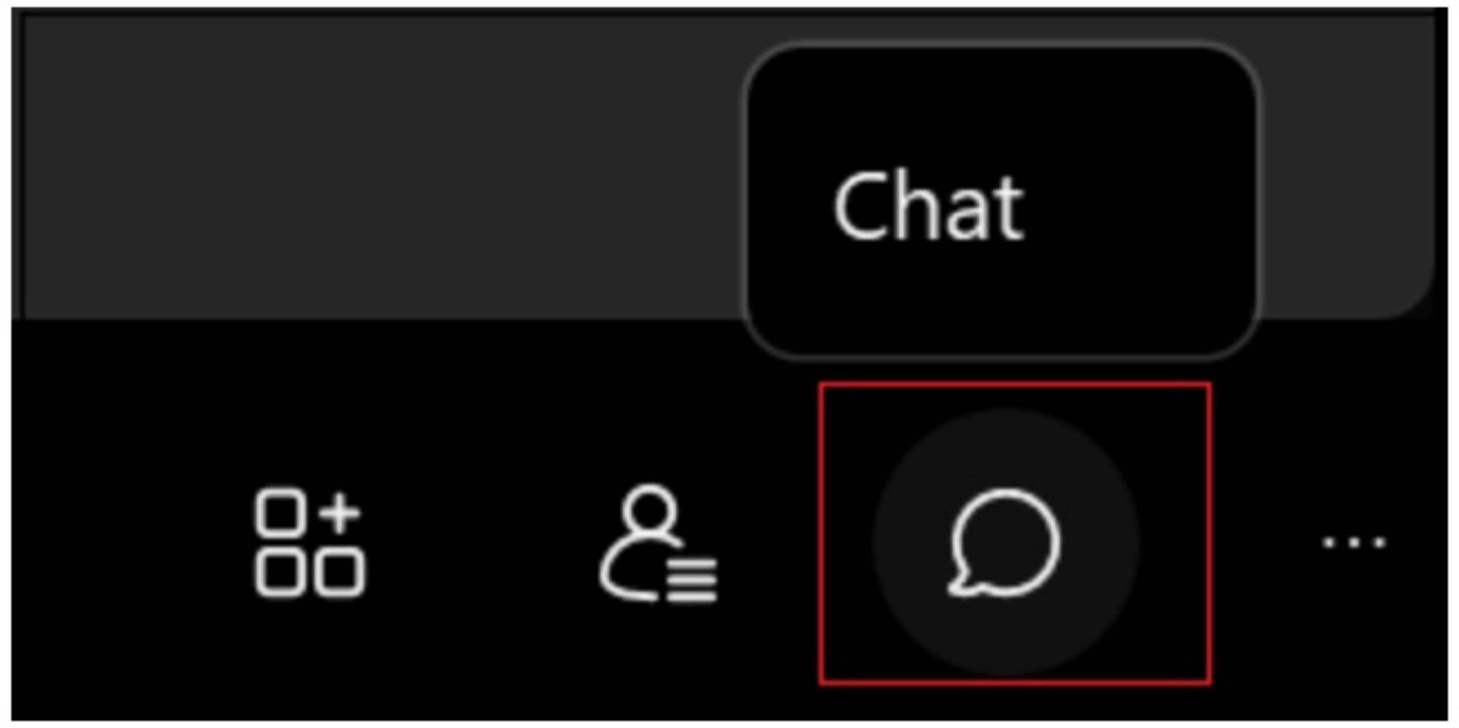
IBM Sterling Order Management

Holiday Readiness 2023

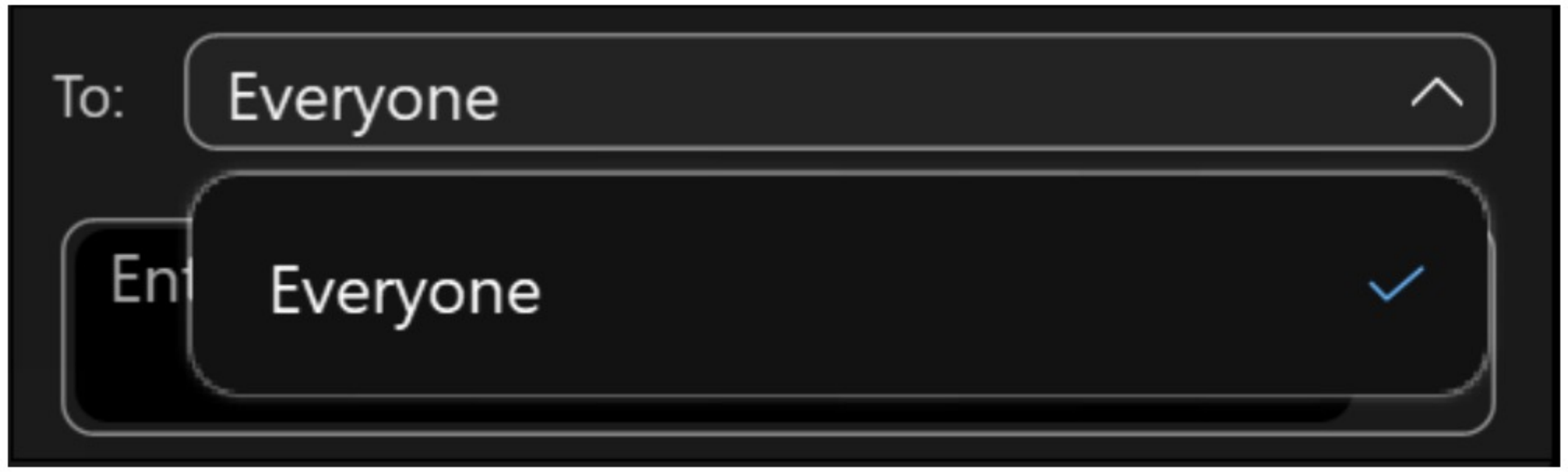
Panel Discussion

Have a Question(s)?

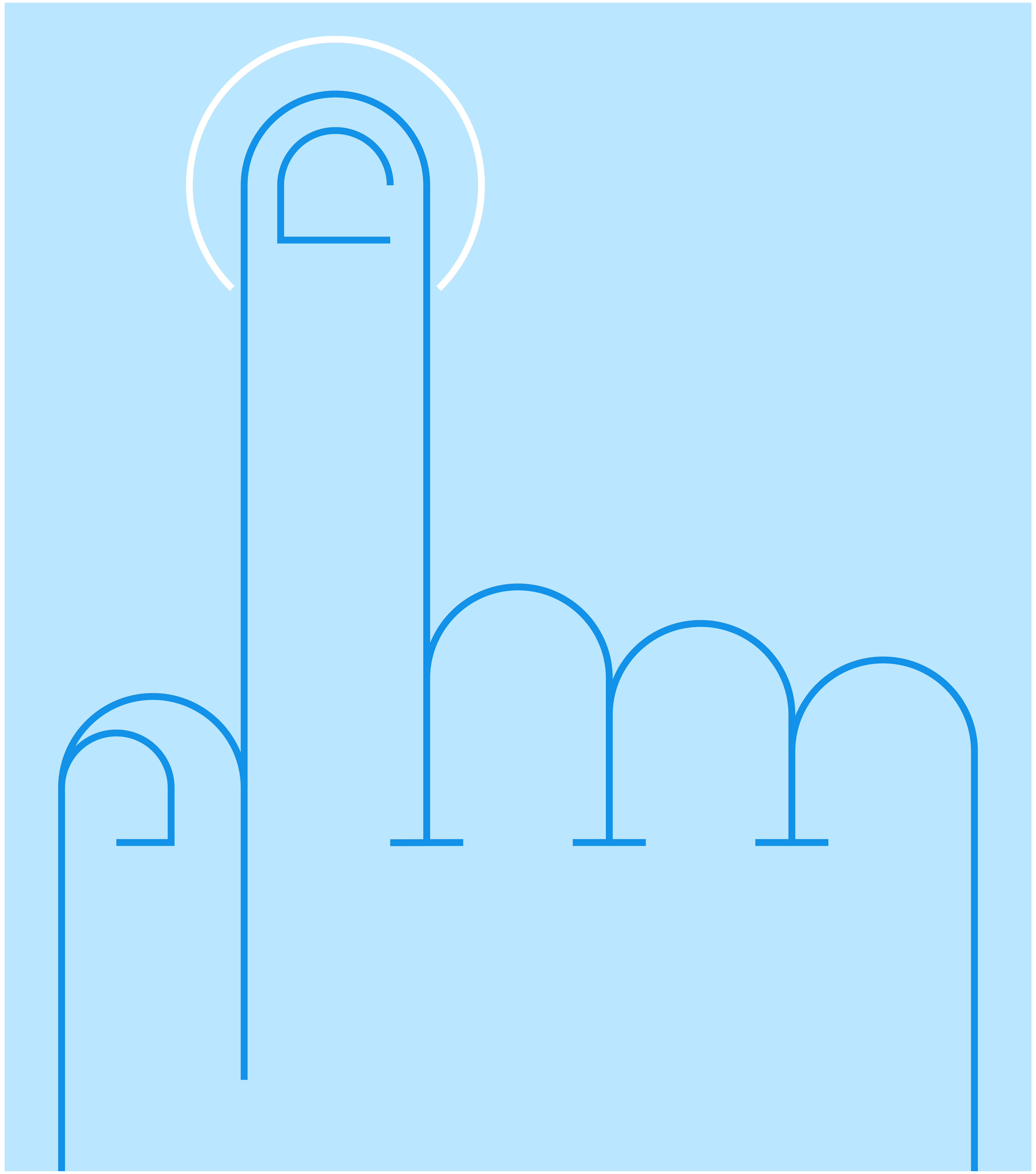
1. Open the Chat panel from the link in the lower right of the meeting window:



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 panelist

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

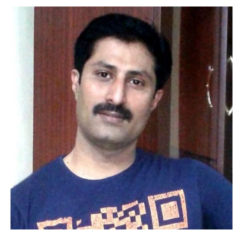
Jitendra Buge
Technical Support Engineer
Order Management Support

Jelena Markovic
Support SME
Order Management Support

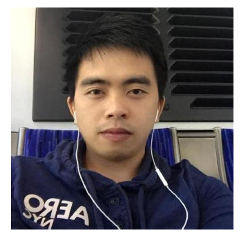
Vishal Aurora
Advisory Software Engineer – IBM Sterling

Vijaya Bashyam
STSM, Chief Architect Containerization and Performance, IBM Sterling

Yaduvesh Sharma
Senior Software Architect - Sterling order management


Bobby Thomas
Performance Architect, SRE - Sterling order management

Paresh Vinaykya
Executive Technical Account Manager – Expertise Connect


Tin Vo
Senior Software Developer - Sterling order management

Let’s Discuss

Our Journey to peak success

- Review promising and sourcing APIs
  - HotSku & OLA
  - Capacity: Resource pool locking & optimization

- Item-based allocation (IBA) use & considerations for peak.

- Sterling Inventory Visibility (IV) consideration.

- IBM Order Management Software Certified Containers
  - Performance
  - Support Mustgather

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!

IBM Holiday Readiness Preparation

IBM Order Management teams are focused year-round on ensuring stability during your peak season.

Performance Experts across Engineering, SRE, and Support collaborate to ensure the IBM platform, operations and support are ready.

The primary objective is to ensure stability and performance, and to quickly identify and mitigate any potential issues that do arise.

Platform Stability

Regular IBM cross-functional retrospectives drive continuous improvement, early identification and mitigation of potential risks

- Production performance and capacity reviews – comparison of current and projected peak workloads
- Internal Platform and application monitoring, alerts and runbooks
- Final major release (23.3) to address any remaining critical product issues
- Published Change Management and Operations freeze
- Internal Executive reviews to discuss status and action of key accounts
- Mobilization of cross-functional SWAT team with for rapid engagement 24x7

Best Practices

Publish critical updates to a robust collection of self-help focused on peak success

- Derived from Support and Engineering experiences with production operations and internal performance testing
- 3 external webinars complete in 2023, 2 more on the horizon
  - Journey to Peak Success (March)
  - Payment Integration & UE Implementation (May)
  - Recommendations & Best Practices (August)
  - Panel Discussion (now!)
  - Peak Day Mitigation (October)
- Serve as basis for internal application configuration audits, proactive reviews

Prescriptive Guidance

Proactive ‘Event Readiness’ reviews in partnership with Expertise Connect for targeted client-specific recommendation

- Peak Questionnaire
- Production performance, alert review
- Application workload & config
- Database workload & config
- Integrations (MQ, IV)
- Housekeeping activities
- Close loop on prior incidents
- Runbooks, mitigation techniques

- Case TS013870308 used to document 2023 proactive review

MC

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.thresh old`

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 →

IBA (Item Based Allocation)

IBA process basically reallocates the promised demands for items of existing customer orders, to more suitable supplies, based on certain supply and demand changes in the system.

Objectives:

– Performance considerations when using IBA during peak, or high traffic.

– Should we be running IBA required to be running for peak?

– Why is it resource intensive process?

Why IBA causes blocking locks ?

IBA documentation link ->

Disable IBA

– Enterprise level IBA rule

– Node level configuration

– Item level configuration

– Purge YFS_IBA_TRIGGER

– Beaware of `manageItemBasedAllocationTrigger`

Best Practices

– Aggressively purge YFS_ORDER_RELEASE_STATUS table.

– Maintain YFS_IBA_TRIGGER table.

– Avoid running IBA with schedule or release workload. Time box the execution to avoid database contention.

Considerations:

– If not required, disable IBA completely instead of just stopping the agent.

– Hot SKU feature should not be enabled in IBA JVM

– If not using IBA, then clear out any old entries from YFS_IBA_TRIGGER table using inventory purge agent

Sterling Inventory Visibility

OMS – IV Integration

Objectives

- Accurate Inventory picture
- Avoid slow API response due to excessive token generation.

If there is an unexpected spike in the number of cancellations or backorders, what are various factors that need to be reviewed?

- Sourcing configurations still maintained at OMS
- Capacity definition, capacity availability or cache capacity
- Node marked as having incorrect availability- Entry in YFS_INVENTORY_NODE_CONTROL
- Node notification time
- Calendar, and Carrier service schedule
- Scheduling Rule, Optimization and Line dependencies
- Unsupported delivery method

Token Management

- Reuse IV token as much as possible; not 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.

Best Practices

- Beware of rate limits
- Use Optimal Payload
- Use New/enhanced APIs
- Avoid redundant Snapshot calls
- Monitor for failed events
- Avoid redundant availability recomputes

Considerations

- Recompute Network Availability API to recompute DG availability
- Using Node fulfillment override to handle node capacity or to turn on/off fulfillment from Stores according to their opening hours.
  - » `expiryTs` *vs.* `onhandEarliestShipTs`
- Optimize on inventory change events (snapshot). If not using different delivery methods, then disable it. Prevent consumer from getting overload.
- Handle expired reservations
  - » `defaultExpirationMinutes`

OMS certified containers

Performance & Optimization

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., September 2023 (10.0.2309.0)
  – 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

How to Succeed

Plan

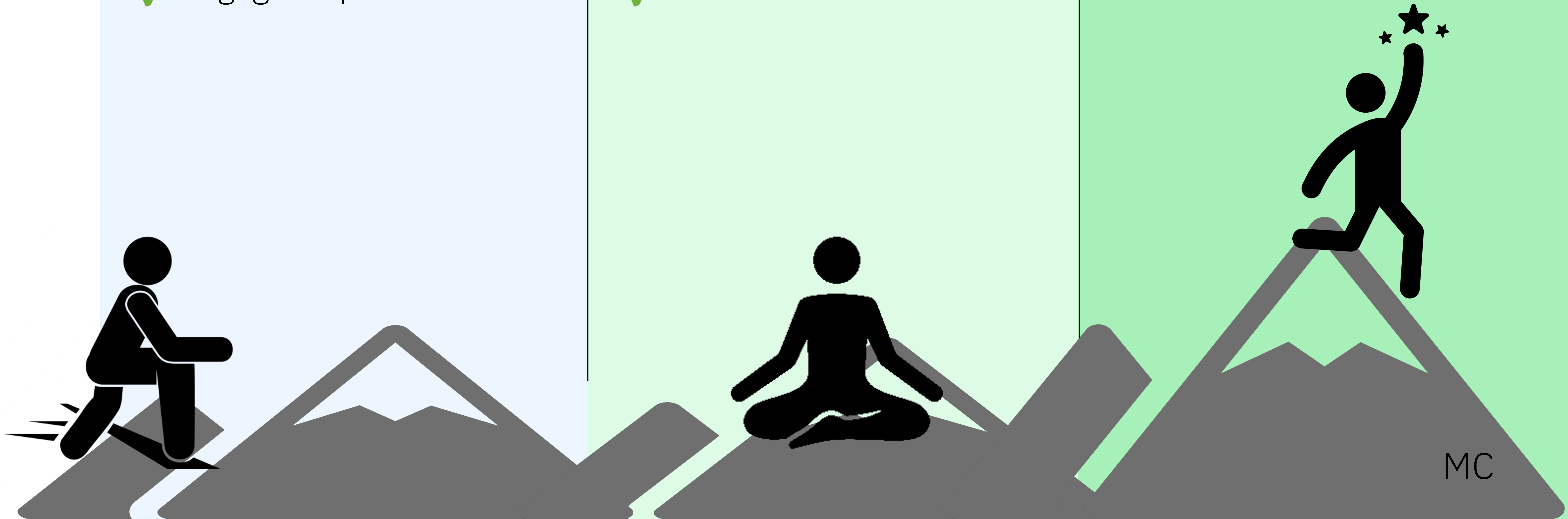
- ✓ Retrospective
- ✓ Latest product levels
- ✓ Detailed projections
- ✓ Catch prior webcasts
- ✓ Engage help as needed

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 <u>no later</u> 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

Apply Recommended Configuration

We leverage learnings from each go-live, peak event, all critical production issues to continuously improve our internal and external published Best Practices

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)*

Application Performance

01

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

02

API Performance

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

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. Refer to slide 9 for additional guidance.
- 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

Application Performance

03

UI Performance

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

Best Practices

- Apply recommended configuration around API performance
- 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_VALUES, 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

Application Performance

04

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

Integration 05

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)

Integration 06

External System

- Sterling Intelligent Promising
  - Inventory Visibility
- Store Inventory Management
- Order Optimizer
- Order Service

Best Practices

- Use connection pool (cached/persistent connection, keep-alive) with appropriate connect and read timeouts.
- Cache authentication token for reuse, regenerate upon expiry, or 401, 403 status codes.
- Use
- Adhere to best practice when invoking Inventory Visibility APIs.
  - Use 100 item-node per payload when invoking APIs for multiple lines.
- Implement polling process to retrieve failed events.
- Space out the sync supply and snapshot calls, check with all stakeholders for ad-hoc execution or special requests during peak.
    - Avoid redundant calls to generate snapshot
- 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.
    - Turning on/off existing nodes.
  - Make sure to review release notes and API documents.
    - Timely refactor the logic to avoid running into unforeseen risks of using deprecated or discontinued APIs.

Database 07

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 & 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

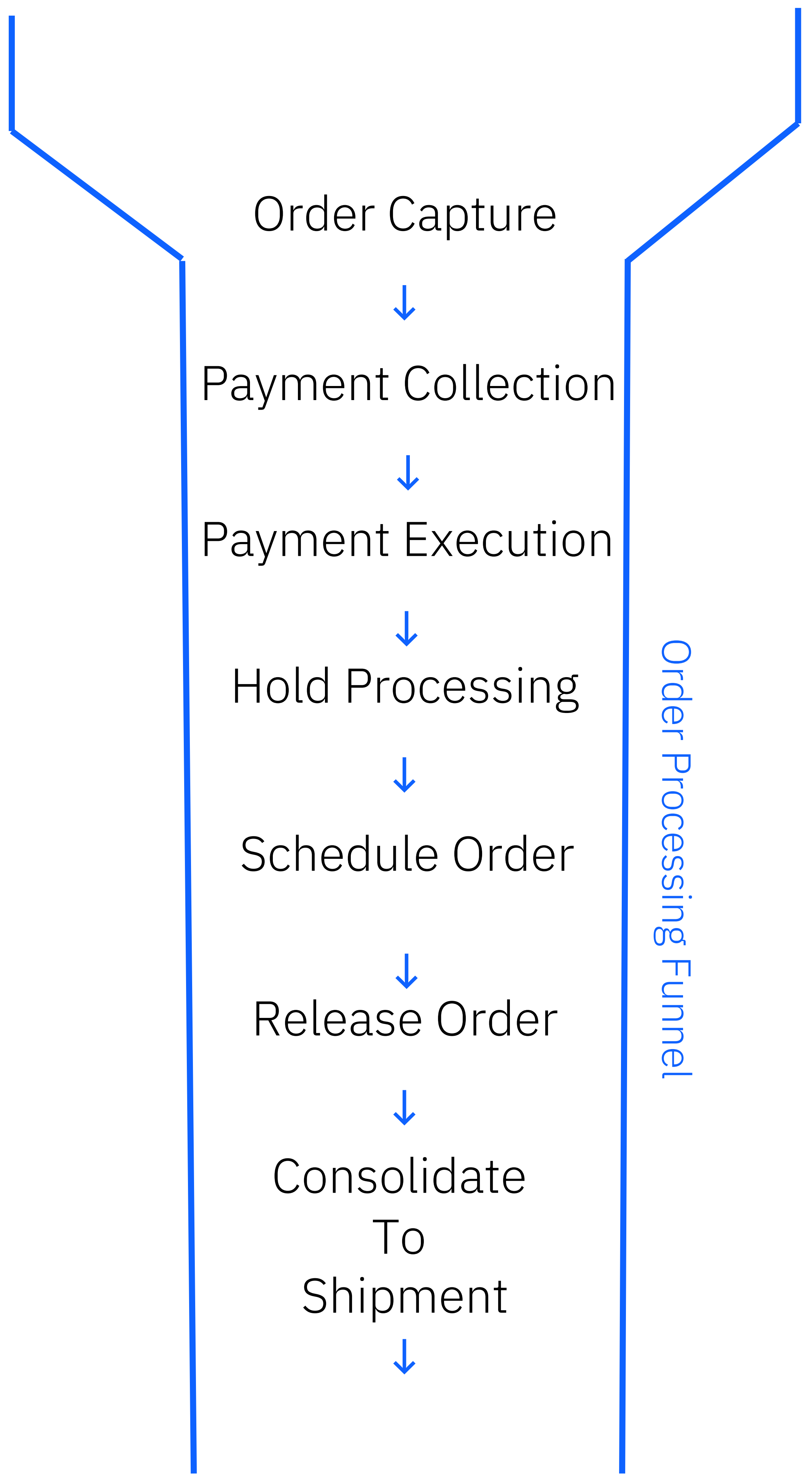
Metric	2022 Peak	2023 Projected Peak	2023 Load Test Peak
Orders / hour (max)	?	?	?
Orderlines / hour (max)	?	?	?
Get Inventory Availability	?	?	?
Reserve Inventory	?	?	?
Inventory Adjustment trickle	?	?	?
Inventory Adjustment burst	?	?	?
Concurrent Store/CC users	?	?	?

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.

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

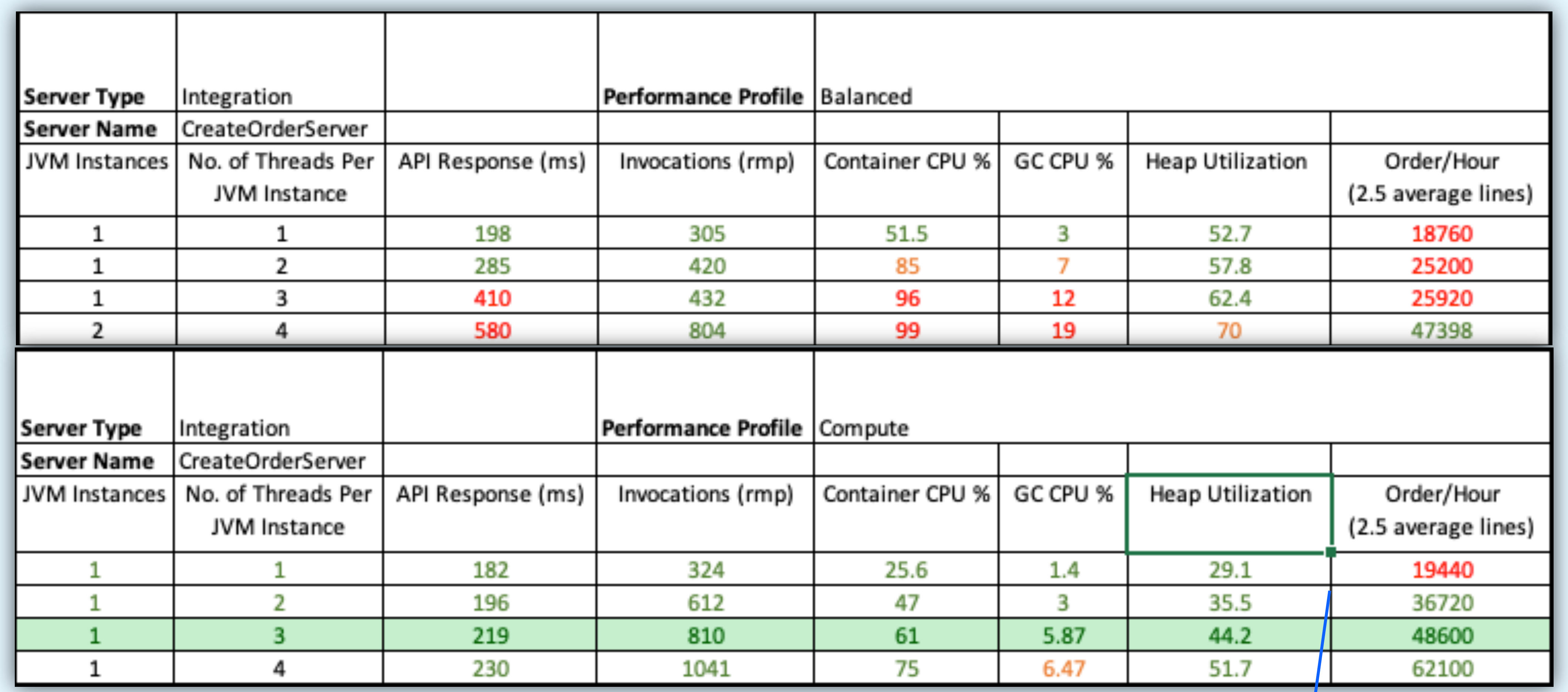
Execute and **monitor** the server performance via **Self Service Tool**

↓ 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

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.

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.

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

↓

03
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.

04
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.

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…)*

- Over the past few years order fulfillment has shifted to stores with BOPIS, 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

SB