

Solr Upgrade Plan



J.D. POWER | **AUTODATA**
SOLUTIONS

Solr Upgrade Plan



J.D. POWER | **AUTODATA**
SOLUTIONS

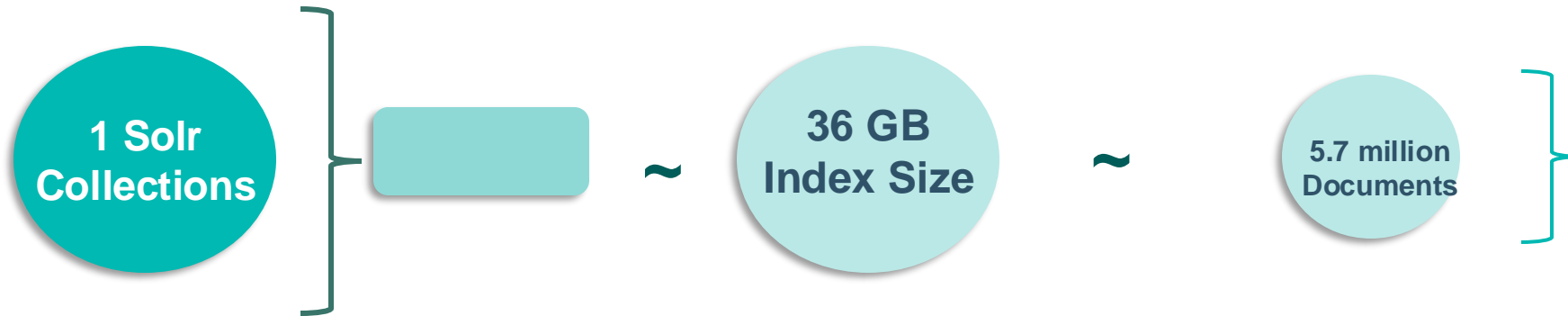
Overview



- JDPA is currently using Solr 7.7
- As a part of this project JDPA wants NextBrick to upgrade solr from 7.7 to 9.8 version , and implement cross dc replication
- NextBrick will do Health Check & Performance Tuning also after the successful migration of Solr, Kafka setup and other dependent services.

Upgrade Objectives

JDPA is currently using **Solr 7.7** with External Zookeeper setup



Upgrade Objectives

1. Upgrade Solr from version 7.7 to 9.8
2. Migrate from CDCR to XDC replication framework
3. Minimize downtime and data loss
4. Ensure data consistency across all data centers
5. Validate and optimize performance post-upgrade



Strategy 1: Direct Upgrade from Solr 7.7 to Solr 9.8

The Direct Upgrade from Solr 7.7 to Solr 9.8 will not work smoothly because:

1. CDCR is Removed in Solr 9.8

- Your existing CDCR setup will break since Solr 9.8 no longer supports it.
- Solr 9.8 does not provide an out-of-the-box alternative to CDCR.

2. Major Breaking Changes Between Solr 7.7 and Solr 9.8

- Index format changes: Direct upgrade means old indexes may not be readable in Solr 9.8.
- Field Type Changes:
 - TrieFields are completely removed → All numeric fields need to be migrated to PointFields.
- Configuration Changes:
 - solrconfig.xml and schema.xml require manual modifications before upgrading.

3. Java & ZooKeeper Upgrades Required

- Solr 9.8 requires Java 17, while Solr 7.7 runs on Java 8.
- Solr 9.8 requires ZooKeeper 3.8+, while Solr 7.7 often runs on older versions

Direct Upgrade is NOT RECOMMENDED in this case due to **CDCR removal** and **breaking changes**.



Strategy 2: Intermediate Migration Strategy: Solr 7→Solr 8→Solr 9.8

Old CDCR Still Works in Solr 8 : Since Solr 8 still supports CDCR, you can upgrade Solr while keeping CDCR functional.
But it doesn't solve the biggest challenge: **CDCR is still removed in Solr 9.x, and Solr 8 does not support the new Kafka XDCR natively**



Strategy 3: Parallel Blue-Green Deployment

The parallel deployment method **directly migrates to Solr 9.8** while keeping Solr 7.7 running.

- ✓ **Zero Downtime:** Solr 7.7 remains live while we set up Solr 9.8 in parallel.
- ✓ **No CDCR Downtime:** CDCR keeps running in Solr 7.7 while Kafka XDCR is tested.
- ✓ **Avoids Solr 8 Upgrade:** Directly migrate to Solr 9.8 without an unnecessary step.
- ✓ **Safer Migration:** Ensures **Solr 9.8 is fully tested before switching production traffic.**

Migration Steps (1/3)

- Infrastructure Setup
 - Provision New Hardware:
 - Set up new servers or cloud instances for Solr 9.8 cluster.
 - Ensure hardware specifications match or exceed the current 7.7 cluster.
 - Install Java 11 or 17:
 - On all new nodes, install the required Java version for Solr 9.8.
 - Install Solr 9.8:
 - Download and install Solr 9.8 on the new nodes.
 - Configure the new cluster with optimal settings for your use case.
- Set up an Apache Kafka cluster to act as the distributed queue between data centers.

Migration Steps (2/3)

- Configuration and Schema Migration
 - Review and Update Configurations:
 - Analyze your current solrconfig.xml and update it for Solr 9.8 compatibility.
 - Example: Update `<requestHandler name="/select" class="solr.SearchHandler">` with any new parameters
 - Set up new servers or cloud instances for Solr 9.8 cluster.
- Schema Updates:
 - Review your schema with 1200 fields for any necessary updates.
 - Example: Update field types if using any deprecated analyzers.
- For more details visit (<https://solr.apache.org/guide/solr/latest/upgrade-notes/major-changes-in-solr-9.html>)
- Configure the CrossDC Solr Module with Kafka details
- Check for the Security upgrades

Migration Steps (3/3)

- Data Migration and Validation:
 - Perform a full reindex of your data after upgrading.
 - Validate search quality and performance against the old cluster.
 - Replay the prod queries to the new cluster to verify the search relevance
- Dual-Write System: Implement a dual-write system to update both old and new clusters during the transition period
- Performance Monitoring:
 - Set up comprehensive monitoring to compare performance between the old and new replication systems
 - Performance benchmarking of the New solr, to optimise some query, indexing pipeline or the jvm params
 - Optimize Kafka settings for your specific use case, including buffer sizes, replication factors, and retention policies
- Testing and Validation: Thoroughly test the new XDCR setup in a staging environment before applying it to production

Software Versions

- Solr 9.8
- Zookeeper 3.8+
- Java 11 (Oracle JDK)
- 2 Kafka Clusters with 3 Zookeepers + 5/7 brokers each. Storage: 12 X 1 TB disk. RAID 10 is optional, Separate OS disks from Apache Kafka® storage Memory: 64 GB vCPU 12 cores

Provisioning the XDCR Kafka Cluster (1/2)

Cluster provisioning

- Hardware Requirements:
 - Use dedicated servers or cloud instances with high CPU, RAM, and SSD storage.
 - Minimum 3 brokers per data center for fault tolerance.
- Network Configuration:
 - Ensure low-latency, high-bandwidth connections between data centers.
 - Configure firewalls to allow Kafka port (default 9092) traffic between clusters.
- Kafka Version:
 - Use Kafka 2.4.0 or later for improved XDCR support.
- Zookeeper Ensemble:
 - Set up a separate Zookeeper ensemble for each data center.
 - Use at least 3 Zookeeper nodes per ensemble for quorum.

Provisioning the XDCR Kafka Cluster (2/2)

Topic Configuration

- Replication Factor:
 - Set replication factor to at least 3 for fault tolerance.
- Partitions:
 - Calculate partitions based on expected throughput and consumer parallelism.
 - Formula: $(\text{desired_throughput} * \text{avg_latency}) / \text{avg_batch_size}$

Scaling and Performance Tuning

- Vertical Scaling:
 - Increase CPU, RAM, and disk resources on existing brokers.
- Horizontal Scaling:
 - Add new brokers to the cluster and rebalance partitions.
- Partition Reassignment:
 - Use `kafka-reassign-partitions.sh` tool to redistribute partitions across brokers.

Monitoring:

- Implement monitoring using tools like Prometheus and Grafana.
- Key metrics: broker CPU, disk usage, network I/O, under-replicated partitions..

Special Cases

Scenario which requires the special handling In this case

- Network Partition:
 - Implement rack-aware partition assignment.
 - Use `min.insync.replicas` to ensure data consistency.
- Broker Failure:
 - Set `unclean.leader.election.enable=false` to prevent data loss.
 - Maintain sufficient replication factor for failover.
- Data Center Outage:
 - Configure consumers and producers to fail over to the secondary data center.
 - Implement automated failover using tools like ZooKeeper for leader election.
- High-Volume Traffic:
 - Implement backpressure mechanisms in producers.
 - Use throttling on broker side: `follower.replication.throttled.rate` and `leader.replication.throttled.rate`.
- Large Messages:
 - Adjust `message.max.bytes` on brokers and `max.partition.fetch.bytes` on consumers.
 - Consider compressing messages or splitting into smaller chunks.

Solr Upgrade Approach

- NextBrick suggests to install **Solr 9.8** with recommended **Zookeeper 3.8+** on Linux OS.
- NextBrick will program to fetch the data from existing **Solr 7.7** indexes and push to the new collections in **Solr 9.8** setup.
- Proper sanity check will be done to verify the complete search features & data migration. QA team/Consumer apps will verify the search & indexing functionalities with new **Solr 9.8** setup.



Solr Upgrade Tasks

- Download and install [Zookeeper 3.8+](#) on 3 Linux VMs with JDK 11.
 - Define Ensemble properties in ZK config file (zoo.cfg)
 - Define GC & Logging properties in ZK environment config file (zookeeper-env.sh)
 - Setup service to start/stop Zookeeper
- Download and install [Solr 9.8](#) on 6 Linux VMs with JDK 11.
 - Define properties in Solr startup script (solr.in.sh)
 - Setup service to start/stop Solr
- Collection creation
 - Modify managed-schema & solrconfig.xml in Solr 8.x for Policy & Activity collections based on configurations available in [Solr 7.7](#)
 - Upload the configs in ZK
 - Create Policy & Activity collection using uploaded configs



Solr Upgrade Tasks

- Nextbrick product script/program
 - Program will query the collections in Solr 7.7 using CursorMark to fetch data.
 - Fetched data will be pushed into Solr 9.8 in batch mode.
 - Auto commit settings will be disabled during this execution and will be controlled by program.
- Steps for data migration
 - Pick a time when system load is at its lowest or setup independent server with Solr 7.7 index solely for this purpose only.
 - Pause all data updates if using existing Solr 7.7 index
 - Execute the script/program to fetch & ingest the data to new Solr 9.8
 - Enable Auto commit in new Solr 9.8
 - Enable data updates



Solr Upgrade Tasks

- Script for data Backup/Restore
 - Script will use Solr API to create the backup of the current index snapshot on the disk.
 - It will take care of cleaning up the old backups.
 - Whenever needed Solr index can be restored from backup to any Solr environment.
- Solr Security
 - NextBrick will implement BasicAuthentication to secure the access to Solr APIs.
 - NextBrick will create Roles, Permissions & Users as per the need of JDPA.
- Test the new setup of Solr 9.8
 - Test sample queries and compare results between Solr 7.7 and Solr 9.8
 - Test search functionalities from Consumer apps
 - Test data indexing from Java services

Risk and Impact Analysis

Kafka Cluster Placement and High Availability:

There were questions about whether the Kafka cluster should reside in the source or destination data center?

- Source and how to ensure high availability if one cluster goes down.
- 2 cluster
- Consumer
- Topics Partitions
- Leader

Risk and Impact Analysis

To ensure high availability in [Apache Kafka 3.9.0](#), you need a fault-tolerant architecture that can handle cluster failures. Here are the key strategies:

- **Multi-Node & Multi-Broker Setup**
 - Deploy **at least three brokers** to ensure quorum-based leader election.
 - Use **replication factor >1** to ensure redundancy of partitions across multiple brokers.
- **Replication & Leader Election**
 - Set **replication.factor=3** to maintain copies of partitions across brokers.
 - **Enable automatic leader election** (**auto.leader.rebalance.enable=true**) to ensure seamless failover if a broker goes down.

Risk and Impact Analysis

Use `min.insync.replicas=2` to require at least two in-sync replicas before acknowledging writes, preventing data loss.:

- **Multi-Region & Cross-Cluster Replication**

- Use **MirrorMaker 2.0** for cross-region replication to another Kafka cluster.
- Deploy Kafka brokers in multiple **availability zones (AZs)** if using cloud infrastructure.
- Implement **geo-redundant storage** in case of region-wide failures.

- **ZooKeeper/KRaft Availability**

- If using ZooKeeper, deploy at least **3 or 5 ZooKeeper nodes** to maintain quorum.
- If using **KRaft mode (Kafka 3.x default)**, have **at least 3 controllers** to ensure fault tolerance.

Risk and Impact Analysis

- **Data Volume and Retention:**

- Concerns were raised about the volume of data and the retention period,
- as Kafka can use a lot of CPU and disk space, especially with high throughput and large payloads.
- Kafka can use a lot of CPU and disk space
- Right compression techniques
- Higher number of brokers and partitions
- Proper monitoring ,altering mechanism (grafana,prometheus)
- Jolo pass yaml file works with jmx port

Risk and Impact Analysis

To prevent **high CPU and disk usage** in Kafka, especially with high throughput and large payloads, you need to optimize **data retention, compression, partitioning, and resource management**. Here's how:

- **Optimize Data Retention Policies**
 - Reduce retention **time** (`log.retention.hours` or `log.retention.ms`)
 - Example: Keep data for 7 days instead of indefinitely.
 - `log.retention.hours=168` # 7 days
 - **Limit log segment size** (`log.segment.bytes`)
 - Example: Reduce to 500MB - 1GB instead of multi-GB segments.
 - `log.segment.bytes=1073741824` # 1GB
 - **Use log cleanup policies** (`log.cleanup.policy=delete` or `compact`)
 - Set delete to remove old logs automatically.
 - Use compact if you only need the latest record per key.

Risk and Impact Analysis

- **Enable Compression to Reduce Disk Usage & CPU Load**
 - Enable compression at the producer level (compression.type in producer config).
 - Recommended types:
 - lz4 – fast and efficient, good balance.
 - zstd – better compression ratio, lower CPU than gzip.
 - compression.type=lz4
- **Manage Partitions & Brokers Efficiently**
 - Avoid too many partitions per broker (high CPU load).
 - Keep less than 4,000 partitions per broker (depends on hardware).
 - Distribute partitions evenly to prevent hot spots.
 - Use rack awareness (broker.rack) to ensure fault tolerance and even load distribution.
- **Optimize Producer & Consumer Behavior**
 - Batch records to reduce overhead:
 - batch.size=16384 # 16KB
 - linger.ms=5 # Small delay to batch messages
 - Tune fetch sizes to avoid excessive small reads:
 - fetch.min.bytes=1048576 # 1MB per fetch
 - fetch.max.wait.ms=500 # Reduce polling CPU usage

Risk and Impact Analysis

- **Offload Cold Data Using Tiered Storage (If Available)**
 - Store older logs in S3, HDFS, or Azure Blob Storage instead of Kafka's local disk.
 - Some distributions like Confluent Kafka support tiered storage for long-term retention
- **Monitor & Scale Resources Proactively**
 - Use Prometheus + Grafana or Confluent Metrics Reporter to track:
 - Disk usage (`df -h` or `du -sh /var/lib/kafka`)
 - CPU usage (`top`, `iostat`, `sar`)
 - JVM memory consumption (`jstat -gc` or `kafka-topics.sh --describe`)
 - Scale brokers horizontally instead of overloading fewer nodes.
- **Cleanup & Maintenance**
 - Delete unused topics (`kafka-topics.sh --delete`).
 - Run log compaction manually to remove redundant records:
`kafka-log-cleaner.sh --topic my-topic`
 - Would you like help with a custom tuning strategy based on your expected workload (e.g., number of events per second, payload size)?

Risk and Impact Analysis

Handling Outages: The team discussed the potential impact of network outages or periods when consumers are not available, emphasizing the need for a robust solution that can handle such scenarios gracefully.

- Mirror maker
- Active passive kafka cluster , mirror maker to take care of cross cluster data replication
- When data center is down we can seamlessly migrate producer applications to the passive cluster
- We have 2 kafka clusters, and mirror maker replicates data from source to target

Risk and Impact Analysis

Security and Authentication: There were issues with the old CDCR handling security, and questions about whether the new solution can handle authentication and security requirements effectively.

Both **Apache Solr 9.8** and **Apache Kafka 3.9.0** provide robust security features to protect data, manage access control, and ensure secure communication. Below is an overview of their key security capabilities.

Solr 9.8 Security Features

1. Authentication & Authorization

- ✓ **Basic Authentication** – Supports username/password authentication via **Jetty's authentication** framework.
- ✓ **Kerberos Authentication** – Enables secure authentication via **SPNEGO**.
- ✓ **OAuth & OpenID Connect** – Can integrate with identity providers (e.g., Okta, Keycloak).
- ✓ **Pluggable Authentication** – Custom authentication plugins via **Java-based Solr plugins**.
- ✓ **Role-Based Access Control (RBAC)** – Users and groups can be assigned roles via **Rule-Based Authorization Plugin**.

Risk and Impact Analysis

◆ Example: Enabling Basic Authentication

Enable security.json:
solr auth enable -credentials user:password

1.

Configure security.json:

```
{
  "authentication": {
    "class": "solr.BasicAuthPlugin",
    "credentials": { "solr": "hashed-password" }
  },
  "authorization": {
    "class": "solr.RuleBasedAuthorizationPlugin",
    "permissions": [{ "name": "read", "role": "users" }]
  }
}
```

Risk and Impact Analysis

2. Encryption & Secure Communication

- ✓ **TLS/SSL Encryption** – Encrypts Solr-Jetty communication.
- ✓ **Zookeeper TLS Support** – Encrypts Solr-Zookeeper traffic.
- ✓ **Data at Rest Encryption** – Can be configured at the disk/storage level using external encryption tools.
- ◆ **Example: Enabling SSL in Solr**

Generate keystore:

```
keytool -genkey -alias solr -keyalg RSA -keystore solr-ssl.keystore.jks
```

1. Add SSL settings in `solr.in.sh`:

```
SOLR_SSL_ENABLED=true
```

```
SOLR_SSL_KEY_STORE=solr-ssl.keystore.jks
```

3. Auditing & Logging

- ✓ **Audit Logging** – Track user access and changes.
- ✓ **Access Logs via Jetty** – Logs all user requests to Solr.
- ✓ **Query Logging** – Monitor search queries to detect anomalies.

Risk and Impact Analysis

Kafka 3.9.0 Security Features

1. Authentication & Authorization

- ✓ **SASL Authentication** – Supports **Kerberos (GSSAPI), SCRAM, OAuth, and PLAIN** authentication.
- ✓ **TLS/SSL Authentication** – Secure broker-client communication.
- ✓ **RBAC with Kafka ACLs** – Define read/write permissions for users and groups.
- ◆ **Example: Enabling SASL_SSL with SCRAM**

Add SCRAM users:

```
kafka-configs.sh --zookeeper localhost:2181 --alter --add-config 'SCRAM-SHA-256=[password=secret]' --entity-type users --entity-name alice
```

Risk and Impact Analysis

Configure Kafka server:

```
security.inter.broker.protocol=SASL_SSL
```

```
sasl.enabled.mechanisms=SCRAM-SHA-256
```

```
listener.name.sasl_ssl.scram-sha-
```

```
256.sasl.jaas.config=org.apache.kafka.common.security.scram.ScramLoginModule required
```

```
username="admin" password="admin-secret";
```

2. Data Encryption

✓ **TLS/SSL Encryption** – Encrypts data in transit between **brokers, producers, and consumers.**

✓ **Zookeeper TLS Support** – Secures Kafka-Zookeeper connections.

Risk and Impact Analysis

Example: Enabling SSL in Kafka

Generate certificates:

```
keytool -keystore kafka.keystore.jks -alias kafka -validity 365 -genkey
```

Update `server.properties`:

```
ssl.keystore.location=/path/to/kafka.keystore.jks
```

```
ssl.keystore.password=password
```

```
ssl.enabled.protocols=TLSv1.2,TLSv1.3
```

3. Access Control (ACLs)

- ✓ **Topic-Level Permissions** – Restrict access to **specific topics**.
- ✓ **Consumer Group ACLs** – Limit which consumers can read messages.
- ✓ **Admin ACLs** – Protect **broker operations** from unauthorized access.
- ◆ **Example: Setting Topic ACLs**

```
kafka-acls.sh --authorizer-properties zookeeper.connect=localhost:2181 \  
--add --allow-principal User:alice --operation Read --topic my-secure-topic
```

Risk and Impact Analysis

4. Auditing & Monitoring

- ✓ **Audit Logs** – Track authentication failures.
- ✓ **JMX & Prometheus Metrics** – Monitor security-related events.
- ✓ **Kafka Connect Security** – Supports **encrypted connectors** for external systems.

Risk and Impact Analysis

1. Performance and Resource Requirements: The team is concerned about the performance impact and the resources needed for the Kafka environment, given the large number of fields and documents in their Solr instance.

The below is a good starting point in terms of resource or system requirements for the Kafka production setup.

We need 2 Kafka Clusters with 3 Zookeepers + 5/7 brokers each.

Storage: 12 X 1 TB disk. RAID 10 is optional, Separate OS disks from Apache Kafka® storage

Memory: 64 GB

vCPU 12 cores

Risk and Impact Analysis

Kafka relies heavily on the filesystem for storing and caching messages. All data is immediately written to a persistent log on the filesystem without necessarily flushing to disk. In effect this just means that it is transferred into the kernel's pagecache. A modern OS will happily divert all free memory to disk caching with little performance penalty when the memory is reclaimed. Furthermore, Kafka uses heap space very carefully and does not require setting heap sizes more than 6 GB. This will result in a file system cache of up to 28-30 GB on a 32 GB machine. You need sufficient memory to buffer active readers and writers. You can do a back-of-the-envelope estimate of memory needs by assuming you want to be able to buffer for 30 seconds and compute your memory need as $\text{write_throughput} * 30$. A machine with 64 GB of RAM is a decent choice, but 32 GB machines are not uncommon. Less than 32 GB tends to be counterproductive (you end up needing many, many small machines).

CPU: Dual 12 core sockets

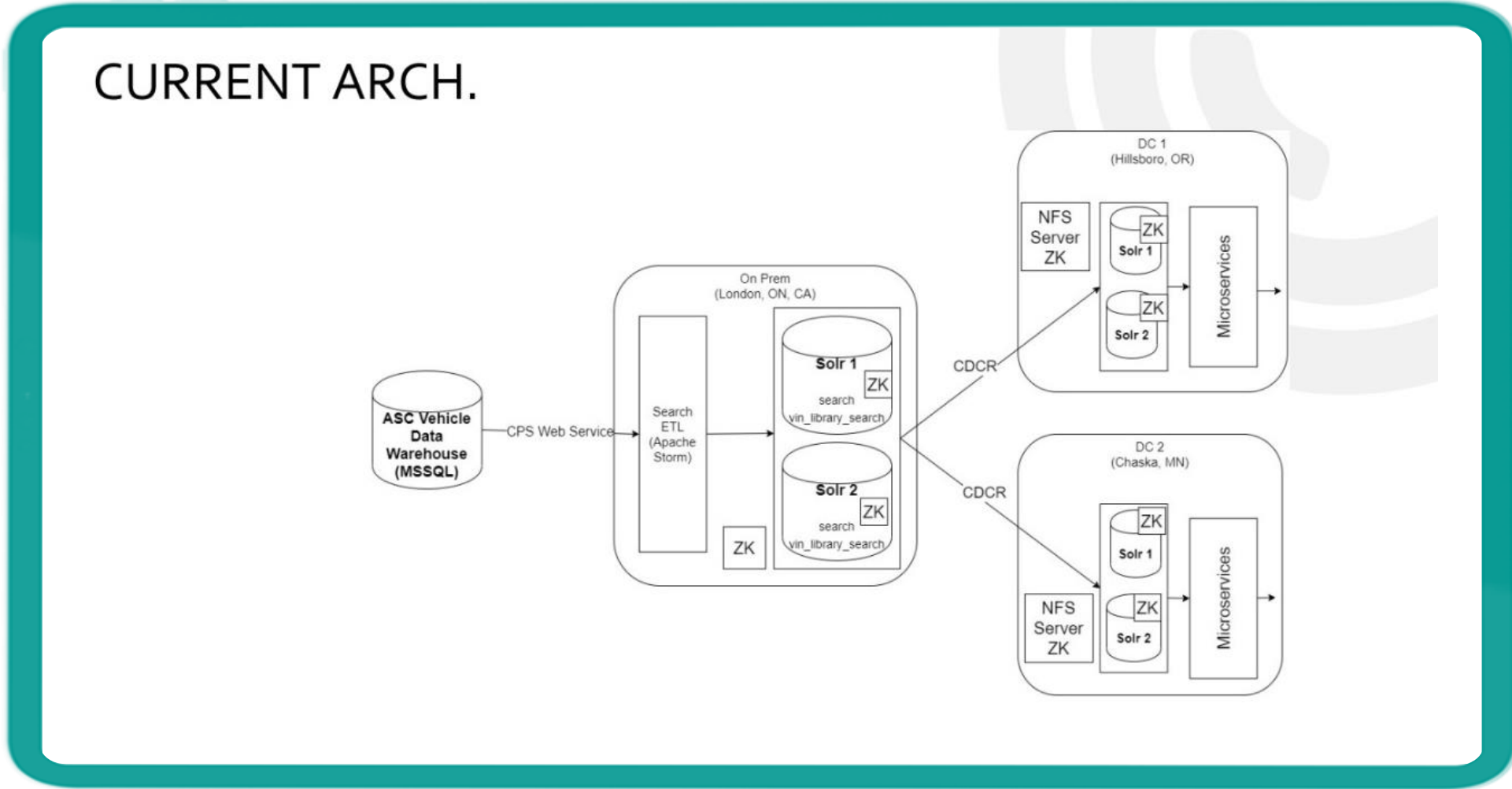
Most Kafka deployments tend to be rather light on CPU requirements. As such, the exact processor setup matters less than the other resources. Note that if [TLS](#) is enabled, the CPU requirements can be significantly higher (the exact details depend on the CPU type and JVM implementation). You should choose a modern processor with multiple cores. Common clusters utilize 24 core machines. If you need to choose between faster CPUs or more cores, choose more cores. The extra concurrency that multiple cores offers will far outweigh a slightly faster clock speed.

Risk and Impact Analysis

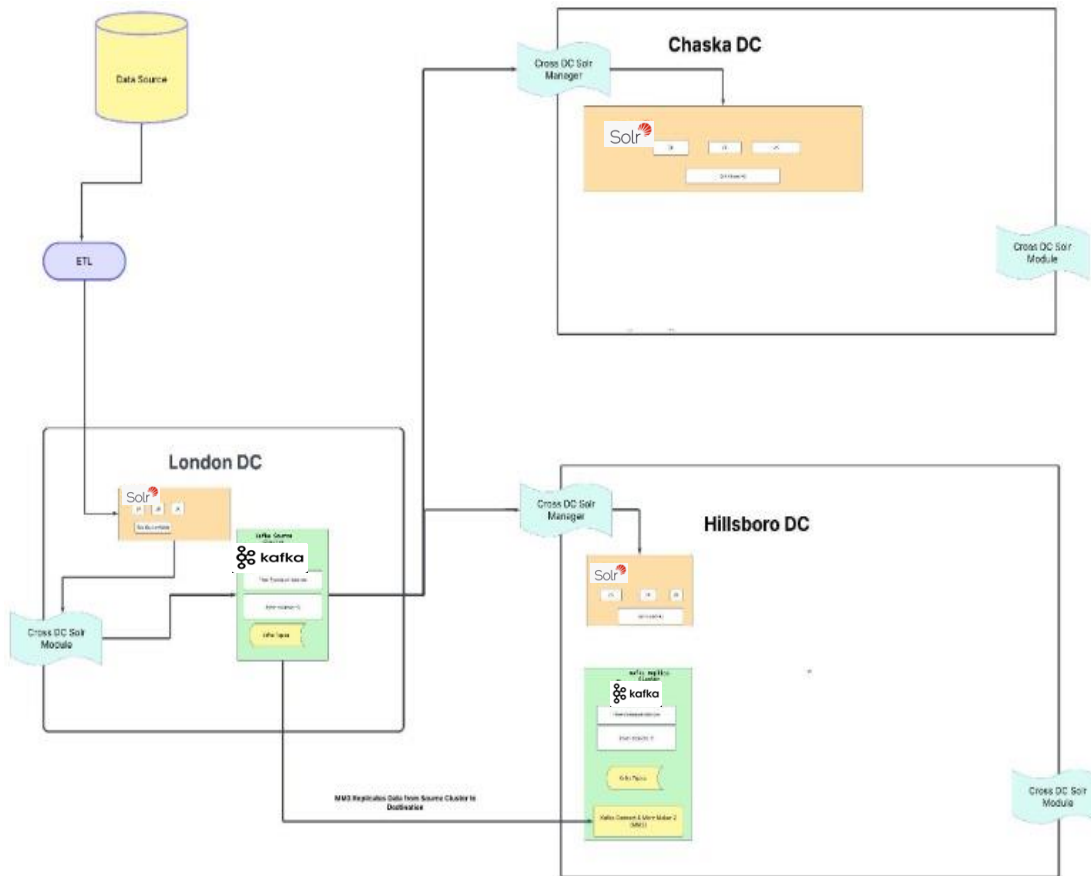
- We can enable batching in the producer to batch multiple records and send them once, this will improve the throughput and reduce the network noise. We can increase the max message bytes on producer and consumer so that we can batch more records when producing.
- We can enable compression to reduce the record size and efficiently use the network.
- If the produce rate is high, we can have more number of partitions for those topics so that the data will be distributed to multiple partitions and shared across the brokers.
- Tune the broker network and IO threads based on the resource utilization and the load.
- Use Prometheus, Grafana monitoring to monitor the cluster, identify the bottlenecks, and further tune the configuration.

To well distribute the load across the partitions and brokers, we can consider using a key & value pair in the record, so that all the records with the same key will end up on the same partition. If using JSON record format, we may see if this is feasible to use JSON Schema, so that we can use Schema Registry service, where we can register our record schema and only produce the record without schema (this will help removing the overhead of sending schema in each record payload). With schema in SchemaRegistry, the producer will serialize the data using the schema and send it to the topic, the consumers will refer to the schemaregistry to fetch the schema and deserialize the record

Current Architecture



New Architecture



Recommended Setup Guidelines:

Source Data Center (DC1):

- Deploy the **Cross-DC Solr Module** to capture updates and publish them to **Kafka**.
- This module monitors the Solr cluster for changes (e.g., document inserts, updates, or deletes).

Target Data Centers (DC2, DC3, etc.):

- Deploy the **Cross-DC Solr Manager** in each target data center.
- The Solr Manager subscribes to the appropriate Kafka topics, consumes messages, and applies the updates to the target Solr cluster.

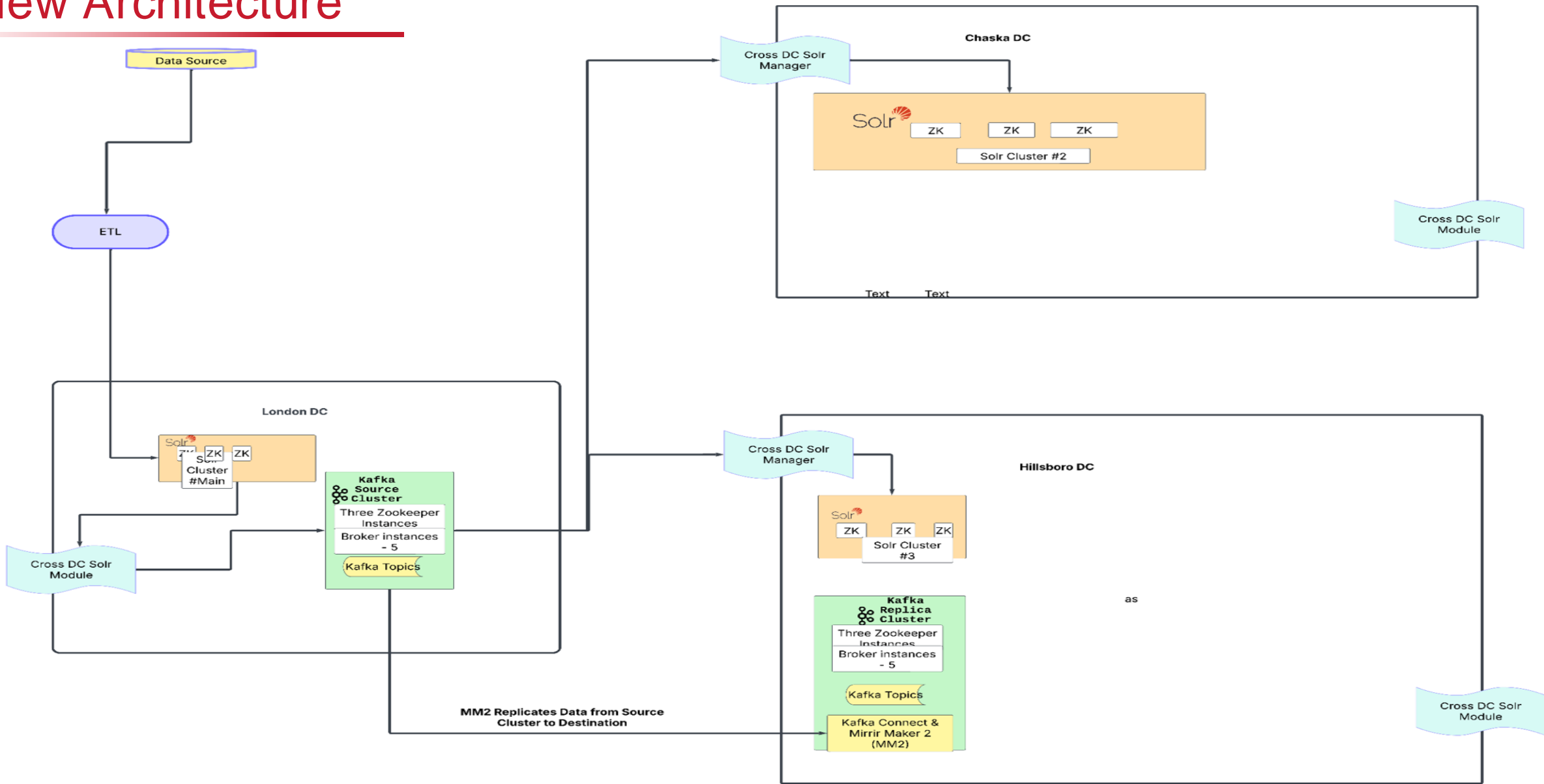
Kafka HA:

- Setup 3 Zookeeper Instances, 3/5/7 Broker instances in the Cluster, with replication factor set to 3 as minimum, min in-sync replicas to 2 to achieve HA.

Kafka DR:

- Setup Kafka Replica Cluster on Target/Backup DC clusters and configure MirrorMaker to Copy the data from source topics to the Replica cluster as a DR solution.

New Architecture



Recommended Setup Guidelines:

Source Data Center (DC1):

- Deploy the **Cross-DC Solr Module** to capture updates and publish them to **Kafka**.
- This module monitors the Solr cluster for changes (e.g., document inserts, updates, or deletes).

Target Data Centers (DC2, DC3, etc.):

- Deploy the **Cross-DC Solr Manager** in each target data center.
- The Solr Manager subscribes to the appropriate Kafka topics, consumes messages, and applies the updates to the target Solr cluster.

Kafka HA:

- Setup 3 Zookeeper Instances, 3/5/7 Broker instances in the Cluster, with replication factor set to 3 as minimum, min in-sync replicas to 2 to achieve HA.

Kafka DR:

- Setup Kafka Replica Cluster on Target/Backup DC clusters and configure MirrorMaker to Copy the data from source topics to the Replica cluster as a DR solution.

New Architecture Description

- It will be Two-tier architecture, which will separate out the Data layer/Search cluster with Application layer.
- Application layer will have multiple App nodes and Java services running on Tomcat.
- Data Layer will have Zookeeper nodes running in AWS and Solr nodes running in On-Prem.
- Solr nodes can have additional security using in-built feature called as BasicAuthentication.
- Two options for communication between Application and Data layer:
 - If Applications use CloudSolrClient then no external Load-Balancer will be required to communicate with Search cluster.
 - If Applications use normal HttpClient then external Load-Balancer should be used to communicate with Search cluster.

Comparison of Current & New Architecture

- Solr Deployment
 - Currently Solr is deployed in Tomcat container shared with other Java services on same VM on JDPA On-Prem.
 - New Architecture will deploy Solr and Java services (in Tomcat) on separate VMs on JDPA On-Prem.
- Zookeeper Deployment
 - Currently Zookeeper cluster is deployed on AWS.
 - New Architecture will keep Zookeeper on AWS similar to current one.
- Communication with Solr
 - Currently Apps are communicating with Solr through Java services using HttpClient method.
 - New Architecture will use CloudSolrClient for communicating with Solr, which will balance the load on Solr nodes automatically.
- Load Balancer
 - Currently Apps are connecting to Java services through Load Balancer.
 - New Architecture will keep Load Balancer between Apps and Java services similar to current one.

Benefits of Tiered Architecture

- Independent VMs will improve performance of each services
- Easy to diagnose the issue if any service don't perform well or suspend or completely went down
- In case of hardware issue only specific service deployed on that VM will get impacted
- Effort on change of hardware and deployment will be less compared to multiple services on same VM
- Scaling up the architecture will be easier, either it is vertical or horizontal

Recommended H/W Spec per Node

	Solr Node	Zookeeper Node
RAM	~ 32 GB	4 GB
HEAP	~ 20 GB	2 GB
CPUs	4	2
DISK	300 – 500 GB	20 – 30 GB

POC Update - NB Custom Code

Environment	Test3 - Source	Test10 - Destination
RAM	16 GB	8 GB
CPU	4	2

Policies collection (policies07-11-2019) documents (232508) ingestion took 2 minutes with below setting

Max-Rows per query - 15,000

Commit time - 60,000 ms

Policies collection (policies06-08-2021) documents (39448) ingestion took 28 seconds with below setting

Max-Rows per query - 10,000

Commit time - 60,000 ms

Activity collection (activity02-18-2021) documents (21091) ingestion took 11 seconds with below setting

Max-Rows per query - 15,000

Commit time - 60,000 ms

POC Update - NB Custom Code

- NextBrick wrote custom code in Java for data migration between Solr collections.
- This code is quite simple and light as its main goal is to just query the data from one collection and ingest in another collection.
- NB code is fast, user friendly and utilizes less CPU.
- NB code generates a jar build which executes easily on command prompt using below parameters. There is not need to deploy on any servlet container.

<Source Solr URL> <Target Solr URL> <Jaas file path> <Source Collection Name> <Target Collection Name> <Filter Query> <Row Count Per Query> <Commit Time> <Output log file name>

Jaas file path and Filter Query are optional so keeping them blank in below example.

```
https://oh1i3sol3:8443/solr http://ae1i10esd1:8983/solr "" activity02-18-2021 activity02-18-2021 "" 15000  
60000 test-migration-activity.log
```

POC Update - NB Custom Code

- NextBrick wrote custom code in Java for data migration between Solr collections.
- This code is quite simple and light as its main goal is to just query the data from one collection and ingest in another collection.
- NB code is fast, user friendly and utilizes less CPU.
- NB code generates a jar build which executes easily on command prompt using below parameters. There is not need to deploy on any servlet container.

<Source Solr URL> <Target Solr URL> <Jaas file path> <Source Collection Name> <Target Collection Name> <Filter Query> <Row Count Per Query> <Commit Time> <Output log file name>

Jaas file path and Filter Query are optional so keeping them blank in below example.

https://oh1i3sol3:8443/solr http://ae1i10esd1:8983/solr "" activity02-18-2021 activity02-18-2021 "" 15000 60000 test-migration-activity.log

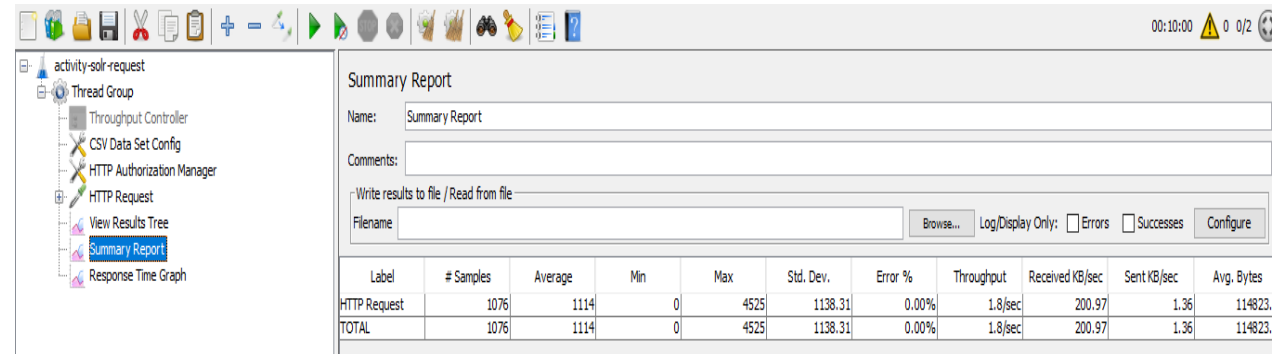
Load Test

1. Load Test Run

1.1. Search Request Details

Duration: 10 min

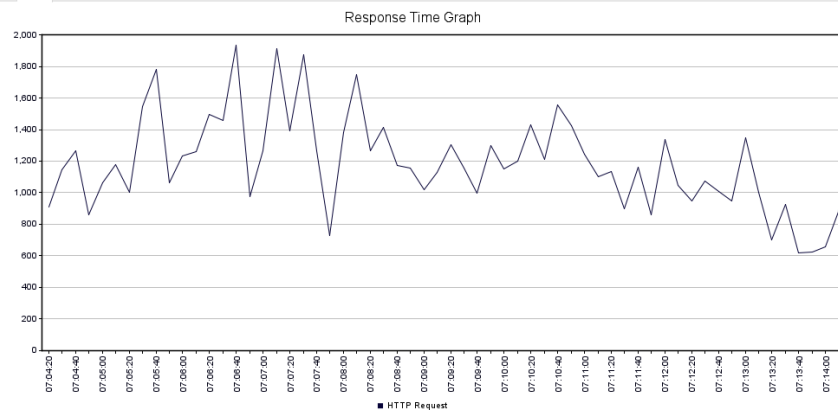
- Throughput 108/RPM
- Average Response Time: 1114 ms
- Min Response Time: 0 ms
- Max Response Time: 4525 ms



The screenshot shows a software interface with a left-hand tree view and a main summary report area. The tree view includes: activity-solr-request, Thread Group, Throughput Controller, CSV Data Set Config, HTTP Authorization Manager, HTTP Request, View Results Tree, Summary Report (highlighted), and Response Time Graph. The main area displays a 'Summary Report' for 'HTTP Request' with the following data:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	1076	1114	0	4525	1138.31	0.00%	1.8/sec	200.97	1.36	114823.
TOTAL	1076	1114	0	4525	1138.31	0.00%	1.8/sec	200.97	1.36	114823.

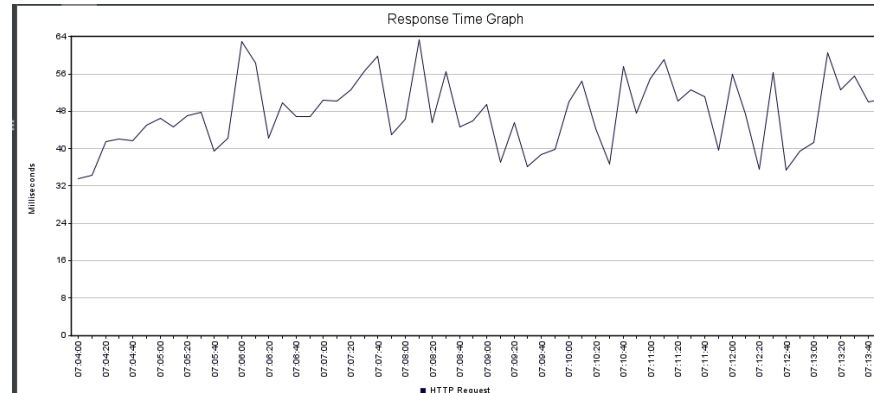
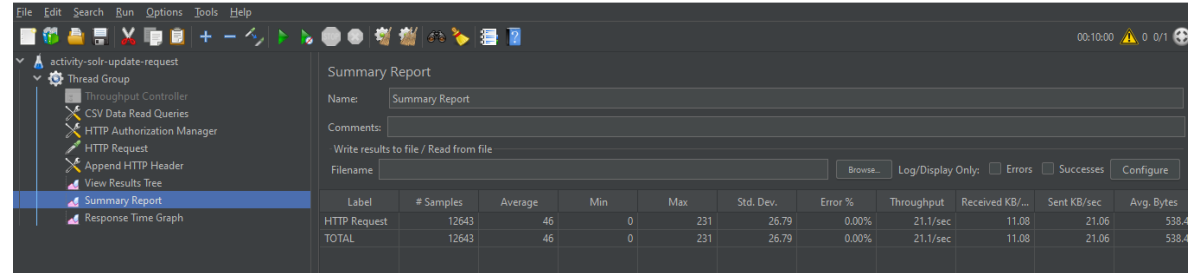
Load Test



1.2. Update Request Details

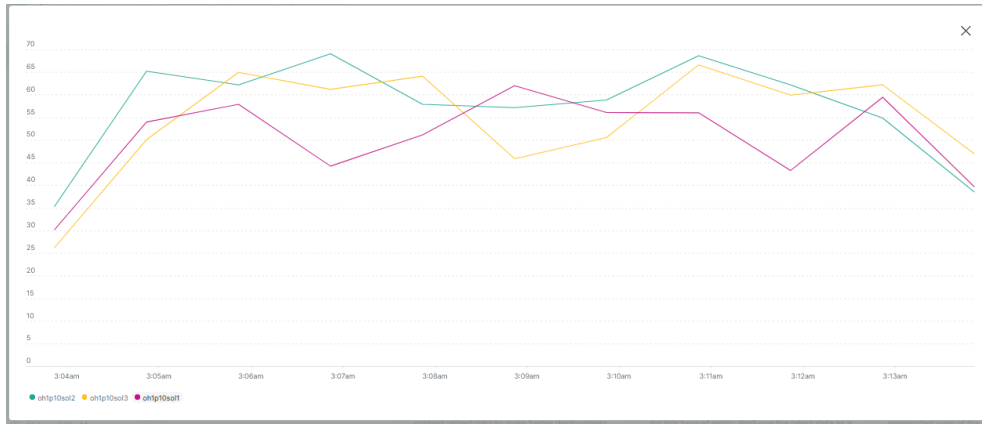
Duration: 10 min

- Throughput 1266/RPM
- Average Response Time: 46 ms
- Min Response Time: 0 ms
- Max Response Time: 231 ms



Load Test

CPU Load Range from 40 to 70



Memory Utilization range from 65 to 70.



Data Migration

Data Migration using NB Custom Code

Datamigration Configuration and parameters

NB Code takes below parameter to execute the datamigration process.

Params passed in NB custom code while migrating data from text3 to test10

```
https://oh1i3sol3:8443/solr http://ae1i10esd1:8983/solr "" activity02-18-2021 activity02-18-2021 ""
```

```
15000 60000 test-migration-activity.log
```

```
https://oh1i3sol3:8443/solr http://ae1i10esd1:8983/solr "" policies07-11-2019 policies07-11-2019 ""
```

```
10000 60000 test-migration- policies07-11-2019.log
```

```
https://oh1i3sol3:8443/solr http://ae1i10esd1:8983/solr "" policies06-08-2021 policies06-08-2021 ""
```

```
10000 60000 test-migration-policies06-08-2021.log
```

Data Migration

Data Migration using NB Custom Code

test3 to test10 datamigration

Stats

MAX_ROWS :: 15000

batchCommitSize :: 60000

*****Source Collection activity02-18-2021 NumFound ::21091::

2023-02-28 08:33:46,969 [main] (SolrReIndex.java:148) - *****Target Collection activity02-18-2021 NumFound
::21091::

2023-02-28 08:33:46,970 [main] (SolrReIndex.java:149) - *****INGESTION PROCESS FINISHED -- Total Records
Processed :: 21091::*****

2023-02-28 08:33:46,970 [main] (SolrReIndex.java:150) - *****Time consumed in Seconds::11::*****

2023-02-28 08:33:46,974 [main] (SolrReIndex.java:151) - *****Time consumed in Minutes::0::*****

Data Migration

Data Migration using NB Custom Code

test-migration-policies07-11-2019.log

```
*****Source Collection policies07-11-2019 NumFound ::232508::  
2023-02-28 08:26:11,059 [main] (SolrReIndex.java:149) - *****Target Collection policies07-11-2019 NumFound ::232509::  
2023-02-28 08:26:11,062 [main] (SolrReIndex.java:150) - *****INGESTION PROCESS FINISHED -- Total Records Processed ::  
232506:*****  
2023-02-28 08:26:11,063 [main] (SolrReIndex.java:151) - *****Time consumed in Seconds::172:*****  
2023-02-28 08:26:11,064 [main] (SolrReIndex.java:152) - *****Time consumed in Minutes::2:*****
```

test-migration-policies06-08-2021.log

```
MAX_ROWS :: 10000  
batchCommitSize :: 60000
```

```
*****Source Collection policies06-08-2021 NumFound ::39448::  
2023-02-28 08:31:19,617 [main] (SolrReIndex.java:148) - *****Target Collection policies06-08-2021 NumFound ::39448::  
2023-02-28 08:31:19,617 [main] (SolrReIndex.java:149) - *****INGESTION PROCESS FINISHED -- Total Records Processed ::  
39448:*****  
2023-02-28 08:31:19,617 [main] (SolrReIndex.java:150) - *****Time consumed in Seconds::28:*****  
2023-02-28 08:31:19,619 [main] (SolrReIndex.java:151) - *****Time consumed in Minutes::0:*****
```

End Point Testing

URL: <https://test-dujour/pas-index-svc/V1/policy>

Body:

Pas-index

```
{  
  "transactionId": "pas_rec_111",  
  "source": "PAS",  
  "id": "test62",  
  "certificate_num": "74271928",  
  "origin_of_record": "PAS",  
  "system_of_record": "PAS",  
  "mi_policy_status": "Canceled Certificate"  
}
```

NB

End Point Testing

The screenshot displays a REST client interface for a POST request. The request URL is `{{index_svc_url}}/pas-index-svc/V1/policy/`. The request body is a JSON object with the following fields:

```
1 {
2   .... "transactionId": "pas_rec_111",
3   .... "source": "PAS",
4   .... "id": "INT10test62",
5   .... "certificate_num": "74271928",
6   .... "origin_of_record": "PAS",
7   .... "system_of_record": "PAS",
8   .... "mi_policy_status": "Canceled Certificate"
```

The response is a JSON object with the following fields:

```
1 {}
2   "statusMessage": "SUCCESS",
3   "statusCode": "200",
4   "transactionId": "pas_rec_111",
5   "solrTransactionId":
6     "ID-ae1110ess2-1685547432949-0-1"
```

The response status is 200 OK, with a response time of 1760 ms and a body size of 297 B.

“Work as a Team, Think as a Team, Problem Solve as a Team”

