AdroitLogic Integration Platform Server

A multi-node container platform for deploying and managing AdroitLogic UltraESB-X
1. Abstract

Based on predictions by Forrester, the B2B e-Commerce market is expected to reach $1.1T in value by year 2019, incorporated by a parallel growth of the B2C market up to $480B.[1] With the ensuing growth of demand for B2B and B2C communication, every enterprise, no matter how small, is met with a strong requirement for its own enterprise integration infrastructure. This, combined with the rapid movement towards service orientation, makes integration products such as Enterprise Service Buses ideal candidates for any enterprise integration framework.

A good balance among enterprise-grade reliability and quality-of-service guarantees, ease of deployment and maintenance, and manageability of expenses, is one of the most sought-after features of an enterprise integration platform, and is unfortunately also one of the greatest hurdles encountered by organizations of all scales, economies and fields of specialization, along their race towards a seamless enterprise integration experience.

It is in this light that AdroitLogic, renowned for its high-performance Enterprise Service Bus UltraESB and its successor UltraESB-X [2], brings forth its Integration Platform Server, a product catered specifically for addressing the abovementioned concerns in the course of deployment of both public-facing and internal enterprise integration solutions as services exposed via containerized UltraESB-X instances.
# Contents

1. **Abstract** ............................................................... 2

2. **Introduction** ........................................................... 5

3. **Project Management and Versioning** ................................ 6
   3.1 Project repository .............................................. 6
   3.2 Configuring a project ......................................... 6
   3.3 Project versions to track changes ......................... 6
   3.4 Browsing project details ................................... 7

4. **Scalability** .............................................................. 8
   4.1 Scaling computational power of the platform ........... 8
   4.2 Vertical scalability of ESB instances .................. 8
   4.3 Scaling availability of ESB instances .................. 8

5. **Fault detection and recovery** ...................................... 10
   5.1 Recovery from a worker node failure ................. 10
   5.2 Recovery from a single instance failure ............ 10

6. **Affinity Deployments** ............................................... 12
   6.1 Affinity via node groups ................................. 12
   6.2 Guaranteed affinity ....................................... 13
   6.3 Dynamic updates ......................................... 13

7. **Cross Data Center Deployment** ................................... 14
   7.1 Fail-over and fail-back for convenience and reliability ............ 14
   7.2 Cross-data-center workload migration ............... 14
   7.3 IPS zones: the all-in-one solution ................... 14

8. **Statistics Through IPS Console** .................................... 16
   8.1 Flexible metrics collection for rich statistical analysis .......... 16
   8.2 Metrics granularity for in-depth analysis ............. 16
   8.3 Built-in metrics visualization .......................... 16

9. **Monitoring and Debugging** ......................................... 18
   9.1 Deployment monitoring .................................... 18
   9.2 Deployment progress bar .................................. 18
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.3 Persistent logs</td>
<td>19</td>
</tr>
<tr>
<td>9.4 Console logs</td>
<td>20</td>
</tr>
<tr>
<td>9.5 Instance snapshots</td>
<td>20</td>
</tr>
<tr>
<td>9.6 Cluster configuration snapshots</td>
<td>20</td>
</tr>
<tr>
<td>10 Conclusion</td>
<td>22</td>
</tr>
<tr>
<td>11 References</td>
<td>23</td>
</tr>
</tbody>
</table>
2. Introduction

Integration between various systems has been a trending topic over the past decade. Enterprise Service Bus (ESB) is the pattern introduced to simplify and effectively address the complexities inherent to the communication among various systems. However, with the rapid growth of the technology and business domains, we can observe a general, steadily growing tendency for cloud-based and containerized deployments in contrast to legacy software installations over traditional, on-premise server-based infrastructures.

While the initial use cases of ESBs were simply limited to the interconnection of different—often legacy—systems, with the general movement towards SOA (service-oriented architecture) and the growing requirements for enterprise integration, enterprises now often need more than a typical ESB for fulfilling their integration requirements. Importance of non-functional requirements such as high availability, convenient deployment, scalability, centralized monitoring and ease of debugging has grown to critical levels, with more and more enterprises turning towards B2B as well as B2C service orientation. Meeting such quality-of-service (QoS) requirements is a major challenge that should be readily dealt with, from the developers’ as well as system deployers’ and administrators’ perspectives.

AdroitLogic Integration Platform Server (IPS) [3] is a solution for meeting all of the abovementioned requirements, while providing a seamless service deployment experience for the developer as well as the system deployment and maintenance teams. It is a multi-node container platform designed to deploy and manage AdroitLogic UltraESB instances in private and public cloud environments. The solution consists of a container orchestration framework that is mainly controlled by Kubernetes [4] and Docker [5], and a console to control the functionalities of the the platform. Inside IPS, UltraESB instances are spawned as lightweight Docker containers that fully utilize the power of containerization such as low resource overhead, high responsiveness and ability to run inside an isolated environment. An IPS deployment may contain one or more physical or virtual machines that drive the actual workload of the platform. Inside these worker machines, IPS spawns and stops ESB containers as required.

![IPS top view architecture](image)

Figure 2.1: IPS top view architecture
3. Project Management and Versioning

Projects are the descendent of Deployment Units used in previous UltraESB versions. Projects contain the actual business integration logic that should be fed into the ESB. In the new version of UltraESB, users can develop Projects easily using Adroitlogic UltraStudio with much sophisticated and convenient ways like developing integration flows using drag and drop editor.

When it comes to organizations that have several integration logics for different systems, dedicated teams and resources for each sub system, there should be a way to effectively manage Projects developed for each sub system. Management can be viewed in several perspectives. Authorization, versioning and ease of accessibility are some of the requirements of above mentioned management.

3.1 Project repository

![Figure 3.1: List of the project available in IPS](image)

IPS comes with an in-built project repository that can store and version projects created by UltraStudio. Users can simply add their Projects to IPS by uploading the built project artifact file and IPS will read it and fetch relevant metadata. You can set the visibility of the Project for a selected set of groups so that only the members of those groups are capable of accessing that particular Project. Depending on their permission level, users inside the authorized groups can perform operations on the Project. For example a user who is in an authorized group and has `edit_project` permission, can edit the uploaded Project. Likewise if he has the `delete_project` permission, he is capable of deleting the project.

3.2 Configuring a project

Usually a Project is built in a configurable manner so that users can change the parameters of the Project without re-building the binary. When a Project is being uploaded, IPS reads the parameters and creates a catalog of external properties for that particular Project. This allows users to easily view and change Project properties through IPS dashboard and IPS makes sure that these property changes are reflected at project deployment phase. If you are having different sets of property values for different phases of deployment such as development, testing and production, IPS provides you the ability to specify deployment environment specific properties too.

3.3 Project versions to track changes

In any Project, changes are inevitable due to possible bug fixes or requirement changes over time. But in some cases, latest version of the updated Project may introduce new set of bugs due to the changes done
on top of previous version. In this case users may need to revert back to a previous version which is the last known stable version. Versioning is one of the powerful aspects of IPS when it comes to the change management of Projects. IPS allows users to keep multiple versions of same Project and also allows to deploy any Project version according to their choice.

3.4 Browsing project details
You can browse the content of an uploaded project through the project browser. There you can view externalized properties and integration flows associated with the project. Through Integration Flow Viewer you can see the graphical representation of Integration Flows of the Project and associated properties of each component in the Integration Flow.
4. Scalability

Scalability is one of the powerful features of IPS. We provide scalability of the platform in different levels.

4.1 Scaling computational power of the platform

IPS consists of one or more physical or virtual machines that act as worker machines so that ESBs are deployed in these machines. There is a practical limitation in number of ESB instances that can be deployed on the worker machines depending on their capacity and computational power. If you need more ESBs to be run on your platform, you can stack up more worker machines in the platform. If you are having excess resources than your requirement, you can simply disconnect worker machines from the platform. It is not required to restart or pause the platform to scale up or down the worker machines. Platform will automatically detect any addition or loss of the worker machines and adopt according to situation. For example if one worker machine goes down, platform will detect it and schedule the ESBs deployed on that worker machine on another worker of the platform.

![Scaling worker nodes](image)

**Figure 4.1: Scaling worker nodes**

4.2 Vertical scalability of ESB instances

In an enterprise deployment, enterprise may have different projects that may require resources in different levels. Some projects may consume more memory while some are more CPU intensive. Depending on the requirement of the project, you can change the memory and CPU consumption thresholds of the ESBs running inside IPS. If you feel that the Project requires more resources than you have presumed, you can change the minimum and maximum resource limits for ESB instances even at runtime and IPS will take care of deploying them in a suitable manner. In case you have requested more resources than the platform can afford to, for example if you requested 4GB of minimum memory although any of the workers inside IPS does not have free 4GB space in memory, platform will keep the instance in pending state until it gets enough resources.

4.3 Scaling availability of ESB instances

UltraESB is inherently scalable and the purpose of scaling is to provide high availability so that the failure on one instance does not affect the system as there are other instances that can take the responsibility of the failed member. Although scalability is possible, there should be some entity who performs the actual scaling operation. IPS takes the responsibility of scaling ESB instances within the platform according to the requirements of the user. This task can be performed in a way that scaling up or down operations does...
not affect to the behaviour already running instances. Typically when a scaling up operation is performed, platform prefers worker machines with low resource consumption to be selected as candidates to deploy new ESB instances. Alternatively if you want to scale your ESB instances in a specific set of worker machines, platform does provide that capability too. Scaling can be performed in two methods. Either by fully removing old set of instances and re-creating them from the beginning or doing a rolling update which removes old instances one by one and creates a new instance for each removed instance at the same time. Former method introduces a small down time in between the transition but it guarantees the consistency of the deployment at any point of the time while latter does not introduce any down time but consistency will be compromised at the transition time as there could be both old and new instances running at that time.

Figure 4.2: Scaling running ESB instances
5. Fault detection and recovery

A production level system should be able to run not only on happy day scenario, but it should be able to remain stable or recover in case of an unexpected issue too.

5.1 Recovery from a worker node failure

There could be scenarios where IPS may lose the connectivity of the worker nodes registered in the platform due to a node crash or a network split. Once the connectivity has lost with the platform, IPS marks the worker machine as not working and tries to handover the responsibility of the crashed worker node to an available worker node.

![Figure 5.1: Recovery of a worker node](image)

5.2 Recovery from a single instance failure

Once we deploy an ESB cluster inside IPS, IPS continuously monitors the health of the ESB instances within the platform. If any instance crashes due to any unexpected issue, IPS automatically creates a new instance to act as the old one and makes sure that the new instance is almost same to the old one from configurations and the functionality.
Figure 5.2: Recovery of a single instance
6. Affinity Deployments

6.1 Affinity via node groups

Situations may arise where you want to give preference to some of the computing nodes in your infrastructure, when it comes to the deployment of certain services. For example, it makes more sense to deploy a project having an integration logic incorporating highly memory-intensive computations, on a work node with higher memory capacity, whereas a CPU-intensive operation should usually be deployed on a multi-core server whenever possible. However, often such preferences should not be made overly restrictive so as not to disrupt the availability of the service; for example, if the multi-core server goes offline, the service should automatically get migrated to the next best candidate in the server group, rather than being taken offline altogether.

IPS provides a specific feature called node groups to provide the affinity aspect for such scenarios. A node group is a collection of worker nodes inside the platform: for example, a set of machines with large memory capacity as discussed in the previous scenario. A cluster can be defined to span across one or more of these node groups, which is an indication to the platform that, at runtime, each worker pod created under that cluster should be spawned only inside one of the nodes belonging to the aggregated collection of nodes from the assigned node groups.

![Image of node groups]

Figure 6.1: Instances of cluster X are spanned only in node group A and B

For example, assume that there is a node group A with nodes 1 and 2, both having 32 CPU cores, another node group B with node 3 having 64 CPU cores, and a third node group C having 3 machines each with only 8 CPU cores. If you define a cluster X to span over node groups A and B, all instances spawned under X would fall into one of the nodes 1, 2 and 3. This effectively ensures that every instance would be run on a
node with at least 32 CPU cores. Now, if you have a project P that extensively utilizes multi-core capabilities for parallel processing, you would be able to gain a significant speedup by deploying P on cluster X rather than any other cluster.

6.2 Guaranteed affinity

Node groups guarantee that the instances of the cluster would always be deployed in one of the specified nodes. If none of the nodes are available, the instance would remain in unstarted (pending) stage until at least one node becomes available for it to be scheduled inside. This guarantees that the abovementioned project P is never deployed on one of the inferior nodes.

In cases where you want to ensure the availability of the service even when the specific high-performance nodes are unavailable, you can expand the cluster to include group C as well. This would give priority to the high-performance nodes whenever they are available (as per the underlying scheduling configuration) but still make the project available via a 8-core instance when both node groups A and B are either offline or already fully occupied in terms of resources.

6.3 Dynamic updates

While the hardware-agnostic nature of IPS allows you to add to and remove any number of instances from the platform at any time, node groups can also be updated parallely to reflect the changes. For example, in the abovementioned scenario, if you receive another 64-core machine for your IT infrastructure, you can plug it into IPS right away and update node group B to include it as well. This can be done with or without restarting the clusters associated with the node group, at your discretion; if you decide not to restart the clusters, the current instance layout will remain unchanged, and the new node will be made available for the cluster right after the next reconfiguration of the cluster (e.g. as part of the next redeployment).
7. Cross Data Center Deployment

7.1 Fail-over and fail-back for convenience and reliability

As much as high-availability is essential for meeting the QoS (quality-of-service) guaranties such as low latency and high throughput, fail-over is essential for the health and reliability of a service deployment. This becomes particularly useful in case of multi-datacenter deployment, where the secondary data centers (SDCs) should take over as soon as the primary data center (PDC) becomes unavailable (which may be due to a variety of reasons ranging from a temporary network partitioning to a complete annihilation due to a natural disaster), within a guaranteed maximum delay. This can come handy not only in emergencies but also as part of the maintenance process, where you can simply shut down the PDC (or disconnect it from the overall network) and be confident that the workload will be handled by the SDC until the PDC returns online, ready to serve again, at which point the system would fail back to PDC.

7.2 Cross-data-center workload migration

In addition, in situations where the PDC is overloaded, SDCs may often be required to automatically share a part of the resource load, spinning up its own set of worker pods for handling further incoming requests. This can be considered as a special case of fail-over, considering the fact that the PDC becomes "partially unavailable" as it may not be able to continue catering for future workloads by scaling up its worker pod population. Lack of such a "watch my back" capability may lead to situations where the SDC keeps on idling passively while the PDC is struggling to handle the entire workload, severely affecting the overall QoS of the system and dragging down the efficiency in terms of resource utilization versus their availability.

7.3 IPS zones: the all-in-one solution

IPS provisions both of these requirements via a unified concept of zones. A zone is a logical grouping of worker nodes with a particular label, and the management console allows configuration of zones via label assignment to individual nodes. A platform-level configuration can be introduced to define which nodes

Figure 7.1: All the instances are scheduled in PDC while it is active
have a higher affinity for scheduling of containers, based on a label-based weighted scoring scheme. At runtime, nodes assigned with higher-ranked labels are given higher priority in scheduling service containers. In addition, based on resource limits assigned to each container and total resource capacities of individual nodes, IPS automatically migrates newly spawned containers to lesser-priority nodes eventually.

For example, if you assign nodes 1 and 2 with PDC label (higher priority) and nodes 3 and 4 with SDC label, containers will be scheduled in nodes 1 and 2 whenever they are available and have sufficient remaining resource capacities. If both 1 and 2 go offline or both of them are scarce of resources, new containers will be spawned automatically in the SDC (nodes 3 and 4). When the PDC becomes available again (in accessibility as well as resource aspects) the process would automatically driven in the reverse direction, induced by the priority-based scheduling mechanism.

This in-built automatic fail-over capability enhances the suitability of IPS for self-managed and robust deployment scenarios across multiple zones and data centers.
8. Statistics Through IPS Console

In any SOA deployment, no matter how big or small, statistics play a crucial part in the management and monitoring process. Statistics can be useful in a variety of situations ranging from identification of resource usage patterns of the deployed system components to identification of critical issues of the system. As such, collection and persistence of as much metrics as possible from the deployment can always come handy in the long run.

8.1 Flexible metrics collection for rich statistical analysis

UltraESB has had a notable history of comprehensive metrics, and the same is true for the tailored worker node offered via IPS. Metrics are reported to Elasticsearch which, as a schema-free document store, allows much flexibility in subsequent manipulation of collected data for generating a virtually unlimited range of statistics and reports. For example, in a deployment of a memory-intensive workflow, you can analyze the memory usage of the entire cluster (aggregation) as well as of individual instances, throughout the course of an entire month or just the last 5 minutes, in order to understand variation of memory usage under different workloads.

8.2 Metrics granularity for in-depth analysis

In addition to system metrics of the container, including CPU usage, memory usage, active thread count and open file descriptor count, worker pods in IPS also report message-level metrics such as the number of received and sent messages, number of errors encountered, etc. Statistics are associated with identities of the corresponding project and integration flow, which can become handy when you want to analyze message reception on a project- or integration flow-level.

8.3 Built-in metrics visualization

![Figure 8.1: System statistics of a ESB instance inside IPS](image)

In addition to comprehensive metrics reporting, IPS allows you to query for commonly used statistics right through the dashboard. The Cluster Statistics perspective displays a summary of system and messaging statistics of a given cluster over time whereas the Instance Statistics perspective offers the same from the perspective of a running worker pod instance. The summary can be expanded or contracted to a time interval.
Figure 8.2: Communicational statistics of an ESB instance inside IPS

of your choosing for a better overview of generic versus specific situations. Additionally, for cases where certain instances of a cluster may seem to be utilizing more resources than the rest, the statistics viewer can also show you the top few instances (in terms of aggregated resource usage) within a given time period, which can be useful as part of further analysis for a possible issue.
9. Monitoring and Debugging

Even the most well-planned, structured and robust deployment may fail due to various causes, ranging from a simple invalid line of configuration to an unhandled exception encountered during a high load, or a concurrency issue in a faulty corner-case piece of logic. Hence monitoring, debugging and troubleshooting are basic requirements of any deployment, and hence have become inherent features of all enterprise-grade integration solutions. IPS is no exception, with its comprehensive array of monitoring and troubleshooting utilities made available at different levels of granularity.

9.1 Deployment monitoring

Most of the issues arise right along the deployment process, due to reasons ranging from configuration errors to rare platform-level faults. For all such cases, IPS provides a deployment monitoring component that monitors a cluster during deployment and alerts the user in case of any failure in the process. The monitoring indicator is available in all perspectives where a cluster configuration can be updated, i.e. in Cluster Configurations and Cluster Deployment, and continues to display the status of the cluster until it reaches a stable state in the deployment process (which may either be a successful completion or a failure).

In addition to the visual indicator, IPS continuously monitors every deployed cluster in the background. If a deployment failure is observed at any point, the corresponding cluster is flagged as failed, for the attention of the system administrators. Similarly, if a previously failed cluster eventually gets deployed successfully, its status is updated to indicate the fact.

![Deployment Monitor and Progress Bar](image)

**Figure 9.1: Deployment Monitor and Progress Bar**

9.2 Deployment progress bar

The deployment process in IPS involves several discrete steps, such as the scheduling of Docker containers, fetching necessary configurations, starting the worker processes and deploying the assigned projects. Fur-
thermore, in case of a deployment failure, IPS inherently retries the deployment by re-spawning any failed instances, which then go through the abovementioned steps completely independent of each other and of any instances that are already running successfully.

This gives rise to a strong requirement from the system administrators’ perspective to observe the current deployment status of the cluster at a given point. This can generally provide hints as to why a deployment process may keep on failing consistently, and help to identify bottlenecks that may be delaying the deployment, such as slow network connectivity when pulling new images, delays in pod scheduling due to resource contention, or long ESB startup time due to a configuration issue leading to a timeout. It can also serve as a confirmation that the required number of instances are up and running healthily at a given point in time.

IPS caters this exact requirement via a deployment progress bar component on the Cluster Deployment perspective. The progress bar is updated continuously, parallel to but logically independent of the back-end deployment process, indicating statuses of deployment stages such as the number of pods currently scheduled and started, number of ESB processes successfully started and the status of deployment of different projects assigned to the cluster. The progress is updated regardless of whether the deployment process has reached a stable state or not, hence serving also as an indicator of the current cluster.

### 9.3 Persistent logs

Perhaps the most sought-after resource for troubleshooting a deployment is application logs. This is particularly true in case of UltraESB-X, as almost any anomaly observed in its runtime environment would be recorded as a log trail, which can be invaluable in investigating any subsequent issue, may it be a simple processing logic error or a complete system failure.

However, since ESB instances in a containerized environment run as spontaneous pods, which have no long-term state and leave no trace once they shut down, obtaining historical logs for troubleshooting can be problematic. Moreover, given that a cluster can potentially consist of dozens or even hundreds of identical pods hosting multiple projects, and an IPS instance may be running several such clusters simultaneously, a mechanism to filter any accumulated logs based on different perspectives such as the cluster, instance ID and project can immensely ease the life of an investigation team.

IPS fulfills this requirement by publishing logs from all ESB instances to a central database for long-term persistence. It also provides you with a full-fledged log viewer right inside the management console. The

![Figure 9.2: Log viewer of IPS to browse persistent logs ESB instances](image_url)

IPS fulfills this requirement by publishing logs from all ESB instances to a central database for long-term persistence. It also provides you with a full-fledged log viewer right inside the management console. The
viewer allows you to filter logs based on different aspects such as cluster name, project name and version, and server name label assigned to the instance. It also allows you to view the logs of a specific ESB instance based on the ID of the corresponding pod, which can be useful in investigating an issue that has occurred in only a particular instance of a large ESB cluster. Different combinations of these filters can be applied to narrow down the log scope to just the problem you are interested in.

The log viewer also allows filtering logs based on more generic aspects like log level and occurrence time, and sorting logs based on a variety of fields including time, originating component name and error code. It is also possible to filter out logs from all terminated instances, useful in most cases where the instance facing the issue is still in running state. Once you have narrowed down the log scope using all of the above filters, you can download the filtered logs as a CSV file for further analysis.

9.4 Console logs

Sometimes it can be useful to monitor a real-time stream of logs as they are generated, particularly in cases where an instance is failing at startup. IPS also offers this feature as a real-time console log viewer, available for ESB instances as well as IPS platform-level management components, via an Instance Console Logs perspective and a Platform Console Logs perspective respectively. These logs are periodically updated to reflect the latest log lines generated by each instance, and in case of a restart of an instance, provisions are made to view the logs of the previous “incarnation”, which can be quite useful in cases where an instance keeps on restarting due to some critical configuration error.

![Console out of a currently running ESB inside IPS](image)

9.5 Instance snapshots

Sometimes it may be necessary to obtain the internal configurations and projects of a running ESB instance in order to reproduce and deeply analyze an issue. For example, if a certain project configuration may be preventing a project from starting up, or a certain combination of projects may lead to a failure in the container. In such cases, accurately reproducing the scenario will be much easier once you get your hands on the actual cumulative configuration that is running inside the instance. This exact feature is offered by IPS by the same name, upon invocation of which the configuration, logs and projects directories of the corresponding ESB instance will be made available as a downloadable archive.

9.6 Cluster configuration snapshots

Similar to instance snapshots, sometimes it may be useful to take a snapshot of the entire cluster, for archival purposes as well as for investigating possible configuration issues. IPS provides this feature in the form of a configuration archival, where a downloadable archive is generated with the platform-level configuration
of deployment and service exposure configurations, along with platform-level configurations and runtime snapshots of all currently available pods of the cluster.
10. Conclusion

With the gaining popularity of service orientation, the general movement towards cloud-oriented deployments and the growing benefits of containerization, the role of an ESB in an organizational IT infrastructure is evolving rapidly. With increasing B2B and B2C communication demand, accompanied simultaneously by requirements for stringent QoS guarantees and convenience in the overall devops lifecycle, modelling the ESB concept for cloud-based deployments has become an absolute necessity, regardless of the nature and scale of the enterprise.

As described in this article, AdroitLogic Integration Platform Server (IPS) is capable of successfully addressing most of the requirements and challenges encountered during an enterprise-level migration to a container-based ESB (UltraESB) deployment. IPS is specifically crafted for deploying AdroitLogic’s high performance UltraESB-X instances as lightweight Docker containers on both public and private cloud environments, enhanced with features such as rapid self-healing, automatic fail-over and multi-faceted, zero-downtime scalability.

Combining the efficiency and flexibility of containerization with the reliability and QoS guarantees offered by enterprise-grade orchestration frameworks, wrapped in a devops-friendly management interface covering all essential aspects of a production-grade ESB deployment including simplified versioning, in-built load balancing, comprehensive statistics, real-time monitoring and in-depth troubleshooting, IPS promises to deliver a smooth deployment experience for any enterprise looking forward to lay their hands on a robust, scalable, performant and easy-to-use integration environment based on the UltraESB-X.
11. References

4. https://kubernetes.io
5. https://www.docker.com