



TIBCO BusinessWorks™ Container Edition

Performance Benchmark & Tuning Guide

Version 2.9.2 | August 2024

Contents

Contents	2
Changing Help Preferences	6
Overview	8
TIBCO BusinessWorks Container Edition Architecture	10
Performance Benchmark Fundamentals	11
Interpreting Benchmarks	12
Misleading Experiments	13
Test Client Limitations	13
Points to Remember	14
Benchmarking and Testing Performance	15
Performance Benchmarking Process	15
Performance Benchmarking Criteria	15
Performance Testing Tools and Techniques	16
Collecting Performance Data	17
Deploying Performance Testing Framework	17
Developing a Performance Testing Plan	18
Build a Baseline Test	18
Compare Baseline to Targets	18
Build Stability Test	19
Develop Incremental Tests	19
Develop Peak Rate Tests	19
Develop Steady State Tests	20
Develop Resource Plan	20
Develop Component Deployment Plan	20

Monitoring and Analyzing TIBCO BusinessWorks Container Edition Components ..	20
JVisualVM ..	21
Monitoring Threads and Taking a Thread Dump Using JVisualVM ..	21
Understanding Thread Dumps ..	27
Identifying Potential Improvement Areas ..	27
Implementing Specific Enhancements ..	28
Comparing Results ..	28
Setting JVM Parameters ..	30
JVM Parameters ..	30
Heap Space ..	30
Heap Dump On Out of Memory Error ..	31
Best Practices ..	32
Engine Tuning Guidelines ..	33
ThreadCount (bw_engine_threadcount) ..	33
StepCount (bw_engine_stepcount) ..	34
Flow Limit ..	35
Application Statistics ..	37
Process Statistics ..	37
Process Execution Statistics ..	37
Activity Instance Statistics ..	38
JVM Tuning Guidelines ..	39
Specifying JVM Heap Size ..	39
JVM Garbage Collection ..	40
Transport and Resource Tuning Guidelines ..	42
HTTP Resource ..	42
HTTP Client Resource ..	44
JMS Resource and JMS Transport ..	45
Impact of SSL on Performance ..	46

Container Tuning Guidelines	47
Horizontal Scaling	47
Vertical Scaling	48
Tuning Parameters	51
HTTP Connector Resource	51
HTTP Client Resource Tuning Parameters	53
JDBC Connection Resource	54
TCP Connection Resource	55
JMS Receiver	56
Blocking Queue Size	56
Debugging Performance Issues	58
Debugging High CPU Utilization Issues	58
Debugging High Memory Utilization Issues	59
Debugging High Latency Issues	61
Performance Improvement Use Cases	63
Performance Improvement Use Cases - Design Time and Deployment	63
Usecase 1: Using File as the Input Type for Parse Data Activity	63
Usecase 2: Schema changes for improved performance	65
Using XSD Schema Type for the Parse JSON activity	67
Usecase 4: Changing XSLT Version to Improve Latency	68
Usecase 5: Repetition Count Tuning for XML Authentication Policy	69
Tools for Memory Monitoring, Tracking, and Analysis	71
TOP Command for Memory Monitoring	71
Native Memory Tracking	72
Jemalloc and Jemalloc	73
Detecting Increase in Heap Allocations with UMDH	75
Memory Saving Mode	79
Performance Use Case - Memory Optimization	79

References	82
TIBCO Documentation and Support Services	83
Legal and Third-Party Notices	85

Changing Help Preferences

By default, documentation access from TIBCO Business Studio™ for BusinessWorks™ is online, through the [TIBCO Product Documentation](https://docs.tibco.com/) website that contains the latest version of the documentation. Check the website frequently for updates. To access the product documentation offline, download the documentation to a local directory or an internal web server and then change the help preferences in TIBCO Business Studio for BusinessWorks.

Before you begin

Before changing the help preferences to access documentation locally or from an internal web server, download the documentation.

1. Go to <https://docs.tibco.com/>
2. In the **Search** field, enter TIBCO ActiveMatrix BusinessWorks™ and press **Enter**.
3. Select the TIBCO ActiveMatrix BusinessWorks™ product from the list. This opens the product documentation page for the latest version.
4. Click **Download All**.
5. A compressed .zip file containing the latest documentation is downloaded to your web browser's default download location.
6. Copy the .zip file to a local directory or to an internal web server and unzip the file.

To point to a custom location:

Procedure

1. Perform one of the following steps in TIBCO Business Studio for BusinessWorks based on your operating system:
 - On Windows OS: Click **Window > Preferences**
 - On macOS: Click **TIBCO Business Studio > Preferences**.
2. In the Preferences dialog, click **BusinessWorks > Help**.
3. Click **Custom Location**, and then browse to the `html` directory in the folder where you extracted the documentation or provide the URL to the `html` directory on your

internal web server.

4. Click **Apply**, and then click **OK**.

Overview

TIBCO BusinessWorks™ Container Edition allows customers to apply the power and functionality of TIBCO BusinessWorks™ Container Edition in order to build cloud-native applications with an API-first approach and deploy it to container-based PaaS platforms such as Cloud Foundry™, Kubernetes, Kubernetes, and Openshift or to similar Docker-supported cloud platforms.

TIBCO BusinessWorks Container Edition enables you to create services and integrate applications using a visual, model-driven development environment, and then deploy them in the TIBCO BusinessWorks Container Edition runtime. It uses the Eclipse graphical user interface (GUI) provided by TIBCO Business Studio™ for BusinessWorks™ to define business processes and generate deployable artifacts in the form of archive files. For more information, see the *TIBCO BusinessWorks Container Edition Concepts* guide for additional details.

Performance plays a very important role in terms of stability, scalability, throughput, latency, and resource utilization. With a view to achieve optimal performance of the TIBCO BusinessWorks Container Edition application, it is important to understand the various levels at which the tuning methods and best practices can be applied to the components.

The intent of the *TIBCO BusinessWorks Container Edition Performance Benchmarking and Tuning* guide is to provide guidelines with respect to performance benchmarking, tuning methodologies and best practices. This document must be used along with other product documentation and project-specific information to achieve the desired performance results. The goal is to assist in tuning and optimizing the runtime for most common scenarios.

This document describes architectural concepts related to performance tuning for TIBCO BusinessWorks Container Edition. The document includes the different tuning parameters, steps required to configure the parameters, and design techniques for better performance. However, you must TIBCO FOCUS® on real customer use cases to understand the issue and the associated solution.

i Note: The performance tuning and configurations in this document is okprovided for reference only. They can be reproduced only in the exact environment and under workload conditions that existed when the tests were done. The numbers in the document are based on the tests conducted in the performance lab for TIBCO BusinessWorks Container Edition 2.5.3 and may vary according to the components installed, the workload, the type and complexity of different scenarios, hardware and software configuration, and so on. The performance tuning and configurations should be used only as a guideline, after validating the customer requirements and environment. TIBCO does not guarantee their accuracy.

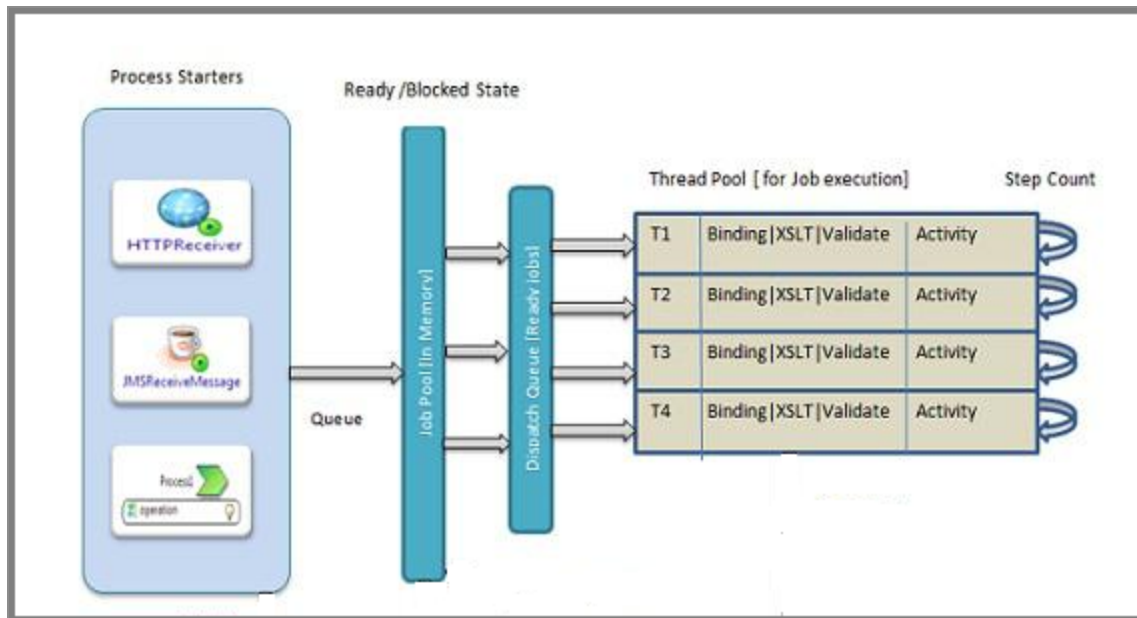
TIBCO BusinessWorks Container Edition Architecture

The most important component in TIBCO BusinessWorks Container Edition is BWEngine. It runs inside a container either on a cloud foundry environment or on a docker based platform. The purpose of BWEngine is to handle a continuous stream of thousands of processes, each with dozens of activities, in an operating environment with finite resources. Resources include the memory, CPU threads, and connections.

The BWEngine performs the following additional functions:

- XML and JSON data transformation and validation
- XPath transitions and flow control
- Connection and session management with recovery and retries
- Exception management and logging
- Management and monitoring services

Message Flow Architecture

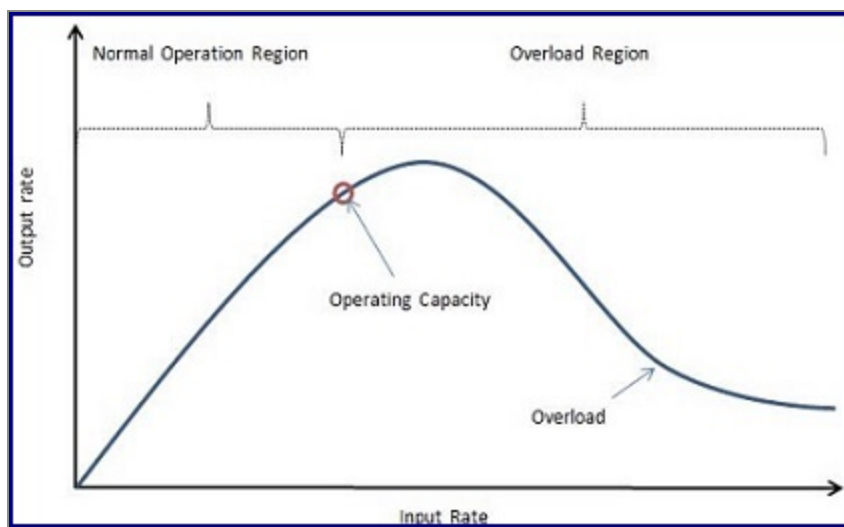


Performance Benchmark Fundamentals

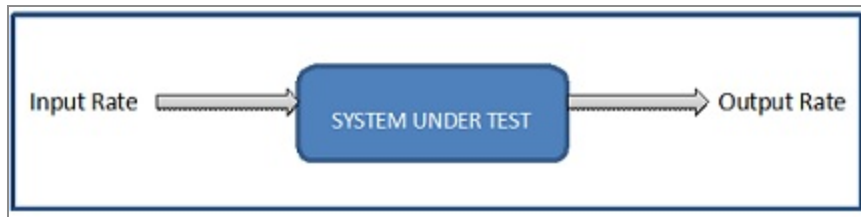
The goal of performance measurement is to understand the performance capabilities and limitations of a system. Every system has limitations, and benchmarks characterize the system in a way that you can understand these limitations.

Benchmarks can be complicated if the system capabilities and limitations vary depending on the demands placed on the system. They also vary based on the resources that are available to the system, for example, CPU, memory, network bandwidth, and disk bandwidth. The set of benchmark measurements must be carefully designed so that the impact of these factors can be understood clearly .

Basic Performance Curve



In the above example, the X axis characterizes the input rate, and the Y axis represents the output rate. The system is exposed to load at a controlled rate and is in a steady-state for some period of time. After an initial stabilization period, both the input and output rates are measured, providing one data point for the performance curve. This example assumes that all other factors are being held constant.



The shape of the performance curve tells us a lot about the system under test. If each input results in a single output, then over the normal operating range, the output rate exactly matches the input rate and within statistical bounds. If each input results in more than one output, or if you are measuring data rates instead of input and output rates, there may be a scale factor relating the two values. However, over the normal operating range there must be a linear relationship between the two rates – that is the curve is a straight line.

The input rate that marks the end of this linear region marks the operating capacity of the system. This may mark the true limits of the system design, or it may indicate that some type of resource limit has been hit. This could be the result of the available physical memory or the bandwidth available on the NIC card or the available CPU cycles getting exhausted. It is important to determine the nature of the limit, as this may indicate ways to alter the environment and increase capacity either by tuning the system or adding resources.

Beyond the operating capacity, further increase in the input rate exceeds the capacity of the system to perform work. Once this occurs, increasing the input rate does not produce the same level of increase in the output. The inputs are increasing faster than the system is producing output. If the input rate continues to increase it reaches a point where the output rate begins to decline. The system is taking resources away from completing work and applying them to accepting the inputs.

Operating under a load is inherently unstable. Inputs arrive faster than the work is getting completed, and this results in inputs piling up. This buffer for memory, disk, and messaging system is finite in capacity. At full capacity the system fails. Thus systems can, at best, operate in the overload mode for short periods of time.

Interpreting Benchmarks

Each benchmark measurement provides a single data point on the performance curve. In order to meaningfully interpret that benchmark you must understand where you are on the

curve.

Failure to understand your position on the curve can lead to significant misinterpretation of data.

Misleading Experiments

One of the most commonly run performance tests is when a large but fixed number of inputs are applied at the fastest possible rate, often by placing them in an input queue and then turning the system on. The output rate is often misinterpreted as the system capacity.

However, if you look at the performance curve, it is likely that the system is actually operating far into the overload region with an output rate significantly below the operating capacity. Such tests characterize the system under overload circumstances, but they do not accurately reflect the capabilities of the system. This is especially true when it is possible to further configure or tune the system to limit the input rate so that it cannot exceed the operating capacity.

Another type of test involves running tests at the low-end of the performance spectrum. While these experiments may be sufficient to establish the slope of the normal operation curve, they give no insight into the actual capacity of the system. They can often lead to false conclusions when comparing designs. Measurements at the low end of the performance curve show only the increased resource utilization.

Test Client Limitations

When the apparent capacity of a system is reached without having exhausted any of the available resources, it is necessary to also consider whether the limiting factor might be the test client rather than the system under test.

A test client with a limited number of threads may not be capable of providing inputs or receiving outputs at the rate required to drive the system to its full capacity. Ideally, the test client is configurable through its own parameters. In some cases, it may be necessary to run multiple copies of the test client, each on a different machine, in order to drive the system under test to capacity.

Points to Remember

Keep the following points in mind while performing the benchmarking and tuning exercise.

- Always document the test design in sufficient detail to allow others to accurately reproduce your results.
- Always range demand until the operating capacity of the system under test has been reached. Further increases in input rate do not result in proportional increases in output rate.
- Always document measured or estimated resource availability and consumption.
- Once an apparent operational limit has been reached, investigate to determine whether a true resource constraint has been reached. Consider adding resources such as adding memory, CPU or changing to a higher network bandwidth.
- If an operational limit has been reached without exhausting available resources:
 - Consider whether tuning the system under test might further increase the operational capacity.
 - Consider whether the design or configuration of the test harness might be the true limiting factor in the experiment.

Benchmarking and Testing Performance

This section outlines the steps required to successfully evaluate and tune a TIBCO BusinessWorks Container Edition environment.

Performance Benchmarking Process

This document must be used as a general guideline and is not representative of any comprehensive tuning that may need to be done for each use case. Additional or fewer steps may be required, depending on individual factors and contextual requirements. Such tuning requires multiple iterations.

One of the fundamental requirements before performing any kind of tuning exercise is to carefully eliminate all external factors that can potentially affect any performance issues.

Performance Analysis and Tuning is an iterative process consisting of following:

- Establish performance benchmarking criteria
- Review TIBCO BusinessWorks Container Edition performance architecture
- Establish performance best practices guidelines
- Identify and review tuneable parameters for the use case

Performance Benchmarking Criteria

The first step when measuring performance is to identify the Service Level Agreement. Performance targets are determined by user response time and message processing requirements.

Examples of performance requirements include:

- Engine throughput or number of messages processed per second
- Processing speed or average process instance duration and latency
- Web response time. The response and request time

- Resource utilization
- Concurrent request, sleep time, registered and if applicable, the concurrent users

Defining the minimum, desired, and peak targets for each requirement helps identifying the type of data to collect and to evaluate the test results.

In addition to these normal load expectations, abnormal error-recovery scenarios under unusually high loads must be considered. For example, the TIBCO BusinessWorks Container Edition process might be receiving or polling messages from a TIBCO Enterprise Message Service queue, and the above targets reflect the normal flow of messages through the queue.

However, if communication to the TIBCO Enterprise Message Service server has been disrupted for an extended period of time, or if the TIBCO BusinessWorks Container Edition Engine shuts down, a much higher load may be experienced when communication is re-established or when the engine restarts.

These scenarios must be addressed when considering the engine performance under load, and to ensure that the throughput does not deteriorate below the target in such situations.

Business requirements also control the decision to use reliable or certified messaging. Certified messaging has an impact on performance.

Performance Testing Tools and Techniques

Once you have established appropriate goals for performance benchmarking, it is necessary to define the performance testing and monitoring framework.

This step significantly varies for each project based on application design and deployment requirements. However, it is important to establish key performance monitoring and benchmark measurement techniques and tools. Monitoring each component requires different techniques. It is important to monitor the application including the CPU, memory and logs, the hardware resources, network and performance metrics using load generation tools.

To monitor application resources like CPU, memory, thread dumps and GC, you can use JVisualVM. For more information on JVisualVM and thread dumps, see [Monitoring threads and taking a thread dump using JVisualVM](#).

You can generate Heap dumps from JVisualVM, and can analyze using memory analyzer tools. For errors during load testing, the application logs can be monitored. To monitor the

container level statistics you can use any of the docker stats, VMware Tanzu, Openshift, GCP UI.

Collecting Performance Data

Begin with creating a set of processes for testing purposes. These can be actual processes that are used in production or more basic processes that represent the production scenarios. The granularity and scope of your performance requirements must determine how processes are used for performance testing.

Measure the memory, and CPU usage of the container during all tests. Identify the TIBCO BusinessWorks Container Edition metrics that can measure conformance with requirements. A general strategy is to begin with summary metrics, and then progress to detailed metrics as areas for improvements are identified.

However, if specific performance goals have been defined, you can tailor data collection to provide only required information. Understand where process instance lifetime is spent and collect detailed process instance metrics while processing a few representative process instances. Calculate total throughput, collect summary metrics while generating a typical number of incoming messages of typical size and frequency.

Conduct separate tests for minimum, desired, and peak target numbers. Wherever possible, restrict other container and engine activities during this time. If average metrics are used, restrict the test window to ensure that data collected is relevant.

Deploying Performance Testing Framework

Developing a framework depends on performance goals established earlier and business requirements.

Allocate adequate resources for running BusinessWorks™ container and testing its performance. Deploy BusinessWorks™ application container and start measuring application performance using any monitoring tool like JConsole/JVisualVM. Verify that your BW application running inside a container is providing the performance metrics, such as memory and CPU usage.

Once you have established the key performance matrix and determined the tools and techniques to measure the components, the next step is to establish an appropriate framework for testing performance.

Frequently, a customer would require building a script using third-party performance testing tools such as HP Load Runner, SilkPerformer or JMeter. These tools are frequently used to build extensible framework to invoke HTTP, SOAP, REST and JMS messages.

Developing a Performance Testing Plan

Developing a performance testing plan involves building an appropriate set of tests to meet business objectives.

This section provides series of tests that can be planned based on overall objectives.

Build a Baseline Test

For initial performance tests, consider deploying the TIBCO BusinessWorks™ Container Edition application inside a single container.

Test the performance of the Application Container with varying payload and workload depending on the requirement. After some basic testing, consider adding more Containers. Repeat the tests for testing scalability with added Containers.

For more information, see [Container Tuning Guidelines](#).

Collect all the performance metrics during the benchmark tests like CPU, memory, I/O, network, latency, throughput, and GC during the tests.

Perform tests for minimum, desired, and peak numbers that are identified as required. Capture and store output for the test runs. When the baseline tests are complete, stop the performance data collection utility, stop sending messages, and then stop the Container.

Compare Baseline to Targets

Compare baseline test results to performance requirements. If requirements are met, begin testing the next use case. If the requirements are not met, continue with the remaining steps.

Build Stability Test

Frequently, many performance issues can be identified in the stability test, where the application is deployed under lower load conditions, such as five to ten concurrent users with a pre-established think time.

This test focuses on end-to-end successful transactions, and does not aim at measuring the overall response time. Since the test system involves various components, it is important to familiarize ourselves with the following:

- Tuning parameter at each component level
- Monitor database resources, if applicable
- Monitor any incomplete TIBCO BusinessWorks Container Edition jobs
- Worst performing activities, that is CPU time and Elapsed Time

The test must be completed end-to-end, and the application developer must fix any issues associated with the run. The overall percentage of error and warning must also be noted.

Develop Incremental Tests

Define tests that measure different criteria including error rate in a steady incremental load; for example, 50 users, 100 users, 250 users, and so on.

The user must also define the payload for testing the deployed services. The application must be tested considering the peak load in the production environment.

Develop Peak Rate Tests

The overall business objectives can be different for each project. To measure the system against an extreme number of users, without failing, a peak load test is designed to determine whether the system can respond to the desired maximum number of users and requests without degradation in response time.

Performance depends on the hardware available in the environment. If the CPU and memory resources are not sufficient, then increasing the numbers further degrades the performance.

Develop Steady State Tests

This test can be run when the business objective requires providing well established quality of service for the business users, such as the number of concurrent users with the least amount of response time.

The steady state keeps steady capacity or steady number of requests per minute, even if the number of concurrent users keep increasing.

This test focuses on maintaining partner quality of service and capacity, and the high number of users.

Develop Resource Plan

The choice of a proper operating system with the appropriate number of CPUs is one of the most important decisions during this testing phase.

Develop Component Deployment Plan

The test plan must extend results obtained in the section, "Develop Hardware and Resource Plan", and design for the optimal production deployment.

In this phase of the test, optimal production deployment is planned based on test results obtained in the previous test.

This can be achieved by increasing the number of instances of components on a single container or multiple containers, and using different load balancing and fault tolerance techniques to achieve optimal production objectives.

Monitoring and Analyzing TIBCO BusinessWorks Container Edition Components

There are various ways and tools to monitor and analyze the TIBCO BusinessWorks Container Edition components, BW application inside a container. Some of them are described in this section.

JVisualVM

JVisualVM that is shipped with the Java SDK is a tool that provides a visual interface for viewing detailed information about TIBCO BusinessWorks Container Edition applications while they are running on the AppNode which resides within the container.

JVisualVM organizes JVM data that is retrieved by the Java Development Kit (JDK) tools and presents the information in a way that allows data on multiple TIBCO BusinessWorks Container Edition applications, which can be viewed quickly. We have to monitor the application remotely by enabling the JMX on the JVM.

Users can monitor CPU, memory, classes, threads, monitor the current state of the thread, running, sleeping, wait, and monitor. JVisualVM displays the thread view in real time.

Monitoring Threads and Taking a Thread Dump Using JVisualVM

Use JVisualVM to monitor threads and take thread dumps for a container.

Procedure

1. Enable JMX on the Application Container by adding the following JMX property in the docker run or within the yml files for remote monitoring. *BW_JMX_Config* environment variable is used to set JMX configuration (RMI host and JMX port) for monitoring the TIBCO BusinessWorks Container Edition application. The value must be provided in *RMI_HOST:JMX_PORT* format.

- **Docker-** For docker run, provide the *BW_JMX_CONFIG* value as below:

```
docker run -e BW_JMX_CONFIG="<RMI_Host>:<JMX_Port>" -d
-p <Host_ApplicationPortToExpose>:<Container_ApplicationPort>
-p <Host_JMXPortToExpose>:<Container_JMXPort> <ApplicatonImageName>
```

Where,

- RMI_HOST is the docker host IP
- JMX_PORT is the remote JMX connection, which gets enabled.
- Container_Application port is the end-user requests that get processed.
- Container_JMXPort handles the JMX connection request.

i Note: Published ports (-p) are exposed ports that are accessible publicly over the internet. These ports are published to the public-facing network interface in which the container is running on the node (host).

- **Kubernetes-** In yml file, provide the environment variable entry as *BW_JMX_CONFIG*. Please refer the following *manifest.yml* file:

```
Containers:
- name: <ApplicationName>
  image: <ApplicationImageName>
env:
- name: BW_JMX_CONFIG
  value: "<JMX_PORT>"
```

Once the application container is running, use the `kubectl port-forward <your-app-pod> <JMX_PORT> port`.

i Note: `kubectl port-forward` forwards connections to a local port from a port on a pod where the container is running.

- **VMware Tanzu-** In yml file, provide the environment variable entry as *BW_JMX_CONFIG*. Please refer the following *manifest.yml* file:

```
applications:
- name: <ApplicationName>
  memory: 512M
  path: <ArchiveName>
  buildpack: <BuildpackName>
env:
```

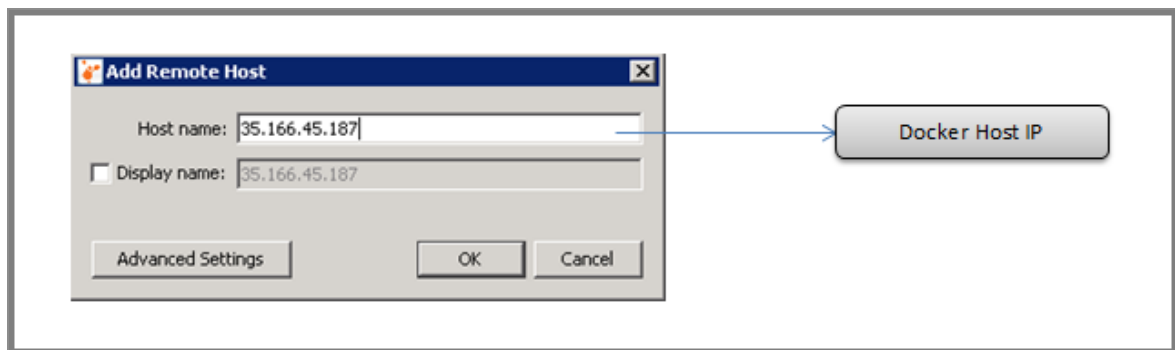
```
BW_JMX_CONFIG: <JMX_PORT>
```

Once the application container is running, add the SSH into the container and map the container port to the localhost or node port using the command below:

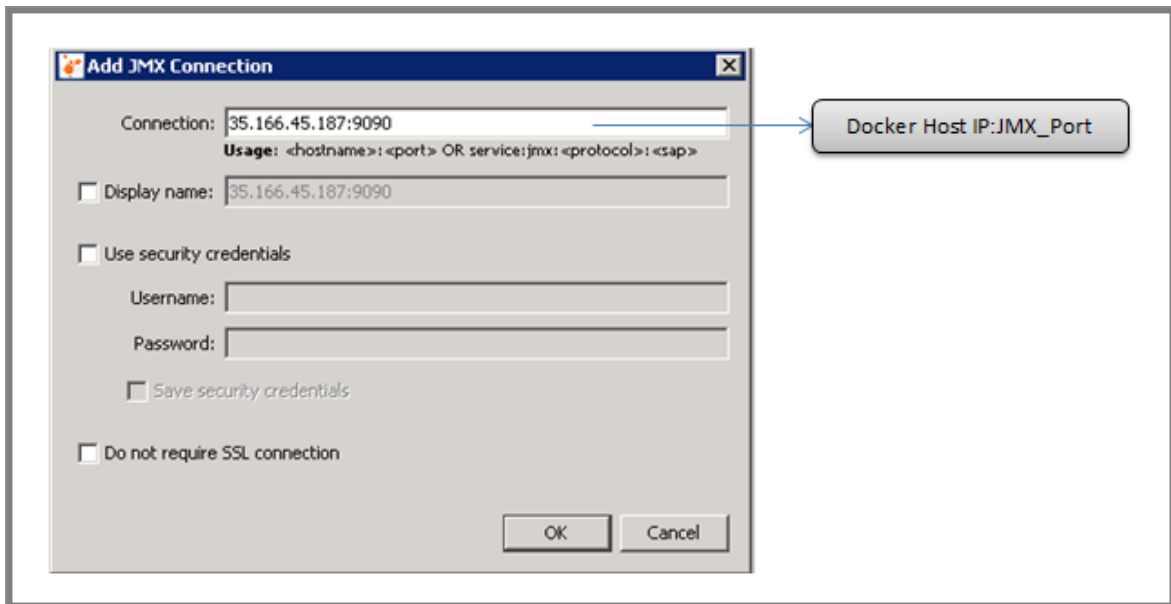
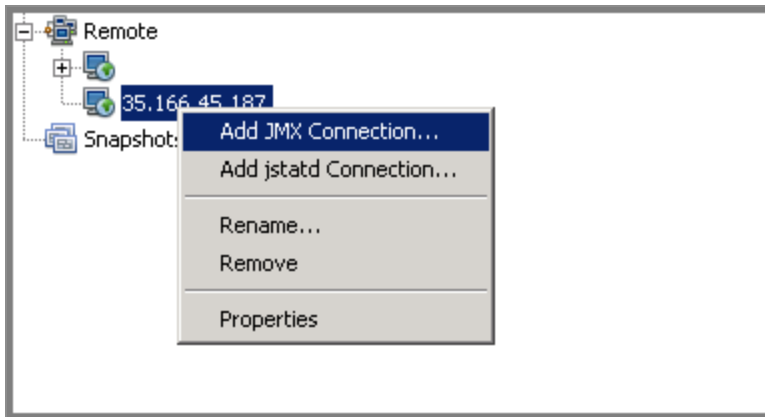
```
cf ssh -N -T -L <Host_Port>:<HostIP>:<Container_JMX_PORT>  
<ApplicationName>
```

Where,

- Host_Port is the port, which is available on the host machine.
 - HostIP is the IP address of the machine from which the JMX connection is being launched.
 - Container_JMX_Port is the JMX_PORT specified in the BW_JMX_CONFIG while the application is launched.
2. Start jvisualvm.exe from the JDK_HOME/version/bin directory.
 3. Connect to the application container remotely or by using the JMX_PORT. To connect remotely, select **Remote** in the **Applications** tab and right-click **Add Remote Host**. Enter the remote **Host name** field.

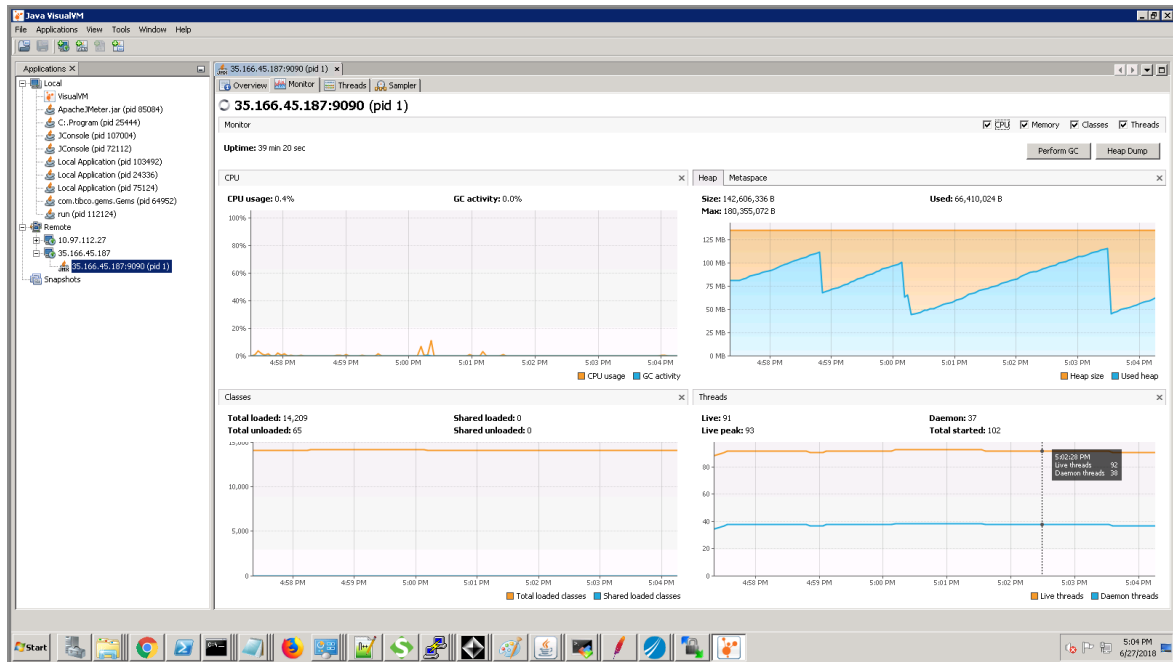


4. Add the JMX connection to the remote host as displayed in the images below.

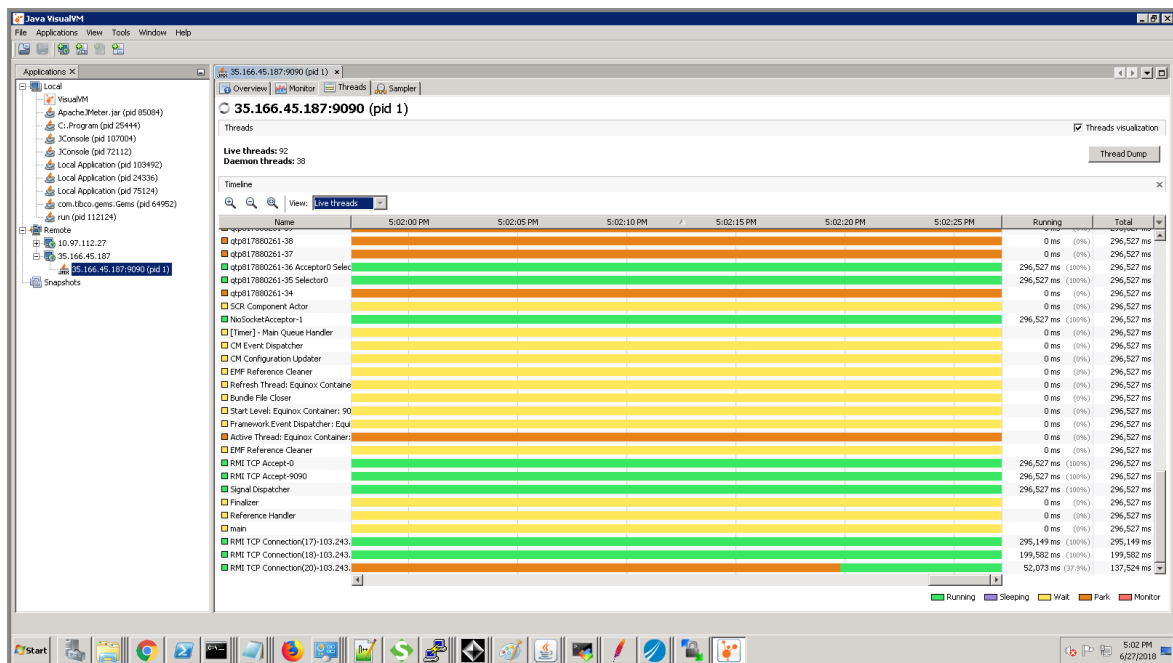


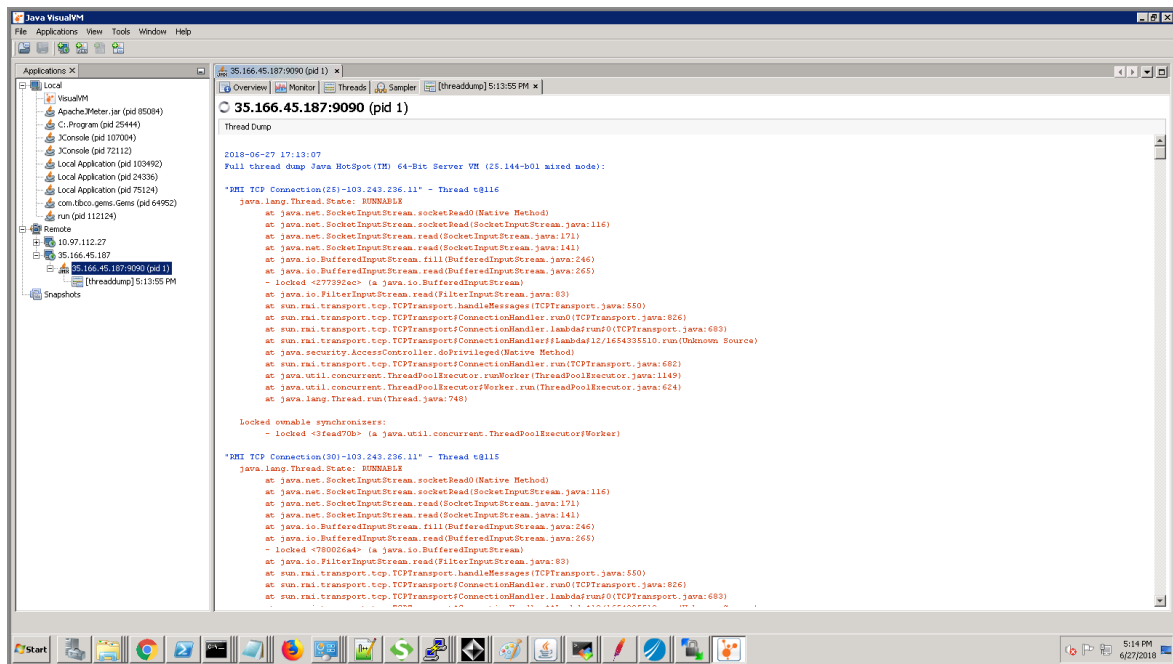
5. Right-click the remote JMX connection for the Container and select **Open**.
6. The AppNode CPU, memory, classes, and threads can be monitored in the **Monitor** tab. The memory chart also provides the maximum memory settings of the JVM. You can perform a manual GC and obtain the heap dump too.

The following figure demonstrates the typical heap usage pattern of the AppNode running inside a container, which is a sawtooth pattern. This sawtooth pattern indicates the normal behavior of the JVM. For more information, see the [Stack Overflow website](#). Here the memory usage steadily increases and then drops due to garbage collection.



- You can monitor the thread states and obtain the thread dump from the **Threads** tab.





8. JVisualVM provides CPU and memory profiling capabilities. By default, the profiling tool is not in a running state until you are ready to profile the application. You can choose from the following profiling options:
 - a. CPU Profiling - Select **CPU Profiling** to profile and analyze the performance of the application in terms of throughput, scalability, or response time.
 - b. Memory Profiling - Select **Memory Profiling** to analyze the memory usage of the application. The results display the objects allocated by the application and the class allocating those objects.
- When you start a profiling session, JVisualVM attaches to the local, or remote AppNode and starts collecting the profiling data.
- When the profiling results are available, they are displayed in the **Profiler** tab.
- JVisualVM has the following plug-ins for the java implementation:
- i. A sampling profiler - Statistical and lightweight
 - ii. An instrumental profiler - Heavier

i Note: CPU and memory usage in the **Monitor** tab are for an application running inside a container. In the actual scenario, we may need to set the container memory greater than the JVM heap size.

Understanding Thread Dumps

Keep in mind the following points while working with thread dumps.

- A thread dump displays the thread name, thread id (tid), which is the address of the thread structure in memory, id of the native thread (nid) which correlates to the OS thread id, thread priority, state (runnable, waiting, blocked, and so on), source code line and method calls.
- Waiting on monitor and waiting for monitor entry - It is very important to understand the difference between the Waiting on monitor state and waiting for monitor entry state. Waiting on monitor means sleep or wait on an object for a specific period or until notified by another thread. Waiting for monitor means to wait to lock an object since some other thread may be holding the lock, which can happen in a synchronized code.
- IBM Thread Dump Analyzer can be used for further analysis. For more information, see <https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=2245aa39-fa5c-4475-b891-14c205f7333c>

Identifying Potential Improvement Areas

If performance requirements are not met and you need to improve the performance, some modifications may be needed.

Modifications may be required in the following areas:

- Adding hardware resources in terms of CPU and Memory
- Modifying JVM parameters
- Increasing engine threads

- Running multiple Containers
- Reducing message size, message burst size or message frequency
- Modifying process design or activity settings
- Tune additional engine parameters and analyze impact
- Tune shared resource level thread pools, connections or any other settings

You can create a prioritized list of modifications using symptoms specific to your environment, such as memory usage patterns or error messages, .

Implementing Specific Enhancements

Scaling involves adding resources or containers. Tuning involves changes to the `manifest.yml` file and changes to the process design. When making any type of change, it is crucial to keep other factors stable so that the effect can be correctly measured.

For example, a sample list of modifications might include:

- Allocate additional memory for the engine JVM
- Increase the number of engine threads
- Enable flow control

These changes can be implemented in an incremental way. The reasonable approach is to implement the first modification, and then test to measure the effect of this change. After a satisfactory value is determined, implement the next modification and measure the effect of the first two changes combined, and so on.

In this example, all the modifications are related to the same resource and memory usage. If the engine requires more memory, then increasing the number of threads in the JVM would be ineffective. However, some scenarios might require measuring the effect of each modification separately, by backing out one change before implementing another.

Comparing Results

After implementing modifications and repeating the test scenarios, compare the adjusted results to the baseline test results.

Exporting performance data to a third-party application to compare test results can simplify this step.

If performance improves because of the modifications, compare the adjusted results to the performance requirements. If the requirements are met, begin testing the next use case. If the requirements are not met, repeat the step, "Develop Incremental Tests" and step "Develop Peak Rate Tests", for additional enhancements.

Setting JVM Parameters

This section describes the JVM parameters for the AppNode.

JVM Parameters

This section specifies some of the JVM parameters and their default values for the AppNode.

Heap Space

JVM parameters that can be configured for the AppNode heap are minimum heap space (Xms) and maximum heap space (Xmx).

The default values for these parameters are `-Xmx1024m -Xms128m`. If you have to change this value, you can update it using the `BW_JAVA_OPTS` environment variable.

For all the platforms the default heap size is 1,024 MB, you can update it using the `BW_JAVA_OPTS` environment variable.

For **Docker**:

```
docker run -e BW_JAVA_OPTS="-Xmx2048M -Xms2048M" -d -p <Host_
ApplicationPortToExpose>:<Container_ApplicationPort>
<ApplicationImageName>
```

For **Kubernetes or VMware Tanzu or Openshift**:

```
applications:
  -name: <ApplicationName>
    memory: 512M
    path: <ArchiveName>
    buildpack: <BuildpackName>
  env:
```

```
-name: BW_JAVA_OPTS  
value: "Xmx2048M -Xms2048M"
```

Heap Dump On Out of Memory Error

The parameter `-XX:+HeapDumpOnOutOfMemoryError` can be set to enable heap dump when the JVM runs out of memory. You have set this parameter in the `BW_JAVA_OPTS` environment variable.

For Example, **`BW_JAVA_OPTS="-Xmx1024M -Xms1024M -XX:+HeapDumpOnOutOfMemoryError"`**

Once JVM throws an OOM exception, heap dump snapshots get generated and it gets stored within the container. Please follow the below steps to copy the heap dump snapshot from the container to your local machine:

- Connect to the container
 - `docker exec -it <containerID> bash`
- Copy the snapshot from a container to the local machine
 - `docker cp <containerId>:/filepath/within/container/host/path/target`

By default the heap dump snapshot gets stored in the root location with the name `java_pid1.hprof`.

Best Practices

This section describes some of the important observations and guidelines developed in field trials.

These guidelines are based on numerous performance improvement projects and details of the new features introduced with TIBCO BusinessWorks Container Edition.

- TIBCO BusinessWorks Container Edition Engine Tuning Guidelines
- TIBCO BusinessWorks Container Edition Transport and Resource Tuning Guidelines
- JVM Tuning Guidelines
- Container Tuning Guidelines



Note: TIBCO recommends to first tune the BWEngine until all the resources are fully utilized and then go for container tuning.

Engine Tuning Guidelines

This section provides high-level steps for TIBCO BusinessWorks Container Edition performance tuning. Specific details are discussed in the subsequent sections.

When tuning the TIBCO BusinessWorks Container Edition engine allocate significant time to select the JVM configurations and the necessary tuning parameters depending on the use cases, payload and workload requirements.

- Since jobs process on a separate Java thread, the number of engine threads controls how many jobs can run simultaneously. The number of threads that an engine can allocate is set in the environment variable section in the `.yaml` file.
- Measuring the available CPU and memory resources on your system under a typical processing load helps you to determine if the default value of eight threads is appropriate for your environment.

For example, if engine throughput has reached a plateau, yet measurements show that CPU and memory are not fully utilized, increasing this value can improve throughput. Typically it is advisable to double the engine threads when tuning the engine if CPU resources are available.

- There are two ways of using CPU resources to the maximum capacity:
 - Optimize engine thread.
 - Tune thread pools, where available, for asynchronous activities.
- Typical numbers of engine threads range between eight and thirty-two. Specifying a value too low can cause lower engine throughput even though spare CPU resources exist. Specifying a value too high can cause CPU thrashing behavior.
- If the pace of the incoming processes still exceeds the number of threads available to run them, consider using flow control.

ThreadCount (bw_engine_threadcount)

The threadCount property specifies the number of threads that the ActiveMatrix BusinessWorks engine allocates. The default value of engine threads is eight.

The jobs in memory are run by the engine. The number of jobs that can be run concurrently by the engine is limited to the maximum number of threads, indicated by this property. This property specifies the size of the job thread pool, and is applied to the BW Application running within the container.

Threads run a finite number of tasks or activities uninterrupted and then yield to the next job that is ready. The thread count can be tuned to the optimum value by starting with a default value of eight threads and then doubling it up until maximum CPU is reached.

The CPU and memory resources must be measured under a typical processing load to determine if the default threadCount value is suitable for the environment. Please refer the below formats to set threadCount value on different container platforms:

Docker:

```
docker run -e BW_ENGINE_THREADCOUNT=16 -p <Host_
ApplicationPortToExpose>:<Container_ApplicationPort>
<ApplicationImage>
```

Kubernetes:

Update the yml file as shown below:

```
env:
  -name: BW_ENGINE_THREADCOUNT
    value: 32
```

VMware Tanzu:

Update the yml file as shown below:

```
env:
  BW_LOGLEVEL: ERROR
  BW_ENGINE_THREADCOUNT: 32
```

StepCount (bw_engine_stepcount)

The engine stepCount property determines the number of activities that are run by an engine thread, without any interruption, before yielding the engine thread to another job that is ready in the job pool.

Exceptions to stepCount can occur when the job in a transaction, is blocked, or is waiting for an asynchronous activity to complete. When a job is in a transaction, the thread is not released until the transaction is complete, even when the stepCount is exceeded.

However, if a job is blocked or waiting for an asynchronous activity to complete, the thread can be yielded even when the stepCount has not been reached.

The default value of this property is -1. When the value is set to -1, the engine can determine the necessary stepCount value. A low stepCount value may degrade engine performance due to frequent thread switches depending on the scenario. Given the nature of the jobs and the number of activities it includes, a high step count value may result in uneven execution of available jobs.

Please refer the below formats to set the step count value on different container platforms:

Docker:

```
docker run -e BW_ENGINE_STEP_COUNT=10 -p <Host_
ApplicationPortToExpose>:<Container_ApplicationPort>
<ApplicationImage>
```

Kubernetes:

```
env:
  -name: BW_ENGINE_STEP_COUNT
    value: 10
```

VMware Tanzu:

```
env:
  BW_LOGLEVEL: ERROR
  BW_ENGINE_STEP_COUNT: 10
```

Flow Limit

This property specifies the TIBCO BusinessWorks Container Edition application's process starters or service bindings flow limit value. There is no default value.

Flow limit is useful when the engine needs to be throttled, as the property specifies the maximum number of jobs that can be started before suspending the process starter. This

ensures that the incoming requests do not overwhelm the engine performance and the CPU and memory is preserved.

If the flow limit is set to a particular value at the application level, each of its process starters contains the same value. For example, if you set the flow limit at application level as eight and the application has two process starters, the flow limit for each of these process starters is eight.

If you want to set the value for one process starter only, set the flow limit at the component level by using the environment variable, `BW_COMPONENT_JOB_FLOWLIMIT`.

For more information on the environment variable, see "Environment Variables for Docker" in *TIBCO BusinessWorks Container Edition Application Development*.

i Note: For component level, use only the `BW_COMPONENT_JOB_FLOWLIMIT` environment variable.

If the number of jobs being created exceeds the flow limit, the engine suspends the creation of new jobs but continues running the jobs in memory. The engine resumes creating jobs when sufficient resources are available. There is no default flow limit value and it is not enforced by the engine unless the flow limit property is specified for an application.

i Note: Only set the flowlimit property for applications using non-HTTP based transports, for example, JMS. If applications are using HTTP-based transports, ensure you set the Max QTP thread value of the HTTP Connector share resource to apply the Flow limit.

i Note: Get the logs for the component states such as Start, Stop, and Resume based on whether the **FlowLimit** is breached or complied by enabling the core runtime logger `com.tibco.bw.core` at INFO level.

Docker:

```
docker run -e BW_APPLICATION_JOB_FLOWLIMIT=32
-p <Host_ApplicationPortToExpose>:<Container_ApplicationPort>
<ApplicationImage>
```

Kubernetes:

```
env:  
  -name: BW_APPLICATION_JOB_FLOWLIMIT  
  value: 32
```

VMware Tanzu:

```
env:  
  BW_LOGLEVEL: ERROR  
  BW_APPLICATION_JOB_FLOWLIMIT: 32
```

Application Statistics

The TIBCO BusinessWorks Container Edition engine collects three types of statistics for an application, application job metrics, process statistics, and execution statistics.

Enabling statistics adds a certain performance overhead.

Application statistics collection can be enabled or disabled from the monitoring dashboard. For more information, see the [Application Monitoring and Troubleshooting](#) section in the product documentation.

We can view process data, activity data, process instance data, and application properties in the Application Statistics.

Process Statistics

Process statistics collection for an application or an individual processes can be enabled or disabled from monitoring dashboard.

For more information, see [Application Monitoring and Troubleshooting](#) guide.

Process Execution Statistics

Using process execution statistics, you can collect information for individual processes and activities of an application such as status, start time, elapsed time and so on.

You can use this information to identify time consuming activities and processes in an application.

Statistic	Description
Status	Status of the activity. For example, completed, faulted or cancelled.
Execution Time	The execution time for an activity is the actual time (in milliseconds) required by the activity to complete.
Elapsed Time	Elapsed time of an activity is the time difference (in milliseconds) between start time and end time of the activity. Between the start and end time, control may get switched with other activities from other jobs. This is the time taken to run an activity plus all the delays in acquiring resources like engine threads, JDBC connections, network, and so on. The elapsed time is eval time plus the time taken for evaluating all the forward transitions from that particular activity.

Activity Instance Statistics

Using activity statistics, you can collect the information for each activity in the process diagram. The following activity statistics is collected:

Process Instance Statistics

The following process instance statistics are collected:

Statistic	Description
State	State can be completed and faulted
DurationTime	The time for a process is the total time taken by the process, including the elapsed time for all the activities run for the process.
EvalTime	The Eval Time for an activity is the actual time (in milliseconds) used by the activity itself to complete while using the engine thread. Asynchronous activities may use other threads not included in this time.

JVM Tuning Guidelines

Each TIBCO BusinessWorks Container Edition engine runs as a multi-threaded Java server application. Processes and other objects used internally by TIBCO BusinessWorks Container Edition are Java objects that consume memory while the engine is running.

Java provides some useful parameters for tuning memory usage. Ensure that you consider these factors when selecting a JVM.

Besides the JVM version and the vendor, the most relevant tuning parameters are:

- JVM heap size
- Garbage collection settings

Specifying JVM Heap Size

The default Java heap size, which varies according to platform, is a conservative estimate made by the developers of the particular type of Java being used.

When you calculate the amount of memory required for an AppNode, you must determine the largest heap size that can reside in a physical memory. For best engine performance, paging to disk must be avoided.

The recommended heap size for a small workload is 1024 MB, for medium 2048MB, and for large workload 4GB or more. The recommendations change based on the scenarios, payload and workload.

To set the JVM available memory, use the following parameters:

- The Initial JVM Size parameter sets the minimum amount of memory used.
- The maximum JVM Size sets the maximum amount of memory used.

For more information, see [Setting Appnode JVM Parameters](#) .

i Note: TIBCO recommends to set minimum heap size to default value and tune the maximum heap based on monitoring of memory usage and garbage collection frequency under load. It is also recommended to run baseline, scalability, and stability tests for identifying the optimum value of maximum heap size.

The total amount of JVM memory required to operate a TIBCO BusinessWorks Container Edition engine must be the memory for each process plus the maximum number of processes that can be in the process pool. If flow control is enabled, the process pool can contain up to the Max Jobs value.

i Note: While uploading ears, deploying applications, and browsing, increase the heap size in `bwagent` and `AppSpace.tra` files. The property specified in the `AppSpace.tra` file applies to all `AppNodes` associated with the `AppSpace`. However, the property specified in the `AppNode.tra` file only applies to a specific `AppNode` and overwrites the property specified in the `AppSpace.tra` file. The property specified in `bwagent.tra` does not overwrite properties in the `AppNode.tra` or the `AppSpace.tra` files.

JVM Garbage Collection

Tuning garbage collection requires good understanding of garbage collection frequency, message size, and longevity, young, tenured, and perm.

Many JVMs have different levels of algorithms to suit application requirements. Hence, it is very difficult to provide general recommendations.

The option `AggressiveHeap` affects the memory mapping between the JVM and operating system, and the option `ParallelGC` allocates multiple threads to part of the garbage collection system.

i Note: The client must try to experiment with these options, but the results are not deterministic. It can be argued that a multi-process system of four to twelve CPUs, using `ParallelGC` can produce better results.

We recommend that the application must not make direct explicit garbage collection. If the application does make a call to explicit garbage collection, to disable it, use `verbose:gc -XX:+DisableExplicitGC` command. These properties can be appended to the `BW_JAVA_OPTS` variable.

Transport and Resource Tuning Guidelines

Performance must be one of the criteria for using the appropriate transport. Besides performance, you must also consider interoperability, reliability, and security requirements.

You have choices of using HTTP or JMS, SOAP over HTTP, SOAP over JMS and TIBCO Active Enterprise™ messages. The best practice of selecting a proper standard for the project is beyond the scope of this document. However, the following general results from a performance point of view are provided for your reference only.

You must use them only as general observations, and for initial understanding. For different environment and different requirements, these results cannot be directly applied to their work. Instead, it is strongly recommended that you create a realistic scenario in the lab and then derive the appropriate results.

HTTP Resource

Performance characteristics of SOAP and REST are closely tied to the performance of the HTTP implementation in TIBCO BusinessWorks Container Edition.

In some situations, you can alter the configuration of the HTTP server that receives incoming HTTP requests for TIBCO BusinessWorks Container Edition. There are two thread pools that can be tuned or set to appropriate values for handling the incoming concurrent requests efficiently.

- **Acceptor Threads:** These are the Jetty server threads. Acceptor threads are HTTP socket threads for an **HTTP Connector** resource that accept the incoming HTTP requests. While tuning these threads, the rule of thumb, as per the Jetty documentation, is **acceptors >= 1<=#CPUs**.
- **Queued Thread Pool:** The Queued Thread Pool (QTP) uses the default job queue configuration. The QTP threads accept the requests from the Acceptor Threads.

You can configure the following two properties to specify values for the Queued thread pool.

- **Minimum QTP Threads:** The minimum number of QTP threads available for

incoming HTTP requests. The HTTP server creates the number of threads specified by this parameter when it starts up. The default value is 10.

- **Maximum QTP Threads:** The maximum number of threads available for incoming HTTP requests. The HTTP server cannot create more than the number of threads specified by this parameter. The default value is 75. This limit is useful for determining the number of incoming requests that can be processed at a time. Setting a high number creates that many threads and drastically reduce the performance.

TIBCO recommends that if you have a large number of incoming requests, you can change these values to handle more incoming requests concurrently. You can also increase these numbers only if you have a higher peak concurrent request requirement, and you have large enough hardware resources to meet the requests.

In addition to the QTP threads, REST services also include the `bw-flowlimit-executor-provider` thread pool, which is a part of the Jersey framework. The QTP threads offload the work to `bw-flowlimit-executor-provider` threads. These threads handle the processing of the RESTful services. The core pool size of these threads is equivalent to the min QTP thread pool that is the settings are picked from the values provided for the QTP pool on the HTTP Connector resource. The pool size values cannot be modified directly, which means that if users want to modify the thread settings, they need to change the QTP thread pool values.

All the incoming requests are processed by one of the available Jersey threads. If in case the Jersey thread is not available, the request is queued (based on blocking queue size), and this holds true until the Queue gets filled up. If the load persists continuously and the Queue is already full, then the Jersey threads are scaled up to `maxQTP`(in other words, Jersey `maxPoolSize`), that is the setting are picked from the values provided for the QTP pool on the HTTP Connector resource.

The `bw-flowlimit-executor-provider` threads are created for every service deployed in the application EAR. If there are 10 services deployed in a single EAR, on a single AppNode container, then the `bw-flowlimit-executor-provider` threads created would be equal to the number of services deployed that is $(10) * \text{CorePoolSize}$.

For more information about configuration details of the parameters, see [Tuning Parameters](#)

HTTP Client Resource

There are two important tuning considerations related to the HTTP client:

- HTTP Client Thread Pool
- Connection Pooling

HTTP Client Thread Pool - If you are using HTTP or SOAP with HTTP Send Request or SOAP Invoke (Reference) activity, it is important to verify that the rate of request received on the HTTP server keeps up with the rate at which the client sends messages. This situation arises when there are very high numbers of concurrent requests being made.

Each Request and Response activity that uses the HTTP protocol, for example, **Send HTTP Request** or **SOAP Invoke** is associated with a unique thread pool.

Each request is run in a separate thread, which belongs to the thread pool associated with the activity. The number of threads in the pool determines the maximum number of concurrent requests a request or response activity can run. This is a cached thread pool with no default value.

Alternatively, you can create a separate client thread pool and define the core and max pool values. Set the value of this property to a reasonable number for your system. If you set the value too high, it may result in extra resources being allocated that are never used.

You may want to increase the value of the max pool size. However, this value must be increased only if the client side has more backlogs compared to the receive side. TIBCO recommends that you design a test framework that helps you monitor such behavior and determine optimal thread pool count.

Connection Pooling - In the absence of connection pooling, the client makes a call to the same server in a single threaded communication. By enabling connection pooling, multithreaded communication is enabled. Hence, by default the **HTTP Client** provides single threaded connection. Single threaded connection can be used for multiple sequential requests. However in cases where there are multiple concurrent requests, enabling connection pooling is required.

If connection pooling is enabled, it can improve performance as a pool of reusable shared connections are maintained. Using the connection pooling feature you can keep the default values for the number of connections that the client establishes with the service, or tune them appropriately. The values must not be set to zero (0).

For more information, see [Tuning Parameters](#).

JMS Resource and JMS Transport

Explained below are some usage recommendations when using JMS resource or JMS transport.

Usage Recommendations

Transport such as JMS can be throttled by limiting the number of sessions. JMS does not deliver more messages until some of the sessions have been acknowledged.

The combination of TIBCO Enterprise Message Service features Explicit Acknowledge and FlowLimit also exists. In this case, location of ack in process makes no difference.

When using Client Ack, the JMS session cannot receive a new message until the current message is acknowledged.

Using TIBCO BusinessWorks Container Edition you can configure multiple sessions to receive messages faster, and set number of sessions higher than the number of engine threads.

Acknowledge and confirm messages as soon as possible, to improve throughput.

By holding Client ack to the end of the process, you block that session. This means you slow down the rate at which TIBCO BusinessWorks Container Edition pulls messages from the JMS server, which holds messages for a longer period of time.

With TIBCO Enterprise Message Service Explicit Ack, a single session is used to receive all messages. This mode allows for more efficient resource utilization, and provides even load distribution across multiple engines.

The best way to increase performance beyond the capability of a single engine is to distribute the load over multiple engines using a load-balanced transport such as JMS Queue, to distribute the work. External mechanisms exist to allow HTTP to be used for this purpose also.

Simple and Text JMS message types have the lowest processing overhead.

To design a long running process to fetch a message and process it, use **Get JMS Queue** message activity in a loop instead of **Wait For JMS Queue** message. In most cases, a JMS starter is sufficient in this scenario.

Usage Recommendations

If possible, choose `NON_PERSISTENT` as the delivery mode in replying to a JMS message.

For multiple JMS Receiver activities on an AppNode, the polling interval value has an impact on the CPU utilization of the AppNode. Lower the polling interval, higher is the CPU utilization, and higher the polling interval, lower the CPU utilization. In an environment where there are constraints on the available CPU resources, TIBCO recommends that the polling interval value must be increased to lower the CPU consumption.

Impact of SSL on Performance

When using HTTP over SSL, the overhead added by SSL on the overall request and response or throughput for a scenario is limited.

For scenario and settings details, see [HTTP Connector Resource](#) .

Container Tuning Guidelines

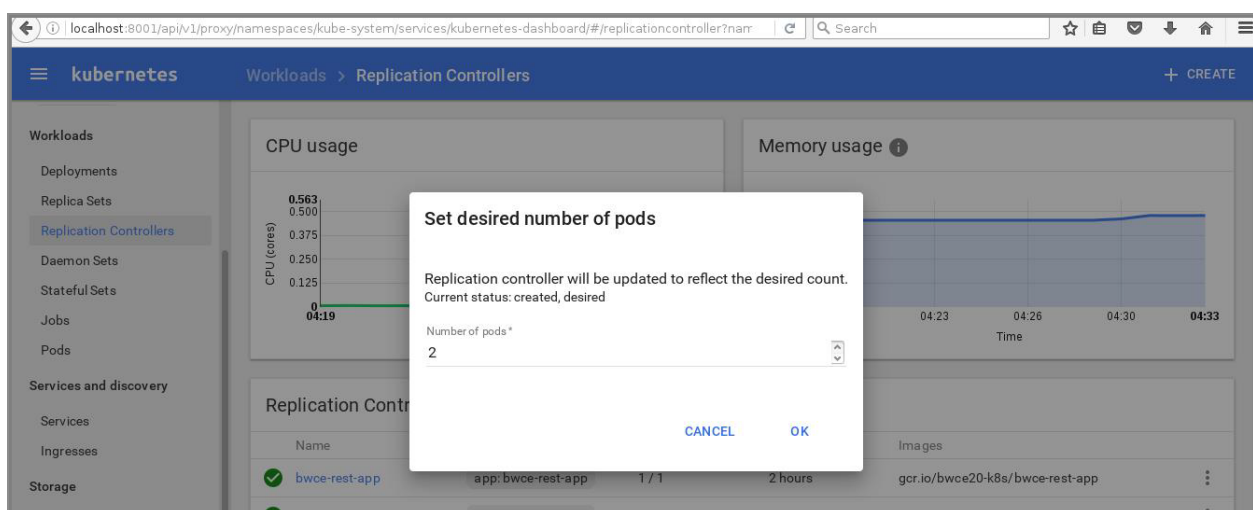
Factors such as user load, or the number of tasks performed by an application, can impact the performance metrics like disk space and memory of the container that the application is using. For many applications, increasing the available memory can improve overall performance. Similarly, running additional instances of an application can allow the application to handle increase in user load and concurrent requests. These adjustments are called as scaling an application.

Horizontal Scaling

Horizontally scaling an application creates or destroys instances of the application. Incoming requests to the application are automatically loaded balanced across all the instances allocated to the application, and each instance handles tasks in parallel with every other instance. Adding more instances allows the application to handle increased traffic and demand.

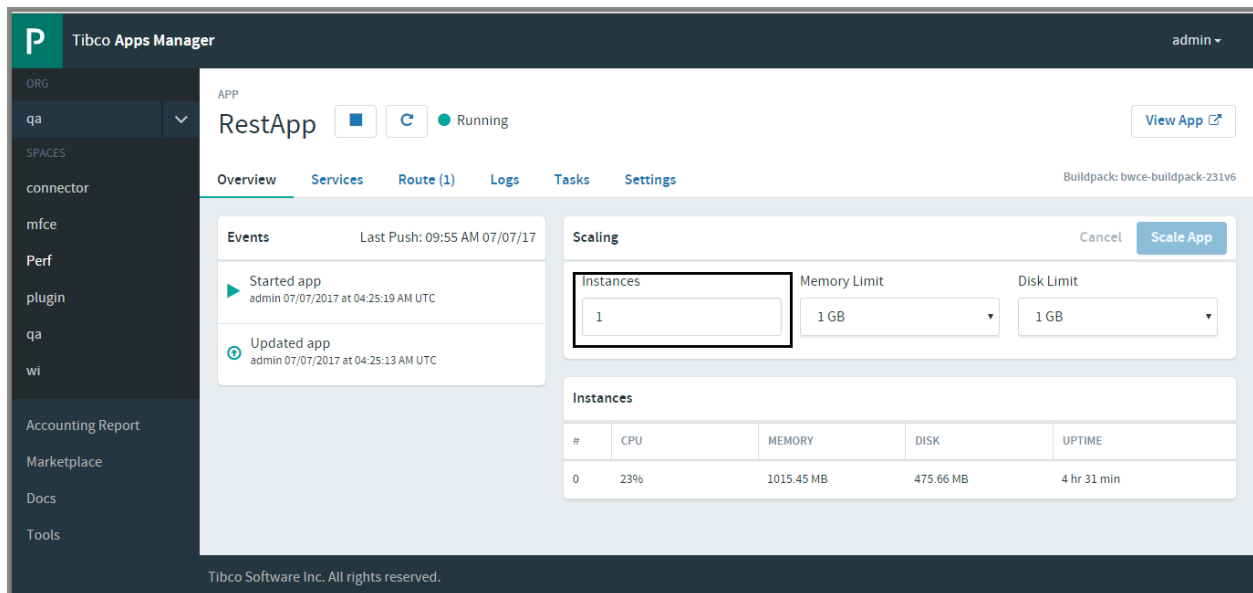
Kubernetes:

For this platform, you can do the horizontal scaling by increasing the PODS that is the containers allocated for that application. Below snapshot shows scaling through the Kubernetes dashboard.



VMware Tanzu:

Horizontal scaling can be achieved through the `cf` cli as well as with the cloud foundry UI. Use **`cf scale APP -i INSTANCES`** to horizontally scale your application. Cloud Foundry increases or decreases the number of instances of your application to match INSTANCES. Below snapshot shows the horizontal scaling through the cloud foundry UI.



Vertical Scaling

Vertically scaling an application changes the memory limit that container platform such as Docker, VMware Tanzu, Kubernetes applies to all instances of the application. We can deploy multiple applications with one `cf push` command by describing them in a single manifest. For doing this, we need to look into the directory structure and path lines in the manifest. We can also define the number of instances and memory allocation parameters in the manifest file, which is used while deploying the application.

Docker:

For Docker, the vertical scaling is achieved by increasing the memory allocated for the container as well as increasing the number of CPUs defined for the container.

Use **`docker run -m MEMORY`** to change the memory limit applied to the container of your application. Memory must be an integer followed by either an M, for megabytes, or G, for gigabytes.

Use **`docker run -cpus 2.00`** to change the number of CPUs allocated for that container. Number is a fractional number. 0.000 means no limit. If you do not specify this parameter, it uses all the CPUs available on that host or instance where the container is running.

Below snapshot shows how to set the CPU and memory for the container:

```
docker run -m 1024M -cpus 2.0 -p
<Host_ApplicationPortToExpose>:<Container_ApplicationPort>
<ApplicationImageName>
```

Kubernetes:

For this platform vertical scaling can be done by increasing the resource limits that are CPU and Memory allocated for the container. One can achieve this by specifying the limits in the manifest.yml file. Below is the snapshot of the manifest.yml file.

```
containers:
  resources:
    limits:
      cpu: 0.5
      memory: 512Mi
    requests:
      cpu: 0.5
      memory: 512Mi
```

Kubernetes schedules a Pod to run on a Node only if the Node has enough CPU and RAM available to satisfy the total CPU and RAM requested by all of the containers in the Pod. Also, as a container runs on a Node, Kubernetes does not allow the CPU and RAM consumed by the container to exceed the limits you specify for the container. If a Container exceeds its RAM limit, it shut downs from an out-of-memory condition. If a container exceeds its CPU limit, it becomes a candidate for having its CPU use throttled.

You can improve reliability by specifying a value that is a little higher than what you expect to use. If you specify a request, a Pod is guaranteed to be able to use that much of the resource.

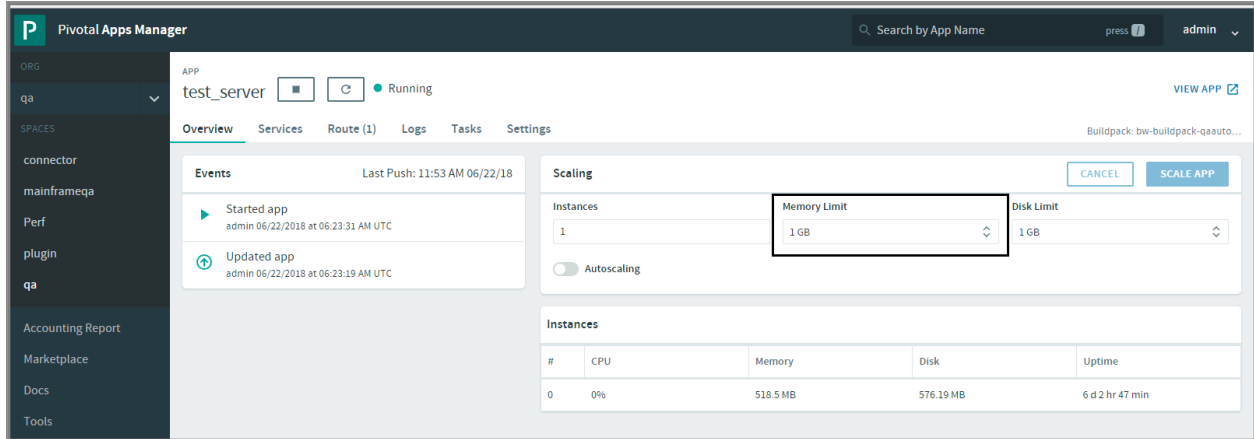
If you do not specify a RAM limit, Kubernetes places no upper bound on the amount of RAM a Container can use. A Container could use the entire RAM available on the Node where the Container is running. Similarly, if you do not specify a CPU limit, Kubernetes places no upper bound on CPU resources, and a Container could use all of the CPU resources available on the Node. In this, you do not need to do the vertical auto scaling.

VMware Tanzu:

For the cloud foundry, you can do the vertical scaling by increasing the container resources that are memory, and disk. You can do this through the cloud foundry command line and the cloud foundry UI.

Use **`cf scale APP -m MEMORY`** to change the memory limit applied to all instances of your application. MEMORY must be an integer followed by either an M, for megabytes, or G, for gigabytes.

The below snapshot shows how vertical scaling can be done through VMware Tanzu UI.



The screenshot shows the Pivotal Apps Manager interface for the 'test_server' application. The 'Scaling' section is highlighted, showing the following configuration:

- Instances:** 1
- Memory Limit:** 1 GB
- Disk Limit:** 1 GB
- Autoscaling:** Off

Below the scaling section, there is a table showing the current state of the instances:

#	CPU	Memory	Disk	Uptime
0	0%	518.5 MB	576.19 MB	6 d 2 hr 47 min

Note: Vertical scaling beyond a certain limit is not good practice in the containerized deployment. Typically you must consider horizontal scaling if the container resource limit requirement is greater than 2cores or 4vCPUs.

Tuning Parameters

This section explains the tuning parameters for the following connectors, connection resources and messaging servers:

- HTTP Connector Resource
- HTTP Client Resource Tuning Parameters
- JDBC Connection Resource
- TCP Connection Resource
- JMS Receiver
- Blocking Queue Size

HTTP Connector Resource

This section describes the tuning parameters for the **HTTP Connector** resource that you can configure in TIBCO Business Studio for BusinessWorks.

Basic Configuration

Field	Description
Acceptor Threads	These are the HTTP socket acceptor threads for the HTTP Connector resource, which pick up the HTTP requests. The default value is 1.
Accept Queue Size	This is the number of connection requests to be queued before the operating system starts sending rejections. The value can be set to either 0 or -1 .These values signify that the queue size is 50 or OS-specific.

HTTP Connector - Basic Configuration

HTTP Connector

General

Package: Name:

Description:

▼ **Basic Configuration**

Host:

Port:

Accept Queue Size:

Acceptor Threads:

Advanced Configuration

Field	Description
Queued Thread Pool	The Queued Thread Pool is the thread pool provided by Jetty that uses the default job queue configuration. The QTP threads accept requests from the Acceptor threads. The default values are: • Minimum QTP threads = 10 • Maximum QTP threads = 75

HTTP Connector - Advanced Configuration

▼ **Advanced Configuration**

Header Buffer Size (B)	4096		Use Non-Blocking IO Sockets	☐
Request Buffer Size (B)	8192		Use Direct Buffers	☒
Response Buffer Size (B)	24576		URI Encoding	
Max Idle Time (ms)	200000		Enable DNS Lookups	☐
Low Resource Max Idle Time (ms)	0		Compression	☐
Linger Time (ms)	0		Compressible Mime Types	text/html,text/xml,text/plain
Max Post Size	2097152		Reverse Proxy Host	
Max Save Post Size	4096		Reverse Proxy Port	0
Minimum QTP Threads	10		Maximum QTP Threads	75

HTTP Client Resource Tuning Parameters

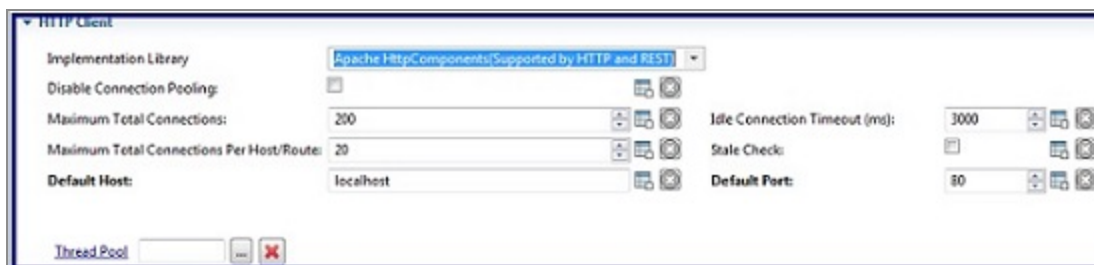
This section describes the tuning parameters for the **HTTP Client** resource that you can configure in TIBCO Business Studio for BusinessWorks.

HTTP Client

You can configure the following fields.

Field	Description
Maximum Total Connections	This is the maximum number of concurrent HTTP connections allowed by the resource instance to be opened with the target service. This property is enabled only if connection pooling is enabled, that is the disable connection pooling parameter is not selected. For applications that create many long lived connections, increase the value of this parameter. Default value = 200
Maximum Total Connections Per Host or Route	This is the maximum number of concurrent HTTP connections allowed by the resource instance to be opened with the target service to the same host or route. This property is enabled only if connection pooling is enabled, that is the disable connection parameter is not selected. This value cannot be more than the maximum total connections. The value for these threads can be modified at design time in TIBCO Business Studio for BusinessWorks as shown in the snapshot. Every connection created here also counts into Maximum Total Connections. Default value = 20

HTTP Client Resource



Thread Pool Resource

You can optionally create the client thread pool that routes the messages to the target service. The thread pool resource can be created by either selecting a thread pool resource template or creating a new one. The values for these threads can be modified at design time in TIBCO Business Studio for BusinessWorks.

The default values are:

- Core Pool Size = 5
- Max pool Size = 10

HTTP Client Thread Pool

The screenshot shows a 'Thread Pool' configuration window. It contains the following fields and values:

- Core Pool Size: 5
- Max Pool Size: 10
- Keep Alive Time (s): 30
- Autostart Core Threads: ☐
- Thread Pool Name Prefix: (empty text box)
- Daemon: ☐
- Priority: 5
- Rejection Policy: BLOCKING (dropdown menu)

JDBC Connection Resource

This section describes the tuning parameters for the **JDBC Connection** resource that you can configure in TIBCO Business Studio for BusinessWorks.

JDBC Connection

You can configure the following field.

Field	Description
Max Connections	This parameter specifies the maximum number of database connections to allocate. Default value = 10

JDBC Connection Resource

JDBC Connection	
Connection Type:	Direct
Maximum Connections:	10

TCP Connection Resource

This section describes the tuning parameters for the **TCP Connection** resource that you can configure in TIBCO Business Studio for BusinessWorks.

TCP Connection

You can configure the following fields.

Field	Description
Maximum Connections	This is the maximum number of simultaneous client sessions that can connect with the server. This parameter is enabled only if connection pooling is enabled. Default value = 10
Maximum Wait Time	This is the maximum wait time in milliseconds to connect to the TCP server. This parameter is enabled only if connection pooling is enabled. Default value = 10000 ms

TCP Connection Resource

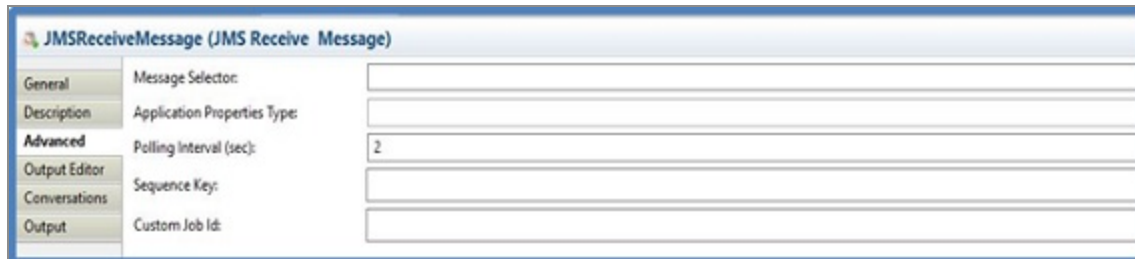
▼ TCPConnection	
Host:	
Port:	0
Enable Connection Pool:	☒
When Exhausted Connection:	Block
Maximum Connections:	10
Maximum Wait Time (msec):	10000
Idle Timeout (msec):	-1

JMS Receiver

You can configure the polling interval variable in the **Advanced** tab of the **JMS Receive Message** activity.

Field	Description
Polling Interval (sec)	This property specifies the polling interval in seconds to check for new messages. If a value is not specified for the property, Setting a value in this field overrides the default polling interval. Default value = 2

JMS ReceiveMessage

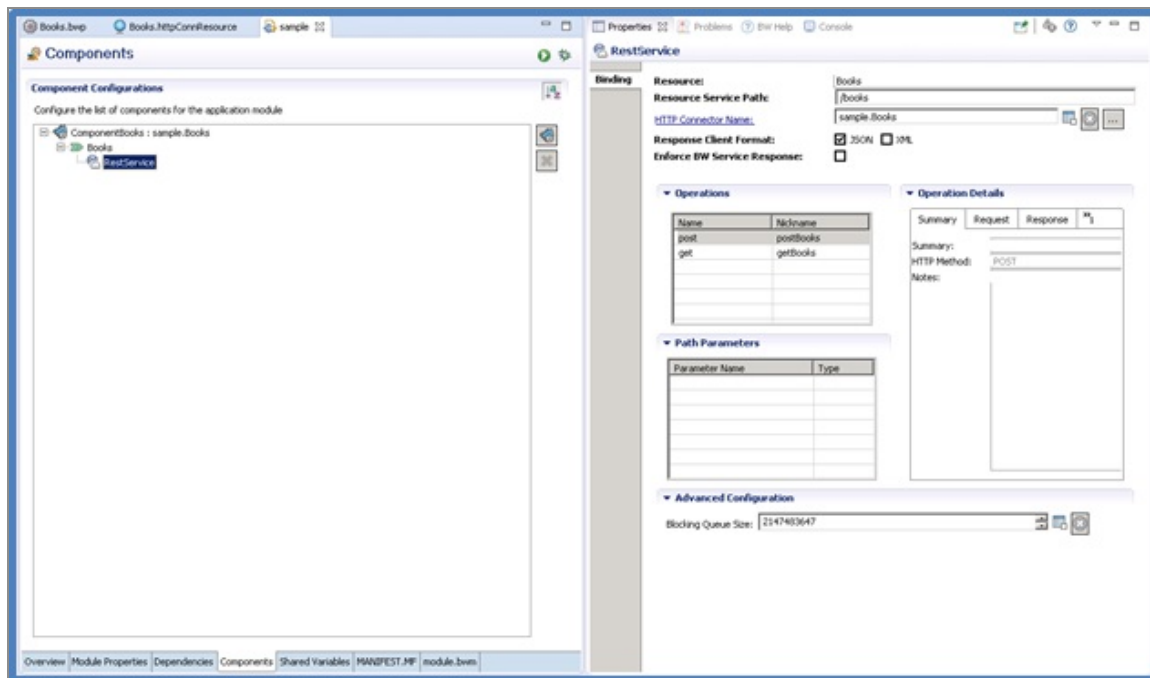


The screenshot shows the configuration window for the JMSReceiveMessage activity. The 'Advanced' tab is active, displaying the 'Polling Interval (sec)' field with a value of 2. Other tabs visible include General, Description, Output Editor, Conversations, and Output. The 'General' tab shows 'Message Selector' and 'Application Properties Type' fields. The 'Description' tab shows 'Polling Interval (sec)' and 'Sequence Key' fields. The 'Output Editor' tab shows 'Custom Job Id' field.

Blocking Queue Size

This is the number of requests to be queued before the operating system starts sending rejections. In case of unavailability of Jersey threads, all the incoming requests get queued up until the queue gets full. The default blocking Queue size is Integer.MAX_VALUE which is unbounded. The Blocking Queue size can be modified at design time in TIBCO Business Studio for BusinessWorks as shown in the following image.

Parameter location: **Bindings > RestService > Advanced Configurations > Blocking Queue Size**



Debugging Performance Issues

This section describes how to debug for performance related issues.

- [Debugging High CPU Utilization Issues](#)
- [Debugging High Memory Utilization Issues](#)
- [Debugging High Latency Issues](#)

Debugging High CPU Utilization Issues

The CPU utilization of container application for a particular service depends on the complexity of service implementation, payload, workload, number of services deployed on the container application, and the CPU or Memory resources made available in the container.

Before you begin

Assuming all the components of the engine are tuned for debugging high CPU utilization issues on the container, collect the following data that helps in further debugging and understanding the issues.

Procedure

1. Capture thread dumps for analyzing the thread state and calls. Capture five thread dumps at an interval of 5 seconds each. Redirect all the collected threads dumps to separate files.
 - The thread dumps can be captured using the *jstack* utility shipped with JDK.

```
./jstack <PID of container application> > ThreadDump_n.txt
```
 - The thread dumps can be captured through *JConsole* or *JVisualVM* utilities.
2. Capture top CPU consuming threads data for 5 minutes run by using a *jvmtop* utility. For more information about the *jvmtop* utility, see support article [KB000034702](#).

```
./jvmtop <PID of container application> > JVM_topthreads.txt
```

3. Capture method level CPU profiling data for 5 minutes run by using a *jvmtop* utility.

```
./jvmtop --profile <PID of container application> > JVM_CPUProfile.txt
```

4. Capture container logs for the test run duration.
5. Capture the CPU and memory utilization of the container application for 5-minutes run.
 - The data can be captured through a *top* utility on Unix.

```
top -p <PID of container application> > top_container.txt
```

- The data can also be captured through *JConsole* or *JVisualVM* utilities.
6. Check the *BW_JAVA_OPTS* and other runtime parameters with which the container is running. This helps analyze the JVM parameters and other engine tuning parameters.
 7. Capture system configurations of containers such as CPU details, RAM, and number of cores where, TIBCO BusinessWorks™ Container Edition applications, external services, and load generator are running. Capture details of */proc/meminfo* and */proc/cpuinfo* files.

```
cat /proc/meminfo and cat /proc/cpuinfo
```

Debugging High Memory Utilization Issues

The memory utilization for a particular service on the container application depends on the complexity of service implementation, payload, workload, the number of services deployed on the container application, CPU, or memory resources made available to the container .

Before you begin

Assuming all the components of the engine are tuned for debugging high memory utilization issues on the container application, collect the following data that helps in

further debugging and understanding the issues.

Procedure

1. Capture the heap dump on the container application when memory issues are observed on the container application.

- The heap dumps can be captured by using the *jmap* utility.

```
jmap -dump:live,file=memorydump.hprof <PID of container application>
```

- The heap dumps can also be captured through *JConsole* or *JVisualVM* utilities.

The heap dump can be analyzed using the memory analyzer tool for checking memory leaks and top components of memory.

2. Capture *jstat* data for checking the allocation and utilization of different memory pools for 5-minutes run.

```
jstat -gc <PID of container application> > jstat_gc.txt
```

3. Capture thread dumps for analyzing the thread state and calls. Capture five thread dumps at an interval of 5 seconds each. Redirect all the collected threads dumps to separate files.

- The thread dumps can be captured by using the *jstack* utility shipped with JDK.

```
./jstack <PID of container application> > ThreadDump_n.txt
```

- The thread dumps can also be captured through *JConsole* or *JVisualVM*.

4. Capture container logs for the test run duration.

5. Capture the CPU and memory utilization data of the container application for 5-minutes run.

- The data can be captured through the *top* utility on Unix.

```
top -p <PID of container application> > top_application.txt
```

- The data can also be captured through *JConsole* or *JVisualVM* utilities.

6. Check the BW_JAVA_OPTS and other run-time parameters with which the container is running. This helps analyze the JVM parameters and other engine tuning

parameters.

7. Capture system configurations of servers such as CPU details, RAM, and number of cores where TIBCO BusinessWorks Container Edition applications, external services, and load generator are running. Capture details of `/proc/meminfo` and `/proc/cpuinfo` files.

```
cat /proc/meminfo and cat /proc/cpuinfo
```

Debugging High Latency Issues

The latency of a particular service depends on the complexity of service implementation, payload, workload, the number of services deployed on a container, and CPU or memory resources made available to the container.

Before you begin

Assuming all the components of the engine are tuned for debugging high latency issues on the container, collect the following data that helps in further debugging and understanding the issues.

Procedure

1. Capture the process execution statistics for the test run. This would help analyze the time spent in individual processes and activities. For more information, see the *TIBCO BusinessWorks Container Edition Application Monitoring and Troubleshooting* guide.
2. Capture thread dumps for analyzing the thread state and calls. Capture five thread dumps at an interval of 5 seconds each. Redirect all the collected threads dumps to separate files.

- The thread dumps can be captured using the *jstack* utility shipped with JDK.

```
./jstack <PID of container application> > ThreadDump_n.txt
```

- The thread dumps can be captured through the *JConsole* or *JVisualVM* utilities.
3. Capture top CPU consuming threads data for 5 minutes run by using the *jvmtop* utility.

For more information about the *jvmtop* utility, see support article [KB000034702](#).

```
./jvmtop <PID of container application> > JVM_topthreads.txt
```

4. Capture method level CPU profiling data for 5 minutes run by using the *jvmtop* utility.

```
./jvmtop --profile <PID of container application> > JVM_CPUProfile.txt
```

5. Capture container application logs for the test run duration.
6. Capture the CPU and the memory utilization data for five runs on the container application.
 - The data can be captured through the *top* utility on Unix.

```
top -p <PID of container application> > top_application.txt
```

- The data can also be captured through *JConsole* or *JVisualVM* utilities.
7. Check the BW_JAVA_OPTS and other run-time parameters with which the container is running. It helps analyze the JVM parameters and other engine tuning parameters.
 8. Capture system configurations of servers such as CPU details, RAM, and number of cores where TIBCO BusinessWorks Container Edition container application, external services, and load generator are running. Capture details of */proc/meminfo* and */proc/cpuinfo* files.

```
cat /proc/meminfo and cat /proc/cpuinfo
```

Performance Improvement Use Cases

Given the wide range of use cases and even more complex and demanding scenarios that the platform can address, the default configuration of TIBCO BusinessWorks Container Edition might require some adjustments to reach optimal performance.

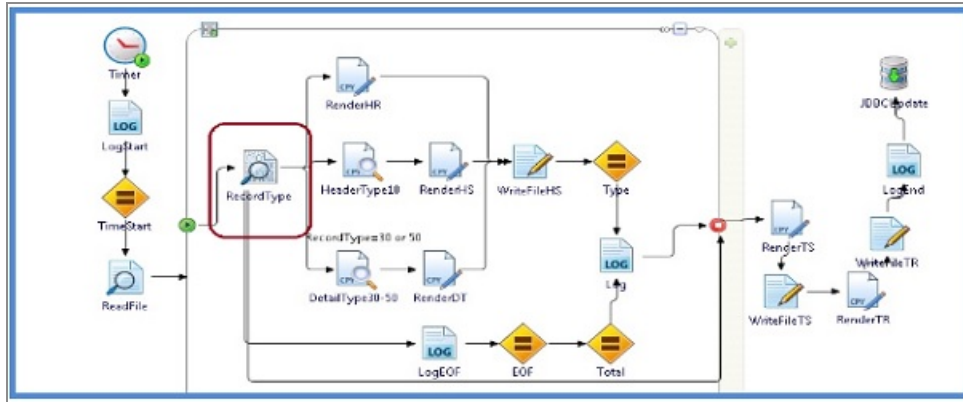
Performance Improvement Use Cases - Design Time and Deployment

To accomplish a business objective or requirement, designing the applications appropriately is an important aspect, as this impacts the overall performance of the application and the system as a whole, to some extent. With complexities being introduced in the way the processes in TIBCO BusinessWorks Container Edition are designed it is imperative that users are able to adhere to best practices which eventually lead to better end-to-end performance at run time. The use cases discussed here are based on the experience for large production implementations and proof of concept scenarios where implementing certain design changes helped improve the performance.

Usecase 1: Using File as the Input Type for Parse Data Activity

Use Case 1: A customer in the banking domain observed that the time taken to parse records was increasing with every record.

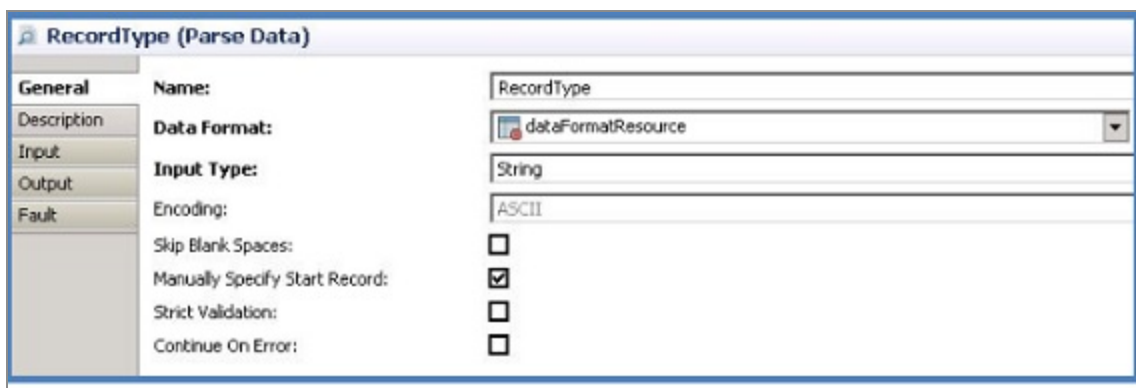
The process is shown in the following image:



The project is designed to parse data in an iterative manner from a file in the text format. This data is converted into a schema which is then evaluated against certain conditions, by the transitions. Based on specific values, the data is rendered in a particular format using the data conversion ActiveMatrix BusinessWorks plug-in. This parsed data is written to files and then eventually updated to the database.

Initial analysis showed that the overall high latency was due to the **Parse Data** activity highlighted in the above image.

Further analysis revealed that the time taken to parse the records was high since the input type of the **Parse Data** activity was configured to string, as displayed in the image below. When the input type is set to string, the entire contents of the source are read. For accessing specific records the search operation is performed in such a way that the entire source file is scanned.



The other option to configure the input type is file. When the input type is file the data is read from the source by means of a pointer. While accessing a specific range of records the search is performed based on the position of the pointer which makes the operation faster.

Testing and Measurement

The testing was focused on the aspects listed below:

- Comparative tests were conducted with input type for the **Parse Data** activity configured to string and file. The tests were performed to parse records in multiple iterations.
- The latency, memory and CPU utilization was measured.
- The overall latency was reduced by almost 10 times when the input type for the **Parse Data** activity was set to file.

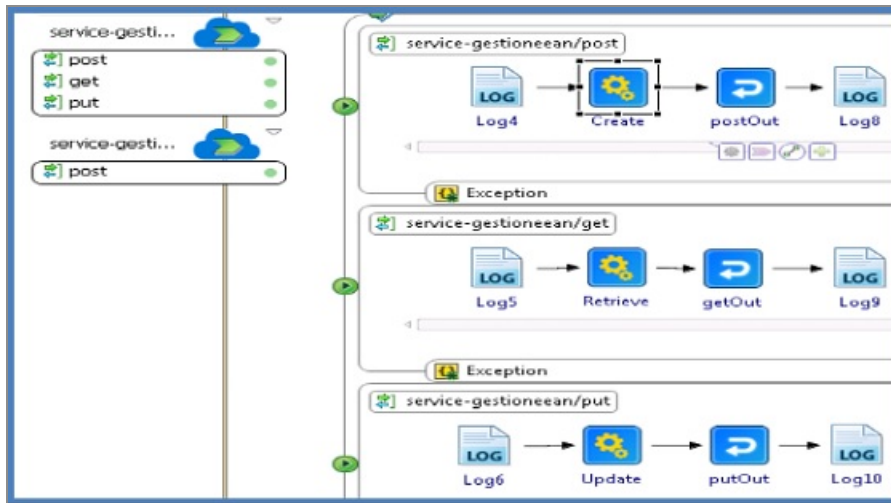
Solution for performance improvement

- TIBCO recommends that for faster processing, use the input type as file.
- In case of both the options, the input for **Parse Data** activity is placed in a process variable and this consumes memory. Hence large memory is required to read large number of records. To reduce memory usage, TIBCO recommends that a small set of records are read, parsed and processed before moving on to the next set of records.

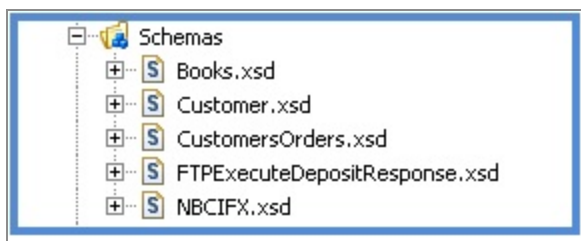
Usecase 2: Schema changes for improved performance

In a customer project comprising of multiple schemas (more than 50), it was observed that the latency for a single request -response was high, that is around few seconds.

The project mainly included multiple REST services as shown in the following image.



Some of the schemas in the project are shown in the following image.



Analysis confirmed that the schema operations were heavy with the current design implementation which contributed to the high latency.

Testing and Measurement

The testing was focused on the aspects listed below:

- With the default settings, a test was run for a single request and the total latency was measured from the logs.
- Few changes were made in the schema definition where the `include` tags were replaced with `import`, and the test was repeated and time was measured. For more information, see

Solution for performance improvement

- If the schemas in the project consist of `include`, these can be replaced with

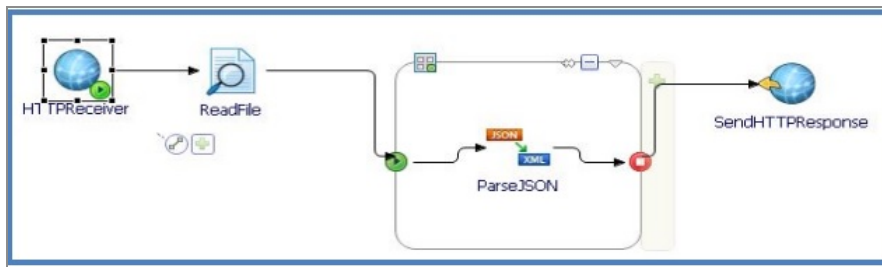
imports as shown in the example in the section, . This reduces the time considerably.

- This design implementation reduced the latency by almost 95 %.

Using XSD Schema Type for the Parse JSON activity

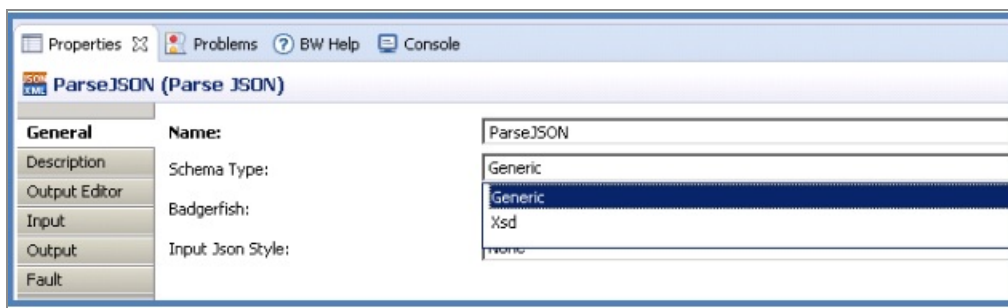
Use Case 3: In a customer usecase slow performance was observed in terms of latency, when the schema type for the **Parse JSON** activity was configured to **XSD** type as compared to **Generic** type. A comparison was done between the **XSD** and **Generic** type of schema for the **Parse JSON** activity.

The process is shown in the following image:



In this process, a JSON file consisting of multiple records or elements is read by the **Read File** activity, parsed by the **Parse JSON** activity in multiple iterations and then converted to XML.

The schema type for the **Parse JSON** activity can be configured to either **XSD** or **Generic** as shown in the following image:



The **Generic** type converts a JSON string to an XML string without using any schema for conversion. The **XSD** type converts a JSON string to an XML document defined using a schema specified in the Output Editor. The user may want to use the **Generic** type a specific schema is not required for conversion or the **XSD** type can be used when conversion needs to be done based on a particular schema.

Testing and Measurement

The testing was focused on the aspects listed below:

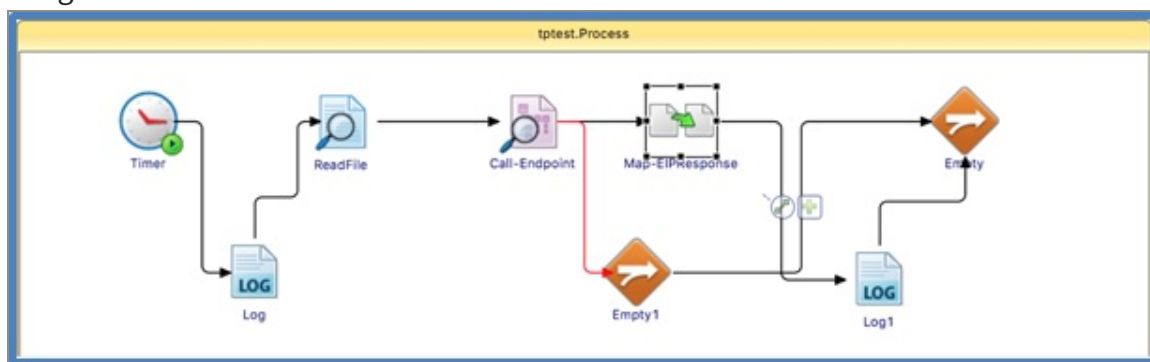
- With schema type as **XSD**, tests were run to process the records from the file and the total time was measured for the end to end process to complete.
- With schema type as **Generic**, tests were run to process the records from the file and the total time was measured for the end to end process to complete.

Solution for performance improvement

- It was observed that with the schema type configured to **Generic** the time taken is 50% less than the time taken when the schema type is configured to **XSD**. TIBCO recommends to configure the schema type to **Generic** for better performance.

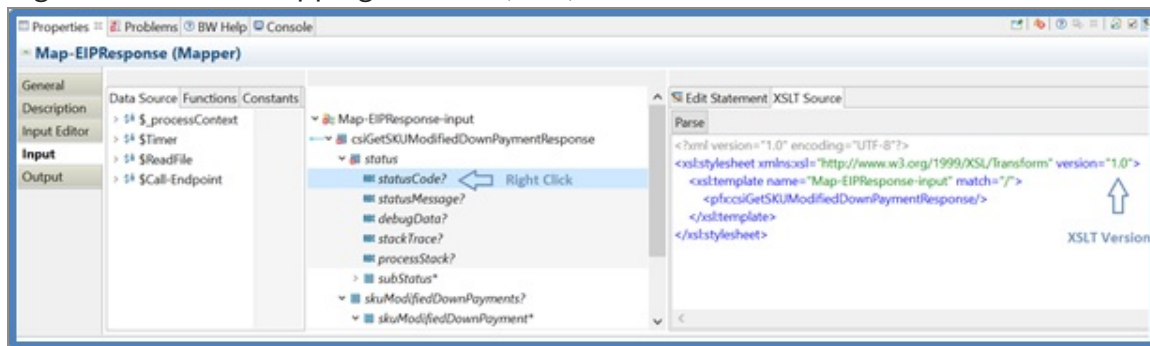
Usecase 4: Changing XSLT Version to Improve Latency

In a customer use case, slower performance was observed in terms of latency, when the XSLT source version for Mapper activity was 1.0 as compared to 2.0. The project is configured to read from an XML file, which is parsed by the Parse XML activity and the output content is further processed by the Mapper activity, as shown in the following image:



The XSLT source for 'Mapper' activity was set to the default 1.0 version as shown in the following image. This version can be configured as follows: Select **Activity** > **Input Tab** >

Right-click on the mapping element (RHS) > Select **Show Edit Tab** -> Select **XSLT Source**.



Testing and Measurement

The testing was focused on the aspect below:

- The tests were run to measure end to latency with XSLT version set to 1.0 and then with 2.0.

Solution for performance improvement

It was observed that the latency improved considerably by 100% when the version was changed to 2.0 from 1.0.

Usecase 5: Repetition Count Tuning for XML Authentication Policy

A use case was designed with XML authentication policy. For XML authentication policy, the username and password used during authentication are set in an XML file. This file consists of a parameter called the `repetitionCount`, which is the number of iterations used to compute the hash for the password. The higher the `repetitionCount`, the harder it becomes for an attacker to crack the password. However, using a higher repetition consumes more CPU time during the password verification. The default value is 1000. The following image shows an example of the XML file:

```
<?xml version="1.0" encoding="UTF-8"?>
<realm xmlns="http://xsd.tns.tibco.com/trinity/realm/2013" hashAlgorithm="PBKDF2WithHmacSHA256" repetitionCount="128">
  <users>
```

In this particular use case, it was observed that the throughput was low and the service

was not scalable although the resources were available.

Testing and Measurement

The testing was focused on the aspects below:

- Load tests were run with a fixed concurrency and the default `repetitionCount` (1000).
- The results provided very low throughput. This was analyzed and the analysis showed that the calls most of the time was spent in the calls related to computing the hash for the password.
- Since the hashing is determined by the `repetitionCount`, this parameter value was reduced to 1 and the tests were run with the same concurrency.

Solution for Performance Improvement

It was observed that setting the `repetitionCount` to 1 improved the throughput by almost 10 times.

Tools for Memory Monitoring, Tracking, and Analysis

For monitoring, tracking, and analyzing memory usage, the following utilities are available.

TOP Command for Memory Monitoring

The `top` command is used for memory monitoring. It works only on Linux platform.

The `top` command produces an ordered list of running processes selected by user-specified criteria, and updates it periodically. By default, ordering is by CPU usage, and it shows processes that consume maximum CPU. The `top` command also shows how much processing power and memory are being used, as well as the other information about the running processes.

Note: This utility works on Linux OS only.

The `top` command output monitors the RSS memory as well as the CPU utilization of the TIBCO BusinessWorks Container Edition AppNode.

```
top -p PID > top.txt
```

Sample output is as follows:

```
Cpu(s):  4.7%us,  1.1%sy,  0.0%ni, 94.1%id,  0.0%wa,  0.0%hi,  0.1%si,
0.0%st
Mem:  65914304k total, 59840516k used,  6073788k free,  3637208k buffers
Swap: 15359996k total,  119216k used, 15240780k free, 43597120k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
925	root	20	0	3650m	997m	27m	S	4.0	1.6	238:05.72	bwappnode-Http

Press 1 on same top output window and it would give usage per core

```
top - 02:13:25 up 160 days, 15:16, 26 users,  load average: 0.00, 0.00, 0.00
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
Cpu0  :  8.3%us,   6.7%sy,   0.0%ni, 85.0%id,   0.0%wa,   0.0%hi,   0.0%si,   0.0%st
Cpu1  :  4.7%us,   0.5%sy,   0.0%ni, 94.8%id,   0.0%wa,   0.0%hi,   0.0%si,   0.0%st
Cpu2  :  4.2%us,   0.4%sy,   0.0%ni, 95.1%id,   0.0%wa,   0.0%hi,   0.3%si,   0.0%st
Cpu3  :  3.8%us,   0.4%sy,   0.0%ni, 95.8%id,   0.0%wa,   0.0%hi,   0.0%si,   0.0%st

Mem: 65914304k total, 59839448k used, 6074856k free, 3637208k buffers
Swap: 15359996k total, 119216k used, 15240780k free, 43597124k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
  925 root        20   0 3650m 997m  27m S  4.7  1.6 238:08.62 bwappnode-Http
```

Native Memory Tracking

The Native Memory Tracking (NMT) is a Java HotSpot VM feature that tracks internal memory usage for a Java HotSpot VM.

Note: This utility works on Windows OS and Linux OS.

For more information about Native Memory Tracking, see docs.oracle.com.

Procedure

1. To enable native memory tracking on the JVM, set the parameter –
`XX:NativeMemoryTracking=summary` in the `BW_JAVA_OPTS` environment variable.
 For example: `BW_JAVA_OPTS="-Xmx1024M -Xms1024M -XX:NativeMemoryTracking=summary"`
2. Establish an early baseline. Use NMT baseline feature to get a baseline to compare by
 running command: `jcmd <pid> VM.native_memory baseline`.

3. Collect the memory data after starting the test runs by running `jcmd <pid> VM.native_memory summary`.
4. To monitor memory changes, use the following command: `jcmd <pid> VM.native_memory summary.diff`
5. If the application leaks a small amount of memory, it takes a while to show up. Comparing the memory pools from NMT output help identify the memory pool contributing to increase in memory.
6. The `jcmd` utility is shipped with JDK. It is under `$JDK_HOME/bin` directory.
For more information about `jcmd` utility, see [docs.oracle.com](https://docs.oracle.com/javase/8/docs/technotes/tools/windows/jcmd.html)

Jemalloc and Jeprof

The jemalloc tool tracks down native memory allocations and identifies native memory leak suspects. Use jeprof utility to analyze heap files.

Note: The jemalloc and jeprof utilities work on Linux OS only.

Procedure

1. Download jemalloc from [GitHub.com](https://github.com/jemalloc/jemalloc).
2. Install the tool by following the steps specified at [GitHub.com](https://github.com/jemalloc/jemalloc).
3. Once jemalloc is built, include jemalloc in BWCE buildpack.
4. Edit the `bwce-buildpack/resources/prestart.sh` file and add the following commands before the `exec ./tibco.home/bw*/*/bin/startBWAppNode.sh` command:

```
export LD_PRELOAD=/root/jemalloc/jemalloc-stable-4/lib/libjemalloc.so
```

```
export MALLOC_CONF=prof:true,lg_prof_interval:30,lg_prof_sample:17,prof_final:true,prof_leak:true
```

5. When the server is started and the memory allocation is done, `jeprof*.heap` files are generated at `./tibco.home/bw*/*/bin/` folder.

As the memory utilization grows, more files would be generated.

6. Analyze the heap files with the `jeprof` command. JDK needs to be installed or included in the buildpack for running `jeprof` commands as it uses local java for analyzing the `jeprof*.heap` files.

```
jeprof --show_bytes <PATH to java> jeprof*.heap
```

The `jeprof` utility is included in the `jemalloc/bin` folder. After execution of `jeprof` command, `jeprof` console opens.

7. Type `top` on the `jeprof` console. For example:

```
jeprof --show_bytes /usr/lib/jvm/java-8-oracle/jre/bin/java
jeprof*.heap
Using local file /usr/bin/w.
Using local file jeprof.19678.0.f.heap.
Welcome to jeprof! For help, type 'help'.
(jeprof) top
```

It shows the following output:

```
Total: 267184 B
258032 96.6% 96.6% 258032 96.6% _3_2_5
3616 1.4% 97.9% 3616 1.4% _nl_intern_locale_data
2048 0.8% 98.7% 2208 0.8% __tzfile_read
1024 0.4% 99.1% 1024 0.4% getpwnam
1024 0.4% 99.5% 1072 0.4% getpwuid
448 0.2% 99.6% 448 0.2% __gconv_lookup_cache
224 0.1% 99.9% 224 0.1% strdup
160 0.1% 99.9% 160 0.1% __tzstring
128 0.0% 100.0% 3760 1.4% _nl_load_locale_from_archive
48 0.0% 100.0% 48 0.0% get_mapping
```

8. To run the `jeprof` command on a single file, use the following command:

```
jeprof --show_bytes <PATH to java> <Heap file name>
```

After execution of `jeprof` command, `jeprof` console opens.

9. Type `top` on the `jeprof` console. For example:

```
jeprof --show_bytes /usr/lib/jvm/java-8-oracle/jre/bin/java
```

```
jeprof.19678.0.f.heap
Using local file /usr/bin/w.
Using local file jeprof.19678.0.f.heap.
Welcome to jeprof! For help, type 'help'.
(jeprof) top
```

It shows the following output:

```
Total: 267184 B
258032 96.6% 96.6% 258032 96.6% _3_2_5
3616 1.4% 97.9% 3616 1.4% _nl_intern_locale_data
2048 0.8% 98.7% 2208 0.8% __tzfile_read
1024 0.4% 99.1% 1024 0.4% getpwnam
1024 0.4% 99.5% 1072 0.4% getpwuid
448 0.2% 99.6% 448 0.2% __gconv_lookup_cache
224 0.1% 99.9% 224 0.1% strdup
160 0.1% 99.9% 160 0.1% __tzstring
128 0.0% 100.0% 3760 1.4% _nl_load_locale_from_archive
48 0.0% 100.0% 48 0.0% get_mapping
```

10. To stop profiling once the analysis is done and leak suspects are identified run command:

```
unset MALLOC_CONF
```

If profiling is not stopped, the jeprof heap files are continuously generated.

Detecting Increase in Heap Allocations with UMDH

The user-mode dump heap (UMDH) utility works with the Windows operating system to analyze the heap allocations for a specific process. UMDH utility is used to locate which routine in a specific process is leaking memory.



Note: The UMDH utility works on Windows OS only.

Before you begin

- Download and install the UMDH utility for Windows OS. For more information, see [Debugging Tools for Windows](#) from Microsoft documentation.
- Enable "Create user mode stack trace database" with `gflags.exe -i bwappnode-umdh.exe +ust` command. Get the process name from task manager.

```
C:\Program Files (x86)\Windows Kits\10\Debuggers\x64>gflags.exe -i
bwappnode-umdh.exe +ust
Current Registry Settings for bwappnode-umdh.exe executable are:
00001000
        ust - Create user mode stack trace database
```

- Before using UMDH, you must have access to the proper symbols for your application. UMDH uses the symbol path specified by the environment variable `_NT_SYMBOL_PATH`. Set the variable to a path containing symbols for your application.

If you also include a path to Windows symbols, the analysis is more complete. The syntax for this symbol path is the same as that used by the debugger.

For more information, see [Symbol Path for Windows Debugger](#) from Microsoft documentation.

For example, if the symbols for your application are located at `C:\MySymbols`, then to use the public Microsoft symbol store for your Windows symbols, using `C:\MyCache` as your downstream store, run the following command to set your symbol path:

```
C:\Program Files (x86)\Windows Kits\10\Debuggers\x64>set _NT_
_SYMBOL_
PATH=c:\mysymbols;srv*c:\mycache*https://msdl.microsoft.com/downloa
d/symbols
```

Procedure

1. Determine the process ID (PID) for the process to investigate.
For more information, see [Finding the process ID](#) from Microsoft documentation.
2. Analyze the heap memory allocations for the process before the memory leak is detected, and save it to a log file.
3. Collect the data at application start up before sending load.
For example, if the PID is 5872, and name of the log file is `log_before.txt`, use the

following command:

```
umdh.exe -p:5872 -f:log_before.txt
```

4. Use the UMDH utility to analyze the heap memory allocations for this process after the memory starts increasing, and save it to a log file.

5. Collect this data at regular intervals when an application starts leaking memory.

For example, if the PID is 5872, and name the log file is `log_after.txt`, use the following command:

```
umdh.exe -p:5872 -f:log_after.txt
```

6. The UMDH utility can compare two different log files and display the change in their respective allocation sizes. To redirect the results into a third text file, use the greater-than symbol (`>`). To convert the byte and allocation counts from hexadecimal to decimal, use the `-d` option.

For example, to compare `log_before.txt` and `log_after.txt` files, and save the results to the file `log_compare.txt`, use the following command:

```
umdh.exe -d log_before.txt log_after.txt > log_compare.txt
```

7. For each call stack that is labeled as "BackTrace" in the UMDH log files, there is a comparison made between the two log files. The snippet of the output is as follows:

```
// where:

//      BYTES_DELTA - increase in bytes between before and after log
//      NEW_BYTES - bytes in after log
//      OLD_BYTES - bytes in before log

//      COUNT_DELTA - increase in allocations between before and
//      after log
//      NEW_COUNT - number of allocations in after log
//      OLD_COUNT - number of allocations in before log

//      TRACEID - decimal index of the stack trace in the trace
//      database
//      (can be used to search for allocation instances in the
//      original
//      UMDH logs).
```

```

+      8856 ( 18400 - 9544)      12 allocs      BackTrace5
+         3 (      12 -      9)      BackTrace5      allocations

ntdll!RtlpAllocateHeap+2298
ntdll!RtlpAllocateHeapInternal+727
MSVCR100!malloc+5B
jvm!JVM_ResolveClass+387AE
jvm!???+0 : 53C415D6
jvm!JVM_GetManagementExt+6A5FF
jvm!JVM_GetManagementExt+786B1
jvm!JVM_GetManagementExt+7A162
jvm!JVM_GetManagementExt+CA4E
jvm!JVM_FindSignal+178329
jvm!JVM_FindSignal+1792E4
jvm!JVM_FindSignal+179491
jvm!JVM_FindSignal+17969F
jvm!JVM_GetManagementExt+82712
jvm!JVM_GetManagementExt+8305F
jvm!JVM_ResolveClass+5F5FF
jvm!JVM_FindSignal+68FA
MSVCR100!endthreadex+43
MSVCR100!endthreadex+DF
KERNEL32!BaseThreadInitThunk+14
ntdll!RtlUserThreadStart+21

```

This UMDH output shows that there were 8856 total bytes allocated from the call stack.

Memory Saving Mode

All activity output variables in a running process instance are stored in a memory, and hence consume memory. Memory saving mode allows memory used by an activity output variable to be released when the value of the variable is no longer needed.

In a memory saving mode, as each activity runs, the list of activity output variables is continuously evaluated to determine if subsequent activities in the process refer to the specific activity output variable. If no activities refer to the activity output variable, the memory used by it is released.

Memory Saving Mode can reduce the memory used by actively running process instances, as well as potentially improve the performance of checkpoints. By default, memory saving mode is enabled. This property enables the usage of memory saving mode, which frees activity output variables once they are no longer needed. The default value is true.

Memory saving is enabled at design-time and run-time.

To disable the Memory Saving Mode:

- For design-time: To disable the memory saving mode, unselect the **Save information to support memory saving mode** checkbox available at **Window > Preferences > BusinessWorks > Process Diagram** in the Memory Saving Mode section. Then, to remove the memory saving variable, right-click on **ActiveMatrix BusinessWorks™ Projects** and select **Refactor > Repair BusinessWorks Projects**. In the dialog, select the **Remove memory saving variables** option. On clicking the **Preview** button, the variables that can be removed from different activities are displayed on the Preview page, then select **OK**.
- For run-time: Configure the following `bwengine` property in the `BW_JAVA_OPTS` environment variable while running the application to disable the Memory Saving Mode:

```
bw.engine.enable.memory.saving.mode=false.
```

Performance Use Case - Memory Optimization

The service under test was a REST implementation with other activities like Mapper, Render JSON, Parse JSON, and Invoke REST API. The implementation had multiple mapper activities with iterator and accumulate output. The max heap of the TIBCO BusinessWorks™ Container Edition application was set to 4 GB for these tests. The testing was focused on analyzing the memory usage of the TIBCO BusinessWorks Container Edition application under load.

Memory usage with `bw.engine.enable.memory.saving.mode` set to false

The following snapshot shows the JVM snapshot of the TIBCO BusinessWorks Container Edition application under heavy load with memory saving set to false.



Memory usage with `bw.engine.enable.memory.saving.mode` set to true

The following snapshot shows the JVM snapshot of the TIBCO BusinessWorks Container Edition application under heavy load with memory saving set to true.



Performance findings

- Enabling memory saving reduced the heap usage of the application under heavy load.
- Enabling memory saving did not degrade the performance of deployed services in terms of latency and throughput.

i Note: The improvements showcased must be used as reference. The performance impact of enabling memory saving mode may vary based on service implementation, workload, and payload on the system.

References

- High Load: https://wiki.eclipse.org/Jetty/Howto/High_Load
- Saw tooth Shaped Graph: <http://stackoverflow.com/questions/7219532/why-a-sawtooth-shaped-graph>
- jvmtop utility: <https://github.com/patric-r/jvmtop>
- Debugging Java Native Memory Leaks: <https://www.evanjones.ca/java-native-leak-bug.html>
- Tracking Down Native Memory Leaks in Elasticsearch: <https://www.elastic.co/blog/tracking-down-native-memory-leaks-in-elasticsearch>
- Discrete sequential memory leak analysis with jemalloc and jeperf: <https://www.igorkromin.net/index.php/2018/06/21/discrete-sequential-memory-leak-analysis-with-jemalloc-jeperf/>

TIBCO Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the [Product Documentation website](#), mainly in HTML and PDF formats.

The [Product Documentation website](#) is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The following documentation for this product is available on the [TIBCO BusinessWorks™ Container Edition](#) page:

- *TIBCO BusinessWorks™ Container Edition Release Notes*
- *TIBCO BusinessWorks™ Container Edition Installation*
- *TIBCO BusinessWorks™ Container Edition Application Development*
- *TIBCO BusinessWorks™ Container Edition Application Monitoring and Troubleshooting*
- *TIBCO BusinessWorks™ Container Edition Bindings and Palettes Reference*
- *TIBCO BusinessWorks™ Container Edition Concepts*
- *TIBCO BusinessWorks™ Container Edition Error Codes*
- *TIBCO BusinessWorks™ Container Edition Getting Started*
- *TIBCO BusinessWorks™ Container Edition Migration*
- *TIBCO BusinessWorks™ Container Edition Performance Benchmarking and Tuning*
- *TIBCO BusinessWorks™ Container Edition REST Implementation*
- *TIBCO BusinessWorks™ Container Edition Refactoring Best Practices*
- *TIBCO BusinessWorks™ Container Edition Samples*

How to Contact Support for TIBCO Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our [product Support website](#).
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the [product Support website](#). If you do not have a username, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the [TIBCO Ideas Portal](#). For a free registration, go to [TIBCO Community](#).

Legal and Third-Party Notices

SOME CLOUD SOFTWARE GROUP, INC. (“CLOUD SG”) SOFTWARE AND CLOUD SERVICES EMBED, BUNDLE, OR OTHERWISE INCLUDE OTHER SOFTWARE, INCLUDING OTHER CLOUD SG SOFTWARE (COLLECTIVELY, “INCLUDED SOFTWARE”). USE OF INCLUDED SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED CLOUD SG SOFTWARE AND/OR CLOUD SERVICES. THE INCLUDED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER CLOUD SG SOFTWARE AND/OR CLOUD SERVICES OR FOR ANY OTHER PURPOSE.

USE OF CLOUD SG SOFTWARE AND CLOUD SERVICES IS SUBJECT TO THE TERMS AND CONDITIONS OF AN AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER AGREEMENT WHICH IS DISPLAYED WHEN ACCESSING, DOWNLOADING, OR INSTALLING THE SOFTWARE OR CLOUD SERVICES (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH LICENSE AGREEMENT OR CLICKWRAP END USER AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE SAME TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of Cloud Software Group, Inc.

TIBCO, the TIBCO logo, the TIBCO O logo, ActiveMatrix BusinessWorks, ActiveSpaces, Business Studio, TIBCO Business Studio, TIBCO Designer, Enterprise Message Service, Hawk, Rendezvous, and TIBCO Runtime Agent are either registered trademarks or trademarks of Cloud Software Group, Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only. You acknowledge that all rights to these third party marks are the exclusive property of their respective owners. Please refer to Cloud SG’s Third Party Trademark Notices (<https://www.cloud.com/legal>) for more information.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: <https://scripts.sil.org/OFL>

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

Cloud SG software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the “readme” file for the availability of a specific version of Cloud SG software on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. CLOUD SG MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S), THE PROGRAM(S), AND/OR THE SERVICES DESCRIBED IN THIS DOCUMENT AT ANY TIME WITHOUT NOTICE.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "README" FILES.

This and other products of Cloud SG may be covered by registered patents. For details, please refer to the Virtual Patent Marking document located at <https://www.cloud.com/legal>.

Copyright © 2015-2024. Cloud Software Group, Inc. All Rights Reserved.