VBackbone - Integration Platform

Author:

Version:

Table of Contents

[1 Objectives 4](#_Toc394387661)

[1.1 Context 4](#_Toc394387662)

[1.2 Data 4](#_Toc394387663)

[1.3 Target 4](#_Toc394387664)

[2 Convention 6](#_Toc394387665)

[3 Solutions 7](#_Toc394387666)

[3.1 Introduction 7](#_Toc394387667)

[3.2 Pivotal App Suite 7](#_Toc394387668)

[3.3 Tibco 7](#_Toc394387669)

[4 Scenario 8](#_Toc394387670)

[4.1 Purpose 8](#_Toc394387671)

[4.2 Workflow 8](#_Toc394387672)

[4.3 Stress 10](#_Toc394387673)

[4.4 Report 11](#_Toc394387674)

[5 Evaluation 12](#_Toc394387675)

[5.1 Criterias 12](#_Toc394387676)

|  |  |
| --- | --- |
| Document information | |
| **Document name** | VBackbone - Integration Platform |
| **Author** |  |
| **Creation date** |  |
| **Reviewed by** |  |
| **Approved by** |  |
| **Status** |  |

**Revisions history**

|  |  |  |  |
| --- | --- | --- | --- |
| Release | Author | Description | Version |
| 29/07/2014 | Nicolas LABROT | Initial version | 1.0 |

**Applicable Documents**

|  |  |  |  |
| --- | --- | --- | --- |
| N° | Name | Reference | Version |
| [AD1] | VBS Standard Model - Integration Platform Selection | VBS Standard Model - Integration Platform Selection-1.xlsx | 1.0 |

**Reference Documents**

|  |  |  |  |
| --- | --- | --- | --- |
| N° | Name | Reference | Version |
|  | N/A |  |  |

# Objectives

## Context

VBS has a lot of services, which do more or less the same tasks, have the same purpose. There is no real reuse. Service are specialized by customer and tightly coupled to the customer. Maintenance is not cost effective. On boarding of new customer is not efficient. New customer, new production lines often lead to duplicate services.

## Data

The 01/05/2013 the database contains, 50 millions of documents.

The database grows now with 2 millions documents per month. The first three days of each month there is peak with around 250000 new documents per day. The maximum for a day is 380000 documents.

The maximum number of job per day is 1300.

## Target

The final target is:

* To handle 5200 job per day;
* To handle 1500000 messages a day;
* To never lost a message;
* To be able to scale;
  + Load increase? 🡺 new nodes
* To implement an HA architecture.

The target is to develop common and general purpose services that can be reused. Service will be specialized by tasks, functions and features, not customer.

Service must be link together with an orchestration layer and must be loosely coupled. Messaging system will do the glue, will be the backbone of the VBS production line.

# Convention

All date/timestamp are in ISODateTimeWithMs in UTC (eg. “2014-07-16T16:23:45.536Z").

# Solutions

## Introduction

We will benchmark two solutions:

* OpenSource and developer centric solution using frameworks mastered at VBS: **Pivotal App Suite**;
* One of the leading closed source solution provider which provide integration and orchestration platform: **Tibco BusinessWorks**.

## Pivotal App Suite

### Information

**Messaging System** : RabbitMQ

**Enterprise Integration Pattern**: Spring Integration

**Batch framework** : Spring Batch

### Environment

* 3 VM on CentOS 6.5 64bits for RabbitMQ Server
* 2 VM on CentOS 6.5 64bits for the client
* 4GB of RAM
* 10GB of local storage
* 1Gb/s network adapter

## Tibco

### Information

**Messaging system:** Tibco EMS

**Integration platform:** Tibco BusinessWorks 6

### Environment

* X VM on CentOS 6.5 64bits for EMS
* X VM on CentOS 6.5 64bits for BusinessWorks
* XGB of RAM
* XGB of local storage
* 1Gb/s network adapter

# Scenario

## Purpose

Create a simple orchestration workflow. Evaluate the cost of a robust workflow creation.

Benchmark the tools : throughput, latency, infrastructure footprint.

Stress the tools : load, stress, ageing, failover.

## Workflow



### Injector

**Injector** is a parametrized perl script. It can generate X dataset as ASCII files. Each file (dataset) contains Y documents (one per line) with a size per document of Z. The filename is “[DATASET\_ID].dta”.

File size can be modulated with “Y” and “Z” parameters.

**Injector** is called every X seconds using crontab. Injector generates files in « /mnt/vbackbone/hotfolder » folder.

For each file, **Injector** creates the report folder structure:

* « /mnt/vbackbone/reports/[DATASET\_ID] »
* « /mnt/vbackbone/reports/[DATASET\_ID]/{poller, rpcclient, rpcserver, split, composition1, composition2, composition3, composition4} »

**Injector** copy too the generated file in « /mnt/vbackbone/reports/[DATASET\_ID]/ ».

### Poller

**Poller** polls every X seconds the hotfolder folder. For each file it sends a message to the queue « hotfolder » with

* Header: {id: [DATASET\_ID], timestamp: [now()]}
* Payload: content of the file

**Poller** generates a flag in « /mnt/vbackbone/reports/[DATASET\_ID]/poller/[UID].txt »

### RPC Client

**RPC client** subscribes to the « hotfolder » queue. For each message it makes a RPC call to the RPC server. The request is sent to the « rpc.request » queue with:

* Header: {id: [DATASET\_ID], timestamp: [now()], replyTo : « rpc.reply », correlationId : XXX}
* Payload: content of the file

“Reply to” channel is set using the « replyTo » header. Correlation between request and reply is done with a correlation id.

**RPC client** generates a flag in « /mnt/vbackbone/reports/[DATASET\_ID]/rpcclient/[UID].txt ».

**RPC client** copy the payload to « /mnt/vbackbone/reports/[DATASET\_ID]/rpcclient/[UID].dta ».

### RPC Server

**The RPC server** subscribes to the « rpc.request » queue. For each message, it waits 20ms +/- 10ms and send the reply to the queue corresponding to the value of the « replyTo » header with

* Header: {id: [DATASET\_ID], timestamp: [now()], correlationId: XXX}
* Payload: payload of the request

**RPC server** generates a flag in « /mnt/vbackbone/reports/[ DATASET\_ID]/rpcserver/[UID].txt »

### Split

**Split** subscribes to the « rpc.reply » queue. After receiving the reply, it parses the payload. For each documents, it sends compositions request to composition1, composition2, composition3, composition4 queues with:

* Header: {id: [DATASET\_ID], timestamp: [now()], documentId: [DOCUMENT\_ID]}
* Payload: A PDF of roughly 100ko

**Split** generates a flag in « /mnt/vbackbone/reports/[DATASET\_ID]/split/[UID].txt »

### Composition clients

Each composition client subscribes to its corresponding queue. For each message it waits 20ms +/- 10ms.

Composition generates a flag in « /mnt/vbackbone/reports/[ DATASET\_ID]/composition[X]/[DOCUMENT\_ID]\_ [UID].txt ».

Composition copy the payload in « /mnt/vbackbone/reports/[ DATASET\_ID]/composition[X]/[DOCUMENT\_ID]\_ [UID].dta »

### About flags

All flags files contains three properties:

* send: timestamp when the message has been sent by the previous step;
* begin: timestamp when the message has been received;
* end: timestamp when the step end.

## Stress

Maximum number of dataset per day is not relevant. What we will test is an increasing number of documents per dataset to reach the target.

### Load

Test how the platform reacts with an increasing volume of data.

**Injector** will generate chronologically 5200x6 datasets with 50, 100, 150,200, 250, 300 documents per dataset.

### Break

Test the platform with a very high volume of data.

**Injector** will generate 5200 datasets with 600 documents per dataset.

### Ageing

Test the platform during a long period of time.

**Injector** will continuously generate datasets with between 50 to 300 documents per dataset.

### Failover

Test the platform with random network and server failures.

Message lost: Break the network between the **RPC server** and the others. Kill the **RPC server**.

Contention point: Break the network between the **Split** and the **composition clients**.

Connection recovery: Break the network between the **broker cluster** and the others.

Connection recovery / master election: Break the network between the **cluster master** and the others. Kill the **cluster master**.

Cluster availability: Break the network between a **cluster node**. Kill **a cluster node**.

## Report

All flags files contains three properties:

* send: timestamp when the message has been sent by the previous step;
* begin: timestamp when the message has been received;
* end: timestamp when the step end.

send: 2014-07-16T16:23:45.536Z

begin: 2014-07-16T16:23:45.536Z

end: 2014-07-16T16:23:45.536Z

**Report** parses « /mnt/vbackbone/reports » directory and it generates:

* Per job consistency reports
  + per step expected output / actuals output : missing flag, doublon flag
* Performance report
  + Per job execution duration
  + Per step throughput,
  + Transition latency between two steps

**Report** generates excels file.

# Evaluation

## Criterias

Criterias are detailed in the [AD1].

TBD