Estimating the Cost for Executing Business Processes in the Cloud

Vincenzo Ferme, Ana Ivanchikj, Cesare Pautasso

Faculty of Informatics, USI Lugano, Switzerland

Abstract. Managing and running business processes in the Cloud changes how Workflow Management Systems (WfMSs) are deployed. Consequently, when designing such WfMSs, there is a need of determining the sweet spot in the performance vs. resource consumption trade-off. While all Cloud providers agree on the pay-as-you-go resource consumption model, every provider uses a different cost model to gain a competitive edge. In this paper, we present a novel method for estimating the infrastructure costs of running business processes in the Cloud. The method is based on the precise measurement of the resources required to run a mix of business process in the Cloud, while accomplishing expected performance requirements. To showcase the method we use the BenchFlow framework to run experiments on a widely used open-source WfMS executing custom workload with a varying number of simulated users. The experiments are necessary to reliably measure WfMS's performance and resource consumption, which is then used to estimate the infrastructure costs of executing such workload on four different Cloud providers.

Keywords: Cloud Resource Cost · Cloud BPM · Business Process Execution · Performance Benchmarking · Workflow Management System

1 Introduction and Motivation

According to the recent trend of Cloud Business Process Management [16], users may move the execution of their Business Processes (BPs) to the Cloud, by deploying a Workflow Management System (WfMS) on rented Cloud infrastructure, a Cloud model known as Infrastructure as a Service (IaaS). A WfMS deployed in the Cloud (i.e., a Cloud WfMS) can deliver elastic scalability in response to dynamic workload changes, which is one of the main motivating factors for moving to the Cloud. In the IaaS context, it is not only important to determine the inherent performance of the WfMS executing the BPs, but also to measure and analyse the corresponding resource consumption, so that an expected level of performance can be guaranteed while keeping costs to the minimum. The focus of this paper is not on optimizing the BP execution in the Cloud, which has received its due attention [1,24]. Instead, we focus on analysing Cloud WfMS's performance [21,25] and resource consumption [11]. Both aspects are relevant for estimating the infrastructure costs of running BPs in the Cloud.

Cloud providers introduce cost models [13] with different sizing of the available resources, granularity of the utilization period, and performance guarantees [15]. In this paper, we present a novel method for estimating Cloud infrastructure costs based on precise measures of the resources (CPU, RAM, Database (DB) Size) consumed to run a mix of realistic BPs with a variable number of simulated users. Such measures are necessary for in-depth analysis of WfMS's efficiency in using Cloud resources and to map how well WfMSs can fit into existing Cloud cost models. To show-case the proposed method, we apply it on workloads executed on Camunda[1], a wide-spread open-source WfMS with numerous customers in different sectors. Previous experiments [25] with three open-source BPMN2.0 WfMSs have indicated Camunda's stable behaviour, both in terms of performance and resource utilization, which makes it a good candidate for Cloud deployment, thus motivating us to use it as the System Under Test (SUT) in this work. Then we map its resource utilization to the expected cost of renting it on four different Cloud providers, i.e., Amazon EC2, Microsoft Azure, Google Cloud and Springs.io, implementing five diverse cost models.

Given the inherent variability of the IaaS Cloud providers' performance [23], we run the experiments in a private Cloud, whose controlled environment makes it possible to guarantee performance measurements' reliability and replicability [9], providing, what can be considered, a baseline for the results obtained in the Cloud. Our assumption is that we have obtained sufficient information for an initial estimation of Cloud costs, and for reducing the set of experiments one would have to perform directly on the best matching Cloud instances.

The remainder of the paper is structured as follows. Sec. 2 explains the proposed Cloud infrastructure cost estimation method, while Sec. 3 defines some useful metrics to be used in the proposed method. Sec. 4 describes the performed experiments in terms of their setup and the experiment environment, while Sec. 5 presents the results from the calculated metrics. Sec. 6 offers an in depth discussion and mapping of those results to the costs of running BPs on the Cloud. Sec. 7 presents related work and Sec. 8 describes the threats to validity of the proposed method. Sec. 9 concludes the paper.

2 Cloud Infrastructure Cost Estimation Method

Before estimating any costs, it is necessary to determine what influences them. In the case of Cloud infrastructure, the direct influence comes from the Cloud providers' pricing policy which uses computing resources (e.g., CPU, RAM) to distinguish among different pricing packages. When executing BPs in a Cloud infrastructure, the necessary resources are determined by: a) the WfMSs input, i.e., the complexity and size of the executed BPs as well as the number of users and the frequency with which they instantiate the BPs, and b) the end user's execution performance requirements, e.g., reducing latency or improving throughput might require higher computational resources.

[1] `https://camunda.org/`

Having this in mind, the cost estimation method we propose is comprised of the following steps: 1) determine the mix of BPs, i.e., the workload mix you plan to execute in the Cloud (Sec. 4.1.1), 2) determine the number of users and the frequency in which they will start the business process instances (BPIs) (Sec. 4.1.2), 3) decide the execution performance requirements you are interested in and how you can measure them (Sec. 3.1, Sec. 4.2.3), 4) run experiments in a stable and noise protected environment using the input determined in steps one and two (Sec. 4.2), 5) analyse experiments' results to determine the resources necessary to achieve the desired performance indicators (Sec. 5), 6) map the necessary resources to the pricing packages of Cloud providers (Sec. 6), 7) select the Cloud providers' offerings that minimise your costs while maximising your resource usage efficiency (Sec. 6), and 8) test and analyse in detail the narrowed selection of IaaS offerings. In the rest of the paper we will show-case the applicability of the proposed method by using BenchFlow, a dedicated performance framework [8], for running a set of realistic experiments on Camunda and mapping the results to a selected set of Cloud IaaS providers.

3 Measurements and Metrics

Performance requirements have to be measurable. Thus, selecting and defining both performance metrics and resource consumption metrics is necessary before applying the method described in Sec. 2. In this section we present a non-exhaustive list of possible metrics to use during the experiments, derived from our experience in benchmarking the performance of BPMN2.0 WfMSs [25]. In order to obtain statistically relevant and reliable results, each experiment is comprised of multiple trials. Thus, the raw data for the metrics are gathered separately for each trial and then aggregated to compute experiment-level metrics.

3.1 Performance Metrics

When testing a WfMS, its performance can be evaluated at the BPI level or at workload mix level. At **BPI level** we obtain as raw data from the DB used by the WfMS, the duration of each BPI execution (D) in milliseconds (ms), which we use to calculate the aggregated metrics among different trials of the experiment. Such metrics include: 1) the *weighted average of the duration* - $wavg(D)$, where the weights are computed based on the number of executed BPIs in each trial; 2) the *minimum, maximum duration* - $min(D), max(D)$ across trials; and 3) the *range of the quartiles of the duration* - $Q1(D), Q2(D), Q3(D)$ which is calculated as the minimum, maximum value of the quartiles among the different trials. The Q1, Q2 and Q3 quartiles show under which value does 25%, 50% and 75% of the data fall [18, Chap.6].

The performance metrics that we evaluate at **workload mix level** based on raw data from the DB are: 1) the *number of BPIs* - $avg(N)$ executed during the experiment; and 2) the *throughput* - $avg(T)$, i.e., the number of executed BPIs per second (s). For each of these metrics we calculate the average among the experiment trials with 95% confidence interval (*ci*), as well as the standard deviation (*sd*). The *ci* is used to set up a range of likely values for the analysed metric in which we can be 95% confident [18, Chap.8].

Based on data from Faban[2], one of BenchFlow's components [8], we additionally calculate the *weighted average of the requests sent by the users per second* - $wavg(REQ/s)$ using the number of requests per trial as weights and the *weighted average response time* - $wavg(RT)$ to the BP instantiation requests in millisecond, where the weight is based on the number of BP instantiation requests in the different trials.

3.2 Resource Consumption Metrics

The resource consumption metrics are particularly important for Cloud deployment due to the Cloud providers' pricing models which uses them as billing base, with CPU, RAM and Disk space being the most frequently used ones [13]. Since BenchFlow [8] uses Docker containers to deploy the WfMS, we obtain the raw data regarding the resource utilisation from the Docker Stats API[3]. CPU and RAM are continuous variables, thus we calculate the expected value of their total usage per trial using the *integral over time* - $avg(itg(CPU))$, $avg(itg(RAM))$ [18, Chap.4]. We apply the trapezoidal rule to approximate the definite integral.

To analyse the WfMS's resource allocation efficiency we use the *weighted average of the efficiency of CPU and RAM usage* - $wavg(e(CPU))$, $wavg(e(RAM))$. The efficiency is computed as the ratio between the $itg(CPU)$, $itg(RAM)$ and the product of the $max(CPU)$, $max(RAM)$ and the number of data points used to calculate that integral, respectively for CPU and RAM. The weighting per trial is based on the mentioned number of data points. This ratio has values between 0 and 100%, with values closer to 100% indicating balanced and thus efficient use of the CPU and RAM without significant changes over time. We also compute the *weighted average CPU, RAM* - $wavg(CPU)$, $wavg(RAM)$ among different trials in percentage (%) for the CPU and in MB for the RAM. The weights are calculated based on the number of CPU, RAM data points per trial.

To observe the dynamics of the CPU/RAM change over time we provide the *maximum CPU, RAM* - $max(CPU)$, $max(RAM)$ metric. Furthermore, we present the *range of the quartile* - $Q1(CPU)$, $Q2(CPU)$, $Q3(CPU)$ *and* $Q1(RAM)$, $Q2(RAM)$, $Q3(RAM)$ calculated as described in the BPI level performance metrics.

The Disk space refers to the occupied space to store the execution data in the DB of the WfMS. We obtain the raw data from the DB information schema by adding the space occupied by the data to the space occupied by the DB indexes used by the WfMS. This is feasible given that Camunda uses MySQL. We calculate the *average Disk space* - $avg(DS)$ among trials with 95% *ci* and *sd*.

[2] `http://faban.org`

[3] `https://docs.docker.com/engine/reference/api/docker_remote_api_v1.22/#get-container-stats-based-on-resource-usage`

4 Experiments Definition

Setting up a performance experiment requires defining: 1) the workload, i.e., the necessary input to the WfMS, and 2) the execution environment of the experiments, i.e., the private Cloud infrastructure and the minimal resources required to execute the workload [9]. This means defining the factors that influence the resource consumption, and the infrastructure costs as mentioned in Sec. 2.

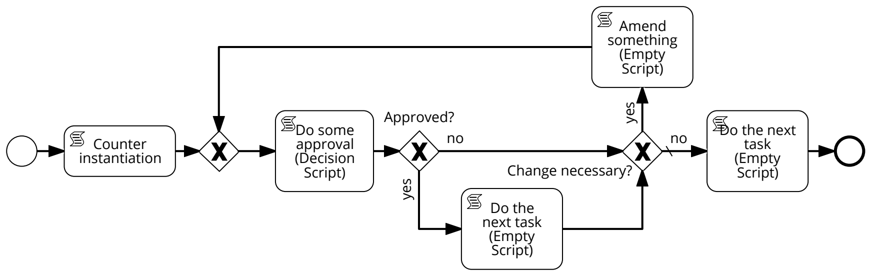
4.1 Workload Definition

The parameters of the workload (workload mix, load functions, and test data) are generic and applicable to different SUTs. However, their specific characteristics depend on SUT's functionality, as well as the experiments' goals. When the SUT is the WfMS, the workload mix refers to the BP models to be executed in the WfMS during the experiments, the load functions define the frequency of BP instantiation and the distribution of executed control flow paths, while the test data might be necessary to start a BPI or during its execution, depending on the BP model characteristics [9].

4.1.1 The Workload Mix In practice, it is challenging to obtain BP models from industry due to their confidentiality. Alternatively, using a workload mix comprised of workflow patterns would result in very simple models, while the synthesis of arbitrary models would not result in a realistic workload. Therefore, we decided to reuse models included in the demonstrations and performance benchmarking suites conducted by vendors, in particular Camunda[4] and Activiti[5]. In order to stay focused on the WfMS's performance, we needed to adjust vendor's models by removing data flows and replacing any external interaction elements (such as message events, pools, Web service tasks, user tasks) with control flow elements internal to the BPs. Within the original control flow structure, Web service tasks and user tasks have been replaced with empty script tasks, except for the scripts necessary to randomly determine the execution path following branching gateways. The duration of timer events has been arbitrarily set to one minute. Message flows have been replaced with control flows to isolate the impact of external interaction. In models where messages are used as boundary events, they have been replaced with exclusive or inclusive gateways, depending on whether an interrupting or non-interrupting boundary event had been used. Furthermore, since loops introduce non-deterministic behaviour which can impact the average duration of the BPI execution and the resources it uses, we limit, using a counter, the number of iterations to a minimum of zero and a maximum of two.

In real-world usage the WfMS deploys concurrently different BP models with different level of complexity. Thus, we use a workload mix comprised of five realistic models with different complexity and different set of used BPMN 2.0 constructs. The smallest BP model is presented in Fig. 9, while all the executable

[4] `https://github.com/camunda/camunda-consulting`
[5] `http://www.slideshare.net/alfresco/introduction-to-activiti-bpm`


Counter instantiation
Do some approval (Decision Script)
Approved?
no
yes
Do the next task (Empty Script)
Change necessary?
yes
Amend something (Empty Script)
no
Do the next task (Empty Script)
Fig. 1. The smallest business process model in the workload mix

models part of the workload mix are available at `http://benchflow.inf.usi.ch/bpm2016` for reproducibility purposes. All models use what zur Muehlen and Recker [20] call the BPMN Common Core constructs (i.e., normal flow, tasks, start/end event and exclusive gateway), while pools have been deliberately omitted to ensure executability of the models. In addition, some of the models also use sub-processes, loops, timer events, terminate end events, inclusive, and parallel gateways. The smallest BP model has 21 elements (both nodes and edges) as evident in Fig. 9, while the largest one has 84 elements, thus they are coherent with findings of empirical studies on modeling practices [5]. Previous experiments with running only individual models vs. running a mix of individual models have revealed comparable execution results [25], thus we have decided to only run experiments where the five models are uniformly represented in the workflow mix: each model is used to instantiate approximately 20% of the BPIs.

4.1.2 The Load Functions With WfMS as SUT there are two types of load functions: the load start and the load distribution functions [9]. Due to the unavailability of execution logs for these particular models, in the **load distribution function** we use random load distribution for the diverging paths with equal probability of choosing among alternative paths. The **load start function**, on the other hand, is defined by the load time (or steady state), the ramp-up period, the number of users and the think time. We use 10 minutes of load time and 30 seconds of ramp-up period. This means that all users become gradually active within 30 seconds, while the BP instantiation requests are being sent for 10 minutes. The load time decision is based on the fact that some Cloud service providers charge a minimum of 10 minutes of Virtual Machine (VM) usage with 1 minute increments thereafter. Furthermore, as described by Skouradaki et al. in [25], such a short load time was appropriate to find significant performance bottlenecks, thus making it suitable for realistic tests that provide insight on the WfMS performance behaviour. The actual duration of the experiment depends on the execution time of the started BPIs, and thus might be longer than 10 minutes. Previous work [8] has shown that changing the number of simulated users impacts the WfMS's performance behaviour. Thus, to reflect realistic usage of WfMSs by differently sized companies or companies which are evaluating their growth strategy, we have decided to simulate 50, 500 and 1'000 users. The think time is the waiting time between a new request the user issues to the SUT, and the time in which the response to the previous request has been received. In this case, the requests refer to instantiation of a new BPI. We use a think time of 1 second, which may or may not reflect real-world workloads, but it serves the purpose of stressing the SUT.

4.2 Experiments Environment

To ensure reliability of the results and more vast exploration of the cost analysis space, automation of the experiments' execution is required. For that purpose we have developed BenchFlow, a dedicated framework for benchmarking WfMSs performance [8], which we use for running the experiments. It automates the configuration and the deployment of the benchmarked WfMSs and its DB. Faban is used to generate the workload.

4.2.1 Execution Environment Configuration To ensure reproducible initial conditions and minimal interferences, the WfMS, the DB and Faban, are all deployed in Docker images [17] on dedicated servers using the Docker Engine 1.9.1 and Ubuntu 14.04.3 LTS (GNU/Linux 3.13.0-40-generic x86_64) as operating system. They interact through two networks of 10Gbit/s each, one dedicated to the communication between the WfMS and the DB and the other one dedicated to other interactions (e.g., issuing the load). The WfMS and the DB run on exclusively dedicated servers. The WfMS on a server with 64 Cores (2 threads) and a clock speed of 1'400MHz mounting 128GB of RAM and a magnetic disk with 15'000 rpm. The DB on a server with 64 Cores (2 threads) and a clock speed of 2'300MHz mounting 128GB of RAM and a SSD SATA disk. Faban's Load Drivers are placed on three servers: one with 64 Cores (2 threads) and a clock speed of 2'300MHz mounting 128GB of RAM, the second with 48 Cores (2 threads) and a clock speed of 2'000MHz mounting 128GB of RAM, and the third with 12 Cores (1 thread) at 800MHz mounting 64GB of RAM. With this resource allocation we have ensured and verified that the DB would not become a performance bottleneck during the experiment.

4.2.2 Workflow Management System Configuration The experiments are run on Camunda 7.4.0. placed in a Docker container with Ubuntu 14.04.01 as operating system and the Oracle Java Server 7u79 VM, run using the host network to avoid performance overhead in the network communication [7]. Standalone deployment is used and the WfMS is configured in accordance with Camunda's web-site suggestions, using Camunda's official Docker image[6]. MySQL Community Server 5.7.10 is used as a DB and is installed on a separate Docker container[7]. WfMS's connection to the DB is through the MySQL Connector/J 5.1.33 with minimum 10 idle connections, maximum 100 connections and an initial thread pool size of 10. The history is set at full level.

4.2.3 Experiments Setup: Resource Allocation Limits While the experimental testbed provides enough capacity to process the workload, deploying

[6] `https://hub.docker.com/r/camunda/camunda-bpm-platform/`
[7] `https://hub.docker.com/_/mysql/`

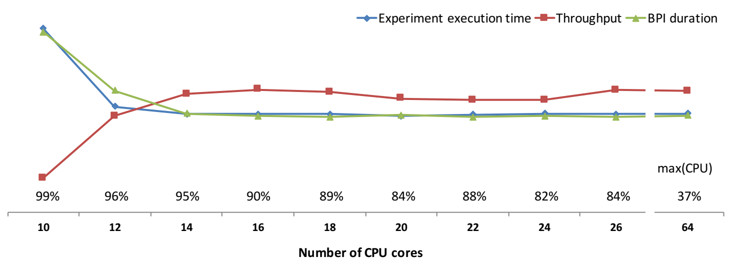


Fig. 2. Normalized performance over number of CPU cores with 500 Users

Table 1. **B**ounded and **U**nbounded resource limits

	Users	**WfMS CPU**	**WfMS RAM**	**DB CPU**	**DB RAM**
B	50	6 Cores	1 GB	6 Cores	2 GB
	500	16 Cores	2 GB	16 Cores	10 GB
	1’000	24 Cores	2 GB	24 Cores	12 GB
U	50, 500, 1’000	64 Cores	128 GB	64 Cores	128 GB

the system in the Cloud requires precise definition of the resources (e.g., CPU, RAM, Disk space) needed by the system to operate under the expected workload and in accordance with the expected performance behaviour. To do so, we first run an **unbounded resource experiment (U)** using the full available capacity (64 CPU cores and 128 GB RAM) for both the WfMS and the DB, for each of the numbers of simulated users (50, 500, 1’000). The purpose is to determine a baseline for WfMS’s performance under different workloads without saturating the system, i.e., step three of the proposed cost estimation method: determining the execution performance requirements (Sec. 2).

Since RAM usage was relatively stable during the **U** experiments, we set it to the amount of GB closest (round half to even) to the maximum used during the **U** experiments and then kept it as a fixed variable when searching for the minimum CPU cores required for obtaining a comparable performance to the one with unbounded resources. We run the experiments with the workload described in Sec. 4.1.1 and in Sec. 4.1.2. We start from the minimum required CPU cores, verified from the fact that the system is saturated with peaks of maximum CPU usage which reach 99% of the available CPU. Then we gradually increment the available CPU Cores and compare the WfMS’s performance results as well as the number of requests per second and the response time to the ones from the **U** experiments. We set the bound at the number of CPU Cores at which the performance metrics start to converge towards the **U** experiments performance results, while the maximum CPU usage is no more than 90% of the available CPU. By doing so we provide a 10% buffer given the intrinsic variability of the system behaviour and the low, but still present non-determinism of the choices of executed paths in the workflow mix. For space reasons we only show the decision graph (Fig. 2) for the experiments run with 500 users, but the same method has been used with 50 and 1’000 users as well. The selected CPU and RAM limits for the **bounded resource experiment (B)** are presented in Table 1.

5 Experiments Results and Discussion

Each of the experiments described in Sec. 4 is comprised of three trials. In each trial, to consider only the steady state of the WfMS, we discard all data for the first five BPIs of each model in the mix, since they have higher duration caused by the warming up of the SUT. We report the results of the experiments in Table 2 (performance metrics of Sec. 3.1) and Table 3 (resource consumption metrics of Sec. 3.2). Results related to the resource utilization efficiency are reported in Table 4. As evident from the tables, the method for identifying the resource boundaries for the **B** experiments (see Sec. 4.2.3), allowed us to obtain comparable performance between the **B** and the **U** executions. The same applies for the resources utilization. In the **B** experiments we see an increased, but not yet high CPU efficiency utilization, while for the RAM it is comparable to the one experienced in the **U** experiments.

Table 2. Performance Metrics Results

		Business Process Instance Level Metrics						***Workload Mix Level Metrics***			
	Users	**wavg(D)** [ms]	**min(D)** [ms]	**max(D)** [ms]	**Q1(D)** [ms]	**Q2(D)** [ms]	**Q3(D)** [ms]	**avg(N)** [bpi]	**avg(T)** [bpi/s]	**wavg(REQ/s)**	**wavg(RT)** [ms]
U	**50**	8’238.13	0	84’568	[1-1]	[2-2]	[3-3]	30’623±22	44.45±0.13	48.85	22.97
	500	9’148.13	0	143’006	[1-1]	[2-2]	[3-3]	272’910±6’024	395.90±8.47	434.14	152.18
	1’000	64’023.83	0	888’118	[1-1]	[2-2]	[3-3]	323’783±5’643	329.81±2.73	512.71	946.27
B	**50**	8’337.41	0	82’855	[1-1]	[2-2]	[3-3]	30’590±43	44.38±0.18	48.84	24.53
	500	9’079.07	0	191’536	[1-1]	[2-2]	[3-3]	273’116±3’063	396.21±4.79	435.27	149.21
	1’000	65’772.55	0	899’168	[1-1]	[2-2]	[3-3]	328’248±2’268	329.80±3.83	519.14	921.57

Table 3. Resource Consumption Metrics Results

	Users	**wavg(CPU)** [%]	**max(CPU)** [%]	**Q1(CPU)** [%]	**Q2(CPU)** [%]	**Q3(CPU)** [%]	**avg(DS)** [MB]
U	**50**	1.61	23.45	[0.64-0.64]	[1.00-1.10]	[2.46-2.54]	840.08±27.68
	500	10.41	36.81	[0.00-0.01]	[14.13-14.27]	[16.06-16.20]	7’012.21±150.97
	1’000	8.92	38.51	[0.83-0.85]	[3.80-4.45]	[18.01-18.39]	8’505.81±236.15
B	**50**	9.06	98.73	[5.26-6.94]	[6.33-8.31]	[8.06-11.19]	836.92±14.01
	500	33.00	94.17	[0.07-0.11]	[43.63-45.61]	[49.22-50.16]	7’305.91±164.49
	1’000	21.41	84.29	[2.13-2.20]	[8.82-10.98]	[43.44-44.15]	8’962.10±170.36
		wavg(RAM) [MB]	**max(RAM)** [MB]	**Q1(RAM)** [MB]	**Q2(RAM)** [MB]	**Q3(RA)** [MB]	**sd(avg(DS))**
U	**50**	706.74	823.38	[638.38-652.39]	[672.44-695.44]	[793.72-806.34]	23.97
	500	998.55	1’163.00	[871.87-878.01]	[998.45-1027.59]	[1’120.07-1’131.58]	130.74
	1’000	1’100.73	1’199.59	[1’031.25-1’053.59]	[1’131.85-1’172.45]	[1’132.37-1’173.28]	204.51
B	**50**	720.07	870.58	[643.66-683.78]	[679.18-716.48]	[759.07-776.83]	12.13
	500	998.97	1’157.24	[876.00-900.38]	[1’000.98-1’011.42]	[1’120.86-1’129.12]	142.45
	1’000	1’100.46	1’189.20	[1’033.29-1’044.38]	[1’150.02-1’165.71]	[1’150.73-1’167.72]	147.53

Regarding the WfMS resource utilization, the disk utilization, as expected, grows with the number of executed BPIs, as evident from the $avg(N)$ and the

$avg(DS)$ metrics. The RAM has a more stable utilization than the CPU, as evident from the $Q1, Q2, Q3$ quartiles of the distribution of the weighted average utilization of the two resources. For the RAM the mentioned quartiles are close to the maximum ($max(RAM)$) value. This means that the distribution

Table 4. Resource Utilisation Results

	Users	**avg(itg(CPU))** **[%*s]**	**sd(avg(itg(CPU)))**	**wavg(e(CPU))** **[%]**	**avg(itg(RAM))** **[MB*s]**	**sd(avg(itg(RAM)))**	**wavg(e(RAM))** **[MB]**
U	**50**	1'149.12±14.51	12.56	7.16	508'873.29±4'274.71	3'702.01	86.89
	500	9'397.68±252.41	218.59	29.13	902'479.20±26'337.98	22'809.36	86.81
	1'000	11'343.29±243.09	210.52	23.41	1'400'305.12±35'146.50	30'437.76	92.95
B	**50**	6'502.40±947.91	820.92	10.41	519'663.08±15'952.49	13'815.26	83.66
	500	29'014.12±549.12	475.56	36.31	878'879.99±7'452.63	6'454.17	86.49
	1'000	26'902.98±128.00	110.85	26.56	1'384'193.35±25'070.42	21'711.62	95.43

of RAM usage during the experiment is comparable to the maximum amount needed by Camunda to handle the constant load issued during the experiment. The achieved efficiency of RAM utilization is greater with greater number of users. In Table 4 we also report the efficiency of CPU utilization. It ranges from 10.41 to 36.31 over the **B** experiments, largely far from what can be considered a good efficiency. This is also evident from the CPU quartiles' values, that are distant from the $max(CPU)$ values, meaning that the CPU utilization has spikes leading to a very high maximum utilization, while the average utilization is much lower. This behaviour is more evident with the **B** workload with 50 users, where Camunda experiences spikes of CPU utilization over 98% while the average value is 10.41%. High efficiency in resource utilization is very relevant in the Cloud context, since it enables the selection of cheaper resources that are efficiently utilized. CPU spikes are particularly deleterious because they require buying more resources that are not efficiently utilized. We have analysed the CPU utilization during the entire load issued to the WfMS, and we have noticed that the spikes occur when the load starts, and the system is warming up, thus needing more resources to execute the requests. The CPU utilization then stabilizes, but still shows some spikes during the entire execution time. Investigating the reasons behind it requires more invasive techniques that go beyond the scope of this paper. However, it is worth mentioning that the experienced spikes, other than the warm up ones, are unexpected given the characteristics of the workload mix, which regardless of some non deterministic choices, is expected to have a constant resource demand under constant load.

Regarding the scalability in the number of users, we can see that Camunda, using the default configuration, experiences a decrease in performance when the number of users increases. This is evident from the $wavg(D)$ metric that increases marginally between 50 and 500 users, and significantly between 500 and 1'000 users. The main reason behind this behaviour is the increase in response time ($wavg(RT)$) that leads to a reduced number of issued start requests per second ($wavg(REQ/s)$). The identified scalability bottleneck clearly does not depend on the unavailability of resources, since as evident from the CPU and RAM quartiles metrics in Table 3, the WfMS had sufficient resources to handle the issued workload. Moreover, we have also verified that the same was true for the DB connected to the WfMS, and that the network was not saturated.

6 Cloud Providers Costs for Various Workloads

Now that we have determined the minimal necessary resources for executing our workload, and analysed Camunda's performance behaviour (see Sec. 5), we can go on with mapping them to Cloud providers' offerings. In the analysed Cloud providers we include what Gartner defines as "leaders" in its 2015 Magic Quadrant for Cloud IaaS report, i.e., Amazon EC2 and Microsoft Azure, as well as the "visionary" Google Cloud, and the newcomer Springs.io. They all use slightly different cost models, offering different flexibility both in renting resources and in the time unit used for billing. The frequently used term "pay-as-you-go" can be misleading on what is actually charged. Amazon[8] and Azure[9] offer "instances" with predefined allocated resources, billed per hour, in the case of Amazon, and per minute, in the case of Azure. Google[10], in addition to the predefined instances, also enables customers to define their custom instance, which although more flexible, is not entirely elastic given that the number of CPUs (or virtualized CPUs) must be even, and the RAM memory per CPU must be between 0.9GB and 6.5GB, while being a multiple of 256MB. Google charges per minute for both types of instances, however, the minimal time that can be charged is set to 10 minutes. Springs.io[11], on the other hand, charges by hour, but with less limitations on the selected resource bounds. It charges for CPU speed which can be incremented in steps of 50MHz within the range between 500MHz and 20'000MHz, while the RAM can be incremented in steps of 128MB within the range between 256MB and 32'768MB.

The mapping of providers' offerings to the resource requirements has been mainly driven by the minimal WfMS's CPU requirements as defined with the bounded resource experiments. This means that the instances presented in Table 5 offer at least the same CPU and RAM as used in the experiment. The prices are on per hour basis, as per provider's official websites[12], for Linux operating system and based on West US region. When multiple instances satisfied the minimal CPU requirement, the most economical one was selected. Since Springs.io uses a concept of simulated core in MHz (core-MHz), core-2GHz of speed are mapped to 1 CPU Core based on the processor used as per Springs.io's documentation. This translates to a maximum of 10 CPU Cores given its limit in maximum number of MHz available. The limit of 10 CPU Cores is not sufficient computational power for running the workloads with 500 and 1'000 users. Thus, Table 5 only contains Springs.io's cost for 50 simulated users. The stated prices do not include any additional charges for storage or data transfer, since we only focus on the CPU and RAM needed by the WfMS.

[8] `https://aws.amazon.com/ec2/pricing/`
[9] `https://azure.microsoft.com/en-us/pricing/details/virtual-machines`
[10] `https://cloud.google.com/compute/pricing`
[11] `http://springs.io/pricing-list/`
[12] The prices in Table 5 are from March 2016 and are subject to change.

Table 5. Selected Cloud Providers' Instances: Resources and Prices[13]

	Cloud Provider	Instance type	CPU	Memory (GB)	Price (USD/hr)
50 Users	Actually used	N/A	5.92 Cores	0.85	N/A
	Amazon (Am)	Compute Optimised - c4.2xlarge	8 Cores	15	0.419
	Azure (Az)	General purpose - basic tier - A4	8 Cores	14	0.376
	Google Predefined (Gp)	High-CPU - n1-highcpu-8	8 Cores	7.2	0.232
	Google Custom (Gc)	N/A	6 Cores	5.4	0.25827
	Springs.io (S)	**N/A**	**12 GHz**	**1**	**0.107**
500 Users	Actually used	N/A	15.07 Cores	1.13	N/A
	Amazon (Am)	Compute Optimised - c4.4xlarge	16 Cores	30	0.838
	Azure (Az)	Compute Optimised - D5 v2	16 Cores	56	1.17
	Google Predefined (Gp)	**High-CPU - n1-highcpu-16**	**16 Cores**	**14.4**	**0.464**
	Google Custom (Gc)	N/A	16 Cores	14.4	0.68872
	Springs.io (S)	N/A	N/A	N/A	N/A
1'000 Users	Actually used	N/A	20.23 Cores	1.16	8.75
	Amazon (Am)	Compute Optimised - c4.8xlarge	36 Cores	60	1.675
	Azure (Az)	Performance optimized compute - G5	32 Cores	448	8.69
	Google Predefined (Gp)	**High-CPU - n1-highcpu-32**	**32 Cores**	**28.8**	**0.928**
	Google Custom (Gc)	N/A	24 Cores	21.6	1.03308
	Springs.io (S)	N/A	N/A	N/A	N/A

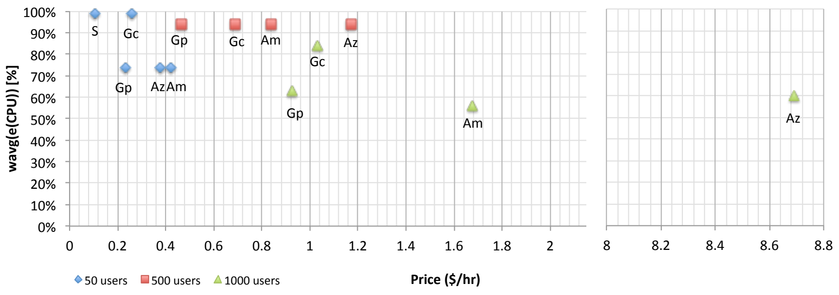


Fig. 3. CPU Usage Efficiency vs Cloud Provider Costs per Workload

Additionally, for each provider we have calculated the CPU usage efficiency as a ratio between the actual used CPU during the experiments, mentioned in Table 5, and the CPU in the provider's instance. As was to be expected for the smallest workload of 50 users, the most flexible provider, Springs.io, offers the best price and the most efficient CPU usage, but it is interesting to see that the custom instances of Google, which offer greater CPU usage efficiency, are less expensive than the predefined instances of Amazon and Azure. The CPU usage efficiency shown in Fig. 3 in relation to the Cloud provider costs for different workload sizes, is also an indicator of how flexible the existing Cloud offerings are. It is evident that for big workload with 1'000 concurrent users, only with the Google custom instance the CPU is used efficiently, since the predefined instances are not flexible enough and do not offer any instances which have between 16 and 32 CPU Cores. As the size of the workload doubles from 500 to 1'000 users, so does the price of the best offer for the minimal required instances. On the other hand, when going from 50 to 500 users, i.e., a ten-fold workload increase, the price becomes three times higher. Thus, the increase in the best offered

[13] The best offered price for each workload is marked in bold

price, marked in bold in Table 5, is not linear to the increase in the number of users while keeping the workload mix fixed. The most significant spread of costs is noticeable for the largest workload where Azure only offers one type of instance, which is over eight times more expensive than competitors' offerings due to the high RAM and Disk space available, which are actually not needed for the executed workload. For the 500 users workload, although the CPU usage efficiency is equal among all the providers since they all offer instances with 16 CPU Cores, the ratio between the highest and the lowest price is 1.5 times. In addition to CPU usage differences, from Table 5 it is evident that all analysed Cloud providers, except for Springs.io, are not flexible with the available amount of RAM, which for all workloads is much higher than the actually needed one. If Springs.io increases the CPU it offers, it might be the case that its cost model would be the most convenient one in the future, both in terms of resource usage efficiency and in terms of costs.

Although the duration of our experiments is less than one hour and the hardware is different from what is offered by Cloud providers, we expect similar trends of CPU, RAM usage for longer running experiments in the Cloud. Thus, we find the price per hour in relation to the efficiency of CPU usage calculated based on the experiments, an appropriate indicator for selecting the most suitable Cloud providers for further analysis directly on the selected providers.

7 Related Work

Cloud BPM - In the context of Cloud BPM, users pay for the sustained usage of the Cloud BPM solution, or for renting the IaaS on which they install the WfMS of interest. All Business Activity Monitoring (BAM) measures related to "process instance times" [19] have been already applied in the literature to improve the performance of BP and scientific workflows execution in the Cloud, especially for what concerns the performance of service based and/or computationally intensive processes [14,24,1]. Janiesch et al. [6] rely on the BPI execution time information to propose a BPM-aware scaling mechanism to scale the resources available to services connected to the BPI, with the goal of improving the turnaround time of executed BPs. The proposed scaling mechanism monitors Cloud resources (mainly the CPU) and performance/Service Level Agreement (SLA) measures, to optimize the execution of service-based BPs. It is evident from the discussed literature that the recent trends towards Cloud WfMSs [4] have introduced many challenges [2], such as the need of a comprehensive evaluation of WfMS's and BP's performance, to better quantify and evaluate the effectiveness of moving the BP execution to the Cloud. To optimise time and cost savings by moving to the Cloud, Han et al. [12] propose a Hybrid architecture of Cloud BPM, where depending on activity's computational-intensity and data sensitivity, an optimisation algorithm determines the place of its execution, i.e., the Cloud or an on premises server. More recently, Gómez Sáez et al. [11] have started evaluating the cost of running scientific workflows on different Cloud providers using a similar cost model. However, they use a scientific WfMS and a workload comprised of a single workflow and 10 simulated users, while we use a more diverse workload mix and simulate a variable number of users. In addition to latency, we compute more detailed metrics not only concerning the resource consumption, but also the WfMS's performance. Furthermore, we do not go into analysing the different categories of instances offered by the Cloud providers, but limit our analysis to the cheapest instance per provider that would be sufficient to provide a sufficient amount of resources for the given workload.

WfMS Performance Benchmarking - In the performance benchmarking area, we refer to the work on benchmarking as means to improve WfMS's performance, and the work on reporting WfMSs' resource usage metrics. Weikum et al. [10] propose a benchmark for comparing the performance of different commercial WfMSs by measuring their throughput to study the impact of the database component. They also derive some useful lessons learned for better characterization and improvement of the benchmarked WfMSs performance. Roller [22] proposes a comprehensive study on an internally developed WfMS, with focus on WfMS's throughput. The author relies on benchmarking and proposes different optimization and caching techniques to improve system's performance. Most of the remaining related work on WfMS performance benchmarking, refers to performance benchmarking using black box approaches and considers WfMS's throughput and latency as performance metrics. Only few of them present performance metrics in terms of resource consumption for executing BPs [21,25].

Brebner and Liu [3] analyse costs of using Cloud IaaS for a service application. However, they only obtain data on the CPU resource consumption, later used to map the consumption to the cost for different instance types offered by the analysed Cloud providers. Our goal is not comparing the costs of different Cloud providers for an arbitrary application. We target a specific middleware, the WfMS, and investigate the relation between the diverse performance and resource consumption of different workloads and the costs of deploying them on different Clouds. To do so, we rely on performance benchmarking research and technologies to benchmark the performance of the WfMS's core components and their intrinsic resource consumption as a system running processes. We rely on BAM research and technologies, outside and inside the Cloud, to define the relevant metrics for characterizing the performance, the resource consumption, and consequently the cost of the WfMSs which are part of our study.

8 Threats to Validity

Construct Validity - We conduct our experiments on one WfMS in its default configuration, since it is the first one utilized by practitioners to evaluate system's performance, a standalone deployment, a single workload mix and different workloads. In the analysis, we only consider the WfMS resources, but a similar approach can be followed for the corresponding DB. To reduce measurement noise, we perform experiments using lightweight Docker containers that are not deployed in a virtualized environment.

Internal Validity - The experiments we perform are inherently subject to variability in obtained metrics value, due to the many factors impacting the runtime of a software system. We mitigate this variability by performing multiple trials for each of the experiments, and we verify the variance among trials in order to provide reliable measures validated by descriptive statistics.

External Validity - The method we propose for estimating the cost of executing the BPs on the Cloud by precisely measuring the resources needed by the WfMS running them, is limited in generalizability by the performance variability in a public Cloud and the different hardware on the Cloud instances compared to the one we have used. Cloud prices and cost models are frequently changed by competing Cloud providers. This may affect the obtained ranking. We are aware of this limitation, and thus we propose the current method as only the first step towards evaluating the cost of running BPs in the Cloud which reduces the set of experiments to be performed directly on the Cloud.

9 Conclusion and Future Work

In this work we have introduced a novel method for estimating the costs of running BPs in the Cloud. We have applied it by running experiments with different workloads on Camunda, a widely used open-source BPMN 2.0 WfMS.

Considering the CPU and RAM bounds determined with the experiments we have surveyed four Cloud providers for best fitting offers. A lack of flexibility concerning resource size and granularity in the offerings has been noted, especially for the largest workload of 1'000 concurrent users, where predefined Cloud instances are too big, while the offerings with real flexibility in terms of CPU vs. RAM combinations include maximum CPU bounds that are too low.

Due to the extreme variability of public Cloud performance [23] and the difference in hardware of the rented Cloud resources, our approach contributes the necessary first step towards measuring the actual cost of executing BPs in the Cloud, and limiting the number of Cloud instances to be involved in actual experiments in the Cloud. In the near future we aim to perform additional experiments in the identified Cloud instances using Cloud benchmarking techniques [3,23], so that we can validate our method Moreover, we plan to apply the proposed method to other BPMN 2.0 WfMSs and to different deployment alternatives, in order to observe differences in resource utilization among the WfMSs, which might lead to different Cloud providers being suitable for different WfMSs subject to different workloads. Lastly, we plan to extend the workflow mix to include Web service calls, events and human tasks to provide a more comprehensive evaluation of the resource needed by all WfMS components.

Acknowledgments This work is partially funded by the Swiss National Science Foundation with the BenchFlow - A Benchmark for Workflow Management Systems (Grant Nr. 145062) .

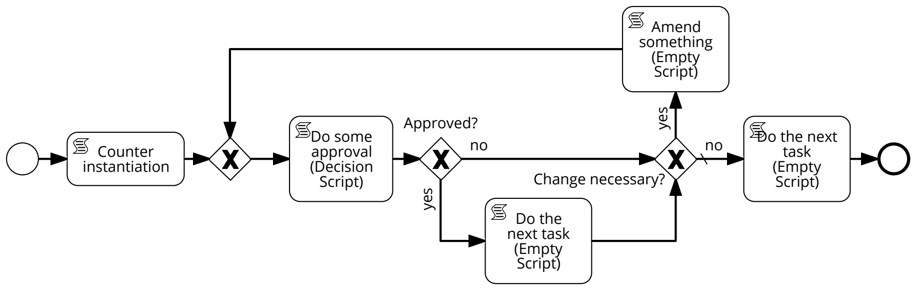
References

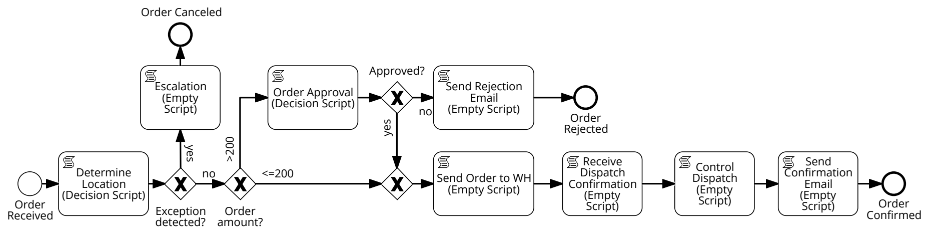
1. Alkhanak, E.N., Lee, S.P., Khan, S.U.R.: Cost-aware challenges for workflow scheduling approaches in Cloud computing environments: Taxonomy and opportunities. Future Generation Computer Systems 50, 3–21 (2015)

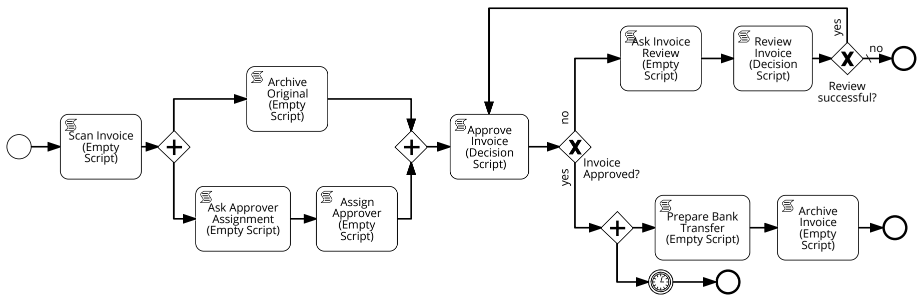
2. Baeyens, T.: BPM in the Cloud, vol. 8094, pp. 10–16. Springer (2013)
3. Brebner, P., Liu, A.: Modeling cloud cost and performance. In: Proc. of Cloud Computing and Virtualization Conference (CCV 2010), Singapore (2010)
4. Cantara, M.: The state of the bpm platform cloud market (id: G00209943) (2011), `https://www.gartner.com/doc/1520715/state-bpm-platform-cloud-market`
5. Chinosi, M., Trombetta, A.: Bpmn: An introduction to the standard. Computer Standards & Interfaces 34(1), 124–134 (2012)
6. Euting, S., et al.: Scalable business process execution in the Cloud. In: Proc. of IC2E '14. pp. 175–184 (March 2014)
7. Felter, W., Ferreira, A., Rajamony, R., Rubio, J.: An updated performance comparison of virtual machines and linux containers. Tech. rep., IBM (July 2014)
8. Ferme, V., et al.: A framework for benchmarking BPMN 2.0 workflow management systems. In: Proc. of BPM '15. pp. 251–259. Springer (2015)
9. Ferme, V., et al.: A container-centric methodology for benchmarking workflow management systems. In: Proc. of CLOSER'16. Springer (2016)
10. Gillmann, M., et al.: Benchmarking and Configuration of Workflow Management Systems. In: Proc. of CoopIS '00. pp. 186–197 (2000)
11. Gómez Sáez, S., et al.: Performance and Cost Evaluation for the Migration of a Scientific Workflow Infrastructure to the Cloud. In: Proc. of CLOSER 2015. pp. 1–10. SciTePress (May 2015)
12. Han, Y.B., Sun, J.Y., Wang, G.L., Li, H.F.: A cloud-based BPM architecture with user-end distribution of non-compute-intensive activities and sensitive data. Journal of Computer Science and Technology 25(6), 1157–1167 (2010)
13. Höfer, C., Karagiannis, G.: Cloud computing services: taxonomy and comparison. Journal of Internet Services and Applications 2(2), 81–94 (2011)
14. Janiesch, C., et al.: Optimizing the performance of automated business processes executed on virtualized infrastructure. In: Proc. of HICSS. pp. 3818–3826 (2014)
15. Lenk, A., et al.: What are you paying for? performance benchmarking for infrastructure-as-a-service offerings. In: Proc. of CLOUD '11. pp. 484–491 (2011)
16. Liu, X., Yuan, D., Zhang, G., Li, W., Cao, D., He, Q., Chen, J., Yang, Y.: The design of Cloud workflow systems. Springer (2011)
17. Merkel, D.: Docker: Lightweight linux containers for consistent development and deployment. Linux J. 2014(239) (Mar 2014)
18. Montgomery, D.C., Runger, G.C.: Applied Statistics and Probability for Engineers. John Wiley and Sons (2003)
19. zur Muehlen, M., Shapiro, R.: Business process analytics. In: Handbook on Business Process Management 2, pp. 137–157. Springer (2010)
20. Muehlen, M., Recker, J.: How much language is enough? theoretical and practical use of the business process modeling notation. In: Proc. of CAiSE 2008. pp. 465–479. Springer (2008)
21. Röck, C., et al.: Performance benchmarking of BPEL engines: A comparison framework, status quo evaluation and challenges. In: Proc. of SEKE. pp. 31–34 (2014)
22. Roller, D.H.: Throughput Improvements for BPEL Engines: Implementation Techniques and Measurements applied in SWoM. Ph.D. thesis, USTUTT (2013)
23. Schad, J., et al.: Runtime measurements in the cloud: observing, analyzing, and reducing variance. Proceedings of the VLDB Endowment 3(1-2), 460–471 (2010)
24. Schulte, S., Janiesch, C., Venugopal, S., Weber, I., Hoenisch, P.: Elastic business process management: State of the art and open challenges for BPM in the cloud. Future Generation Computer Systems 46(0), 36–50 (2015)
25. Skouradaki, M., et al.: Micro-benchmarking BPMN 2.0 workflow management systems with workflow patterns. In: Proc. of CAiSE'16. Springer (2016)

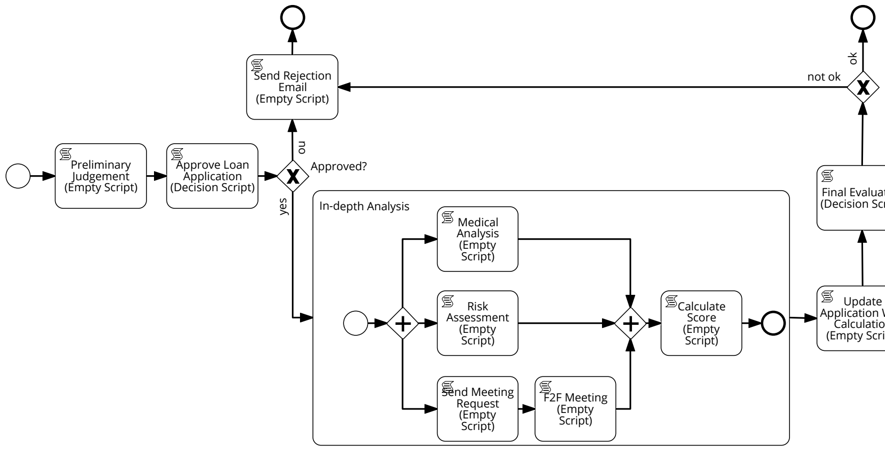
Appendix - Workload Mix Models

Due to space limitations we include the five BPMN process models we used in the experiments as part of this appendix. The executable models can be downloaded from `http://benchflow.inf.usi.ch/bpm2016`.

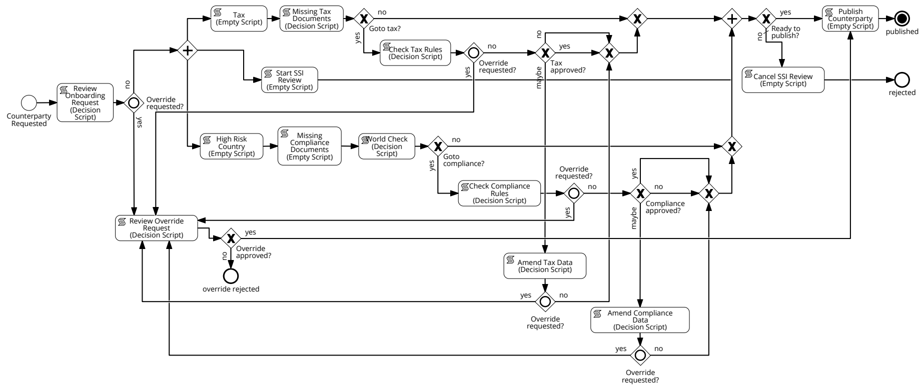

Amend something (Empty Script)
Counter instantiation
Do some approval (Decision Script)
Approved?
no
yes
Do the next task (Empty Script)
Change necessary?
yes
no
Do the next task (Empty Script)


Order Canceled
Escalation (Empty Script)
Order Approval (Decision Script)
Approved?
Send Rejection Email (Empty Script)
no
Order Rejected
yes
>200
yes
Order Received
Determine Location (Decision Script)
no
<=200
Exception detected?
Order amount?
Send Order to WH (Empty Script)
Receive Dispatch Confirmation (Empty Script)
Control Dispatch (Empty Script)
Send Confirmation Email (Empty Script)
Order Confirmed


Ask Invoice Review (Empty Script)
Review Invoice (Decision Script)
yes
no
Review successful?
Archive Original (Empty Script)
Scan Invoice (Empty Script)
Approve Invoice (Decision Script)
no
Invoice Approved?
yes
Ask Approver Assignment (Empty Script)
Assign Approver (Empty Script)
Prepare Bank Transfer (Empty Script)
Archive Invoice (Empty Script)



Preliminary Judgement (Empty Script)
Approve Loan Application (Decision Script)
Approved?
no
yes
Send Rejection Email (Empty Script)
In-depth Analysis
Medical Analysis (Empty Script)
Risk Assessment (Empty Script)
Send Meeting Request (Empty Script)
F2F Meeting (Empty Script)
Calculate Score (Empty Script)
Update Application With Calculation (Empty Script)
Final Evaluation (Decision Script)
not ok
ok



Counterparty Requested
Review Onboarding Request (Decision Script)
Override requested?
no
yes
Tax (Empty Script)
Missing Tax Documents (Decision Script)
Goto tax?
no
yes
Check Tax Rules (Decision Script)
Override requested?
no
yes
Tax approved?
no
yes
maybe
Start SSI Review (Empty Script)
High Risk Country (Empty Script)
Missing Compliance Documents (Empty Script)
World Check (Decision Script)
Goto compliance?
no
yes
Check Compliance Rules (Decision Script)
Override requested?
no
yes
Compliance approved?
yes
no
maybe
Ready to publish?
yes
no
Publish Counterparty (Empty Script)
published
Cancel SSI Review (Empty Script)
rejected
Review Override Request (Decision Script)
Override approved?
yes
no
override rejected
Amend Tax Data (Decision Script)
yes
no
Override requested?
Amend Compliance Data (Decision Script)
yes
no
Override requested?