

BUILDING SCALABLE WORKFLOWS USING MICROSERVICES ARCHITECTURE

Abstract

This paper focusses on the need to leverage microservices architecture for building scalable business workflows leveraging open-source workflow engines that are BPMN2.0 compliant

BUILDING SCALABLE WORKFLOWS USING MICROSERVICES ARCHITECTURE

Today every business needs to support workflow capability part of its processes and often, the development teams jump at designing a ground-up solution which looks simply to start-with but gets unwieldy as the complexity starts building up in the codebase resulting in a maintenance nightmare.

WORKFLOW FRAMEWORKS TO THE RESCUE

In order to avoid reinventing the wheel, it is often a better choice to rely on some of the time-tested opensource workflow frameworks like Flowable, Camunda to the rescue. While the idea of the workflow frameworks to model a business process is nothing new, but deploying the workflows as a microservices is the focus for this paper. In fact, the Microservices architecture and workflows are great combination as it helps the architects to design more elegant and scalable solutions to business processes that are long-running requiring orchestration of many steps involving end-user and system actions.

Some of the key advantages that we have seen from our experience implementing the workflows as microservices are

- Clear separation on the business boundary (bounded context) as each microservice owns the business process, data and the corresponding business rules.
- Each microservice can be scaled based on the business need as they are deployed in the Kubernetes environment with the right sizing and auto-scale based on thresholds.
- Each microservice is responsible to move the workflow stages through its endpoint

- Each microservice consumes the BPMN2.0 compliant XML file from the workflow engine/framework thus ensuring that the model can be shown to the functional users to explain the business process anytime while the dev-team relies on the XML that is imported into the microservice project.
- As the microservice architecture ensures a separate repository and a CI-CD pipeline, this provides the required atomicity to manage the build, deploy, test and release of the workflow in lower and production environment by the team that owns.

ENCAPSULATING THE WORKFLOW ENGINE AND WRAPPING IT AS A MIROSERVICE

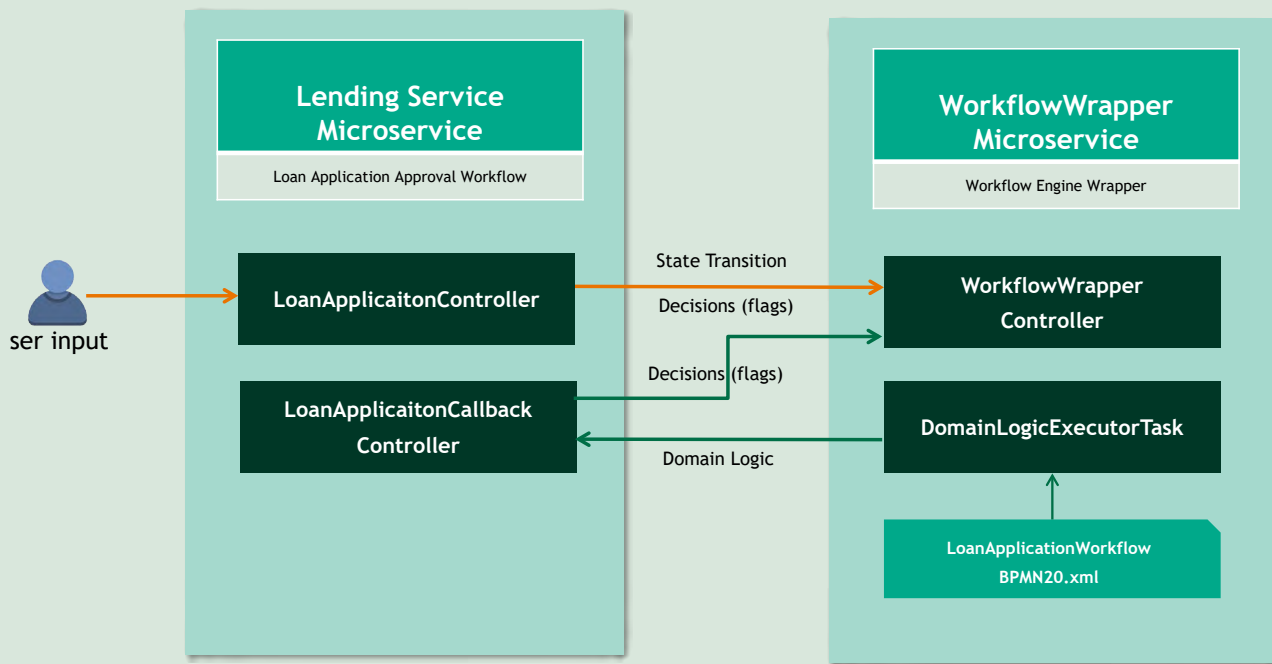
Given there are multiple options of workflow solutions in the opensource space with varying levels of features, community support and adoption levels across enterprises, we felt the need to encapsulate the workflow engine framework as a microservice and created a wrapper with methods that can be used by the business microservices. This helps to ensure that the development teams are more focused around building their business workflow microservice and delegate to the wrapper microservice for the actual state management / progressing through the workflow stages.

EXAMPLE – LENDING SERVICE

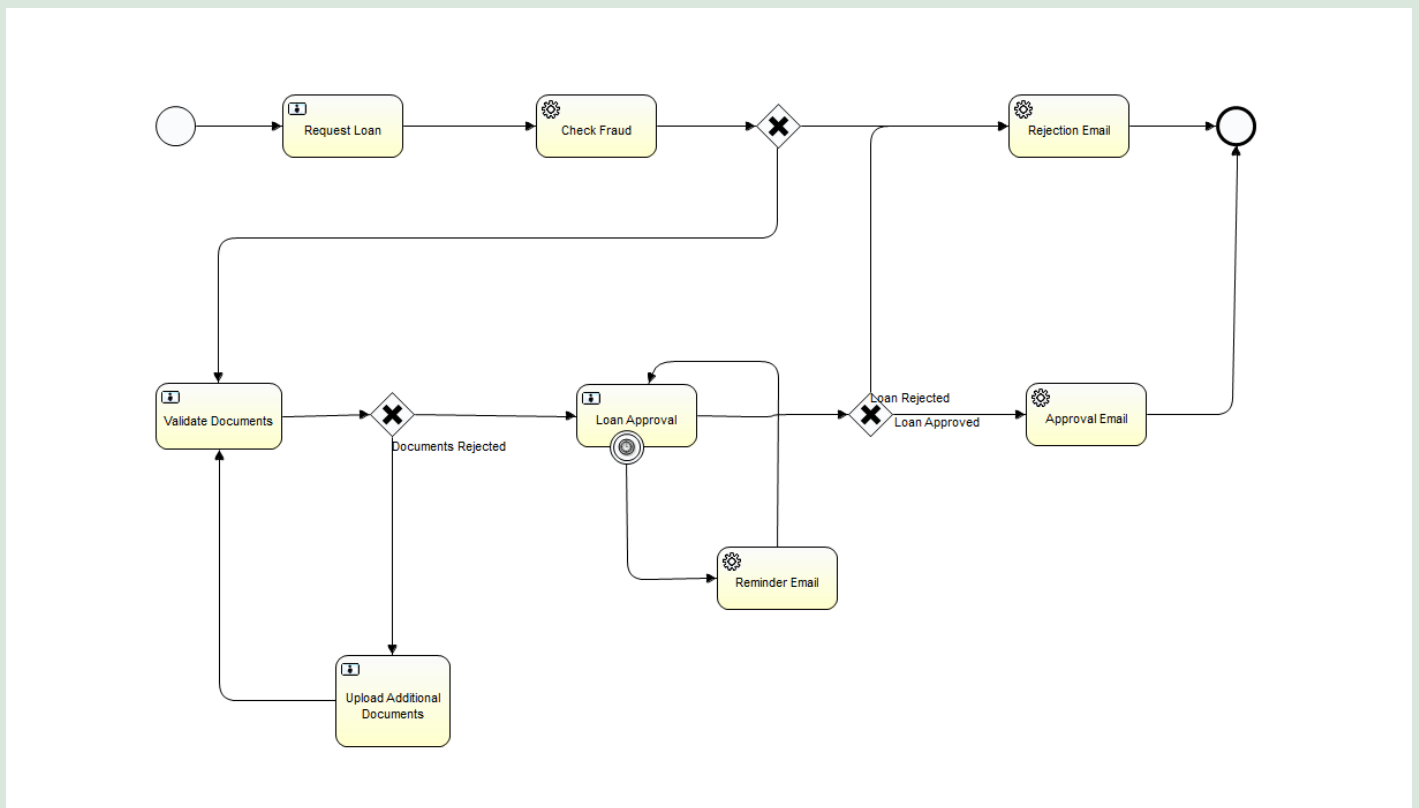
Let's us consider an example of building a lending service that needs to accept the documents from the customer and take it through a workflow to approve or reject the loan.

The following infographic depicts how a domain-specific microservice is implemented using Flowable Workflow Engine by keeping the Flowable specific implementation inside a wrapper microservice using SpringBoot microservices framework. The following diagram explains how the two microservices interact with each other.





For explanation, we will take example of Loan Application approval process which is depicted using Flowable Editor as shown in below diagram.



Any business flow consists of tasks that can be broadly categorized into two types.

- **User Tasks** (Tasks which require User/Human intervention ex Request Loan, Validate Documents) – follows orange path.
- **Automated Tasks** (Tasks which don't require User/Human Active intervention ex Check Fraud, Reminder Email etc.) – follows green path.

Let us consider **Validate Documents** step for first category. Whenever the loan application reaches that stage user/bank employee will review the documents and will either approve or reject. To do the same Lending Microservice API will be called as given below in the code snippet.

```

LoanApplicationController.java
1 package com.abc.lending.controller;
2
3 import org.json.simple.JSONObject;
4
5 @RestController
6 public class LoanApplicationController {
7     @Autowired
8     LendingService lendingService;
9
10    //Every User Task will have an API associated
11
12    public ResponseEntity<?> startLoanApplication(@RequestBody JSONObject data
13
14    public ResponseEntity<?> submitLoanApplication(@RequestBody JSONObject data
15
16    @PostMapping("/validateDocuments")
17    public ResponseEntity<?> validateDocuments(@RequestBody JSONObject data,
18        @RequestHeader("Authorization") String authToken) {
19
20        lendingService.validateDocuments(data);
21        return new ResponseEntity<String>("Application Submitted Successfully", HttpStatus.OK);
22    }
23
24    public ResponseEntity<?> approveLoan(@RequestBody JSONObject data,
25
26 }

```

Inside the validateDocuments(), after performing business logic, audit and database calls, state transition will be done by making REST api call to wrapper microservice as shown below

In the code snippet below, there are 2 REST api calls (orange arrow) from LoanApplicationController to Workflow wrapper Controller, one to set the variable/decision flag and the other to set the transition (next state).

```

@SuppressWarnings("unchecked")
public void validateDocuments(JSONObject data) {
    // call 1 - set decision parameters
    // call 2 - complete the user Task

    String processId = data.get("loanNumber").toString();

    JSONObject request = new JSONObject();
    request.put("decision", false);
    String url = "http://wrapperMicroservice:8080/setVariables/" + processId;
    Boolean result = restTemplate.postForObject(url, request, Boolean.class);

    if (result) {
        // save comments in database and then complete the task

        String url2 = "http://wrapperMicroservice:8080/complete/" + processId;
        restTemplate.getForEntity(url2, String.class);
    }
}

```

For the second category (Automated Tasks), let us consider the example of Check Fraud, Whenever the loan Application reaches Check Fraud Stage an event will be triggered inside the Wrapper Microservice

Inside the XML file, every such task is linked with a class, or a delegate expression as shown below, in this case Check Fraud Task is linked with "checkFraud" delegate Expression.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance">
  <process id="Loan" name="Loan" isExecutable="true">
    <documentation>Loan Process Model</documentation>
    <startEvent id="startevent1" name="Start"></startEvent>
    <userTask id="usertask1" name="Request Loan" xmlns:flowable="http://flowable.org/bpmn" flowable:formFieldValidation="true"></userTask>
    <sequenceFlow id="flow1" sourceRef="startevent1" targetRef="usertask1"></sequenceFlow>
    <serviceTask id="servicetask1" name="Check Fraud" flowable:delegateExpression="${checkFraud}"></serviceTask>
  </process>
</definitions>
```

After this we create a bean for the same and create a class for the same, Here FraudCheckCallBack Class is created

```
package com.abc.common;

import org.springframework.context.annotation.Bean;

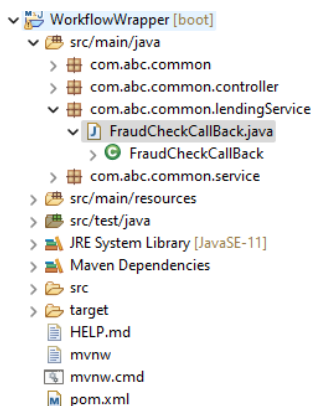
@Configuration
public class ApplicationConfiguration {

    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }

    @Bean
    public FraudCheckCallBack fraudCheck() {
        return new FraudCheckCallBack();
    }
}
```

Now when the loan Application Reaches Check Fraud Stage, FraudCheckCallBack execute method is called on its own (depicted by arrow from xml to DomainLogicExecutor Task)

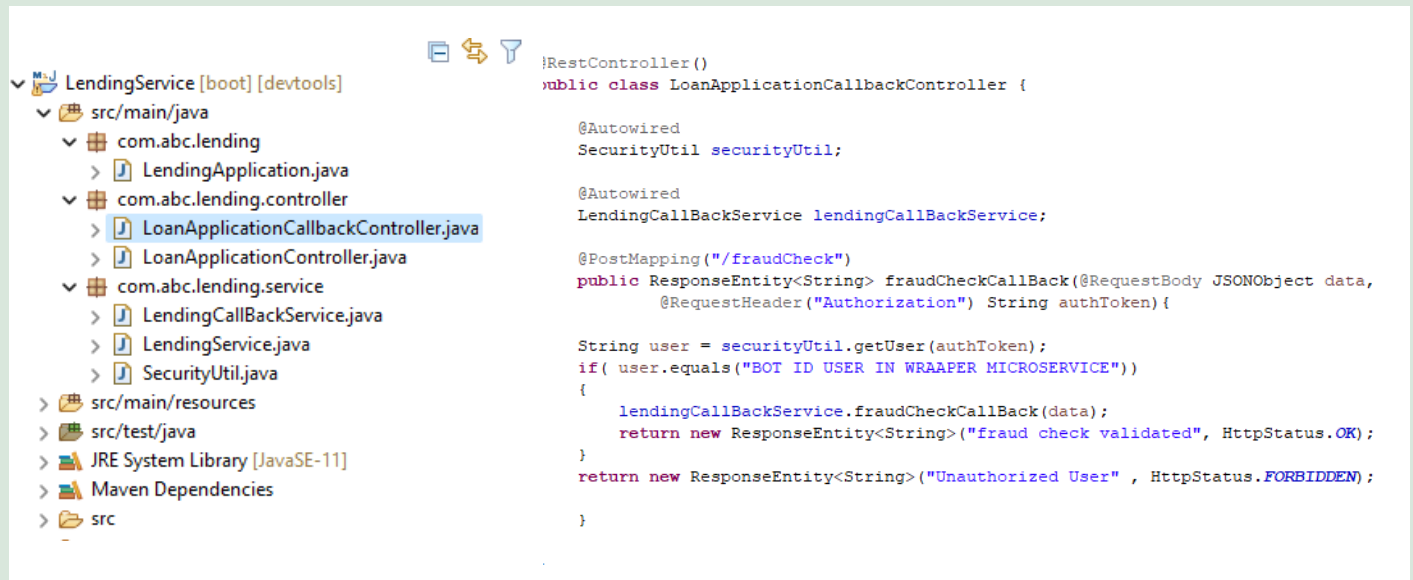
Inside the execute method, REST Call to Lending Service is made as shown in below code. Below REST call denotes arrow from DomainLogicExecutorTask to LoanApplicationCallBack Controller.



```
import org.flowable.engine.delegate.DelegateExecution;
8
9 public class FraudCheckCallBack implements JavaDelegate{
10
11     @Autowired
12     RestTemplate restTemplate;
13
14     @SuppressWarnings("unchecked")
15     @Override
16     public void execute(DelegateExecution execution) {
17
18         //every loan case is associated with a process id ,
19         //mapping stored in a table at start of loan application.
20
21         String processId = execution.getRootProcessInstanceId();
22         JSONObject data = new JSONObject();
23         data.put("processId", processId);
24         String url = "http://loanMicroservice:8080/fraudCheck";
25         restTemplate.postForObject(url, processId , String.class);
26
27     }
28
29 }
```

Lending Microservice has two controllers to distinguish the source of API calls. The reason for the segregation is to have clear separation of responsibility as follows.

- 1) LoanApplicationController (user actions invoke API of this controller)
- 2) LoanApplicationCallbackController (Wrapper Microservice invokes the domain microservice to perform any system actions (automated) which is available only in the domain microservice – ex: fraudCheck.



```
!RestController()
public class LoanApplicationCallbackController {

    @Autowired
    SecurityUtil securityUtil;

    @Autowired
    LendingCallBackService lendingCallBackService;

    @PostMapping("/fraudCheck")
    public ResponseEntity<String> fraudCheckCallBack(@RequestBody JSONObject data,
        @RequestHeader("Authorization") String authToken){

        String user = securityUtil.getUser(authToken);
        if( user.equals("BOT ID USER IN WRAPPER MICROSERVICE"))
        {
            lendingCallBackService.fraudCheckCallBack(data);
            return new ResponseEntity<String>("fraud check validated", HttpStatus.OK);
        }
        return new ResponseEntity<String>("Unauthorized User" , HttpStatus.FORBIDDEN);
    }
}
```

Now inside the fraudCheckCallBack method, business logic is written – For example, a call to third party API or decision based on data to determine if there are any fraud situation with the application. After that same decision is informed to Wrapper Microservice via a REST call by sending the decision flag/variable (denoted by arrow from LoanApplicationCallBackController to WorkflowWrapperController)

```
package com.abc.lending.service;

import org.json.simple.JSONObject;

@Service
public class LendingCallBackService {

    @Autowired
    RestTemplate restTemplate;

    @SuppressWarnings("unchecked")
    public void fraudCheckCallBack(JSONObject data) {
        // call to a third party API with necessary parameters
        // After making call based on its result , we set decision variables
        String processId = data.get("loanNumber").toString();

        JSONObject request = new JSONObject();
        request.put("fraud", false);
        String url = "http://wrapperMicroservice:8080/setVariables/" + processId;
        restTemplate.postForObject(url, request, String.class);
    }
}
```

Code snippets to explain how the Wrapper Microservice controller works

For every basic functionality like creating a case, completion of user tasks an end point is written in Wrapper Microservice (Four Functionalities needed for transition from start to end are explained below)

Deploying workflow/Business Process Model.

We can deploy not only one but multiple process models in wrapper microservice, to do so we need to add the xml file under the resources folder and call the below API to deploy it to flowable as shown below. RuntimeService , TaskService and RepositoryService are flowable supplied classes.

```
@GetMapping("/deploy/{fileName}")
public String deploy(@PathVariable String fileName)
{
    return workflowService.deployProcessDefinition(fileName);
}

@Service
public class WorkflowWrapperService {

    @Autowired
    RuntimeService runtimeService;

    @Autowired
    TaskService taskService;

    @Autowired
    RepositoryService repositoryService;

    //fileName is same as name of bpmn file
    public String deployProcessDefinition(String fileName) {

        repositoryService.createDeployment().addClasspathResource(fileName).deploy();
        return "process is deployed\n\n";

    }
}
```

Creating an Instance/Case of the Process Model

To create a case, we need to know the key to the process model defined in the XML

```
@GetMapping("/create/{key}")
public ResponseEntity<String> create(@PathVariable String key)
{
    String response = workflowService.create(key);
    return new ResponseEntity<String>(response , HttpStatus.OK);
}

//key represents the key given in bpmn file
public String create(String key)
{
    ProcessInstance processInstance = runtimeService.startProcessInstanceByKey(key);
    String processId = processInstance.getId();
    // for every loan application a unique id is created to track it.
    return processId;
}
```

Setting Decision Flags

Inside the xml, for every decision path a variable is set, later during the flow we set its value either to true or false, in case of multiple paths condition like $x=1$, $x=2$ can be used

```
@PostMapping("/setVariables/{processId}")
public ResponseEntity<Boolean> setVariables(@PathVariable String processId ,
    @RequestBody JSONObject data)
{
    boolean response = workflowService.setVariables(data , processId);
    return new ResponseEntity<Boolean>(response , HttpStatus.OK);
}

public boolean setVariables(JSONObject data , String processId)
{
    try {

        Task task = taskService.createTaskQuery().processInstanceId(processId).list().get(0);
        String taskId = task.getId();
        for(Object key : data.keySet())
        {
            taskService.setVariable(taskId, key.toString(), data.get(key));
        }

        return true;
    }
    catch(Exception e)
    {
        return false;
    }
}
```





Completion of User Task

When a user task like validate Documents is completed, same is informed to flowable via REST call

```
@GetMapping("/complete/{processId}")
public ResponseEntity<String> completeTask(@PathVariable String processId)
{
    String response = workflowService.completeTask(processId);
    return new ResponseEntity<String>(response , HttpStatus.OK);
}

public String completeTask(String processId)
{
    Task task = taskService.createTaskQuery().processInstanceId(processId).list().get(0);
    String taskid = task.getId();
    taskService.complete(taskid);
    return "User Task Ends";
}
```


Main

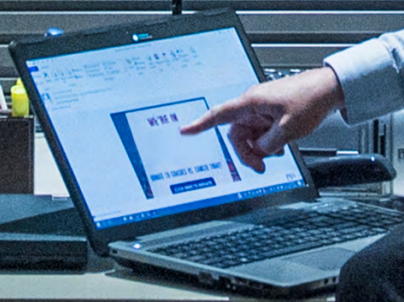
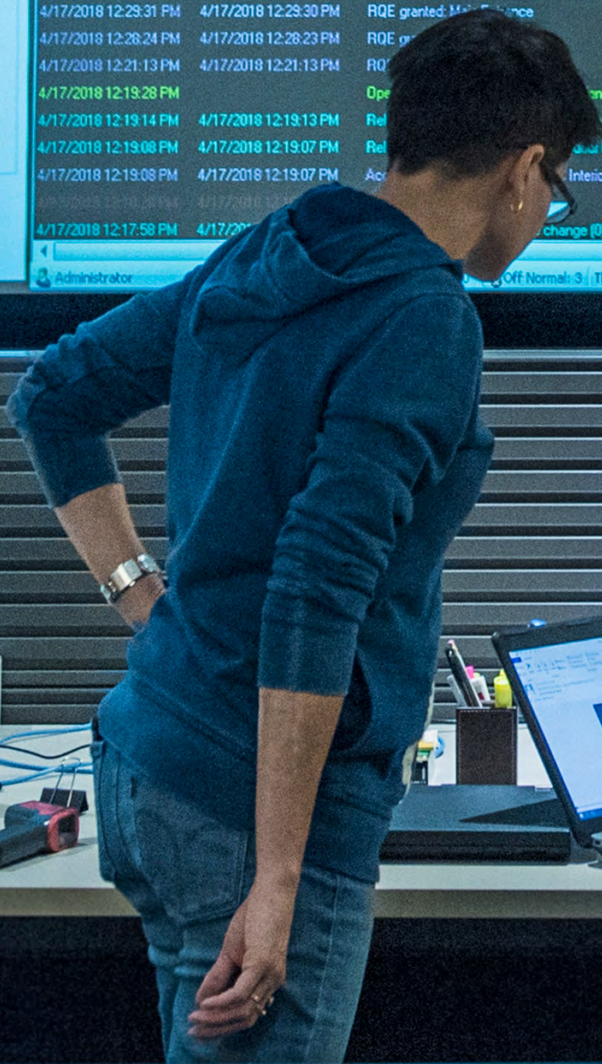
Administration Customization Manager
 Alarm Viewer Event Viewer Graphics Status Viewer Who's Inside Report Manager Enrollment Manager Badge Designer NVR / DVR Viewer CCTV Viewer

Configuration System Activity

4/17/2018 12:31:20 PM	4/17/2018 12:31:20 PM	Relay NUL Interior state change (1) door trigger	W\NET.001.0002.001.02.BR06	DIGI*TRAC In
4/17/2018 12:31:20 PM	4/17/2018 12:31:20 PM	Access granted: Siamak Badiee NOC Interior	W\NET.001.0002.001.02.SM06	DIGI*TRAC Tra
4/17/2018 12:30:16 PM	4/17/2018 12:30:15 PM	Relay Main Entrance state change (0) none	W\NET.001.0002.001.01.BR01	DIGI*TRAC In
4/17/2018 12:30:15 PM	4/17/2018 12:30:14 PM	Expansion input Main Entry cutoff secure	W\NET.001.0002.001.01.XR01	DIGI*TRAC Ext
4/17/2018 12:30:15 PM	4/17/2018 12:30:15 PM	Expansion relay Main Entry Cutoff state change (0) none	W\NET.001.0002.001.02.XR01	DIGI*TRAC In
4/17/2018 12:30:14 PM	4/17/2018 12:30:14 PM	Input Mantrap secure	W\NET.001.0002.001.02.BI01	DIGI*TRAC Ext
4/17/2018 12:30:09 PM	4/17/2018 12:30:08 PM	Relay Main Entrance state change (0) control zone disabled	W\NET.001.0002.001.01.BR01	DIGI*TRAC In
4/17/2018 12:30:09 PM	4/17/2018 12:30:08 PM	Alarm at expansion input Main Entry cutoff	W\NET.001.0002.001.01.XR01	DIGI*TRAC In
4/17/2018 12:30:09 PM	4/17/2018 12:30:08 PM	Expansion relay Main Entry Cutoff state change (1) control zone actuated	W\NET.001.0002.001.02.XR01	DIGI*TRAC In
4/17/2018 12:30:08 PM	4/17/2018 12:30:08 PM	Forced entry at input Mantrap	W\NET.001.0002.001.02.BI01	DIGI*TRAC In
4/17/2018 12:29:38 PM	4/17/2018 12:29:38 PM	Relay Mantrap state change (0) none	W\NET.001.0002.001.02.BR01	DIGI*TRAC In
4/17/2018 12:29:38 PM	4/17/2018 12:29:37 PM	Input Electric Room secure	W\NET.001.0002.001.02.BI07	DIGI*TRAC Ext
4/17/2018 12:29:38 PM	4/17/2018 12:29:36 PM	Expansion relay XRelay 01 state change (0) none	W\NET.001.0002.001.01.XR01	DIGI*TRAC In
4/17/2018 12:29:37 PM	4/17/2018 12:29:36 PM	Relay Main Entrance state change (0) none	W\NET.001.0002.001.01.BR01	DIGI*TRAC In
4/17/2018 12:29:34 PM	4/17/2018 12:29:34 PM	Relay Mantrap state change (0) control zone disabled	W\NET.001.0002.001.02.BR01	DIGI*TRAC In
4/17/2018 12:29:34 PM	4/17/2018 12:29:34 PM	Forced entry at input Electric Room	W\NET.001.0002.001.02.BI07	DIGI*TRAC In
4/17/2018 12:29:34 PM	4/17/2018 12:29:33 PM	Expansion relay XRelay 01 state change (1) control zone actuated	W\NET.001.0002.001.01.XR01	DIGI*TRAC In
4/17/2018 12:29:32 PM	4/17/2018 12:29:30 PM	Relay Main Entrance state change (1) door trigger	W\NET.001.0002.001.01.BR01	DIGI*TRAC In
4/17/2018 12:29:31 PM	4/17/2018 12:29:30 PM	RQE granted: Main Entrance	W\NET.001.0002.001.01.SM01	DIGI*TRAC Tra
4/17/2018 12:26:24 PM	4/17/2018 12:26:23 PM	RQE granted: Main Entrance	W\NET.001.0002.001.01.SM02	DIGI*TRAC Tra
4/17/2018 12:21:13 PM	4/17/2018 12:21:13 PM	RQE granted: Main Entrance	W\NET.001.0002.001.01.SM04	DIGI*TRAC Tra
4/17/2018 12:19:28 PM	4/17/2018 12:19:28 PM	Open door at SMSVIEW-TBROWN	W\NET.001.0002.001.01.SM04	DIGI*TRAC Tra
4/17/2018 12:19:14 PM	4/17/2018 12:19:13 PM	Relay Main Entrance state change (0) none	W\NET.001.0002.001.01.BR01	DIGI*TRAC In
4/17/2018 12:19:08 PM	4/17/2018 12:19:07 PM	Relay Main Entrance state change (0) door trigger	W\NET.001.0002.001.01.BR01	DIGI*TRAC In
4/17/2018 12:19:08 PM	4/17/2018 12:19:07 PM	Access granted: Siamak Badiee NOC Interior	W\NET.001.0002.001.01.BR01	DIGI*TRAC Tra
4/17/2018 12:17:58 PM	4/17/2018 12:17:58 PM	Relay Main Entrance state change (0) none	W\NET.001.0002.001.01.BR01	DIGI*TRAC In

Administrator | Off Normal: 3 | Threat Level: Level 00

Data Center Floor Plan

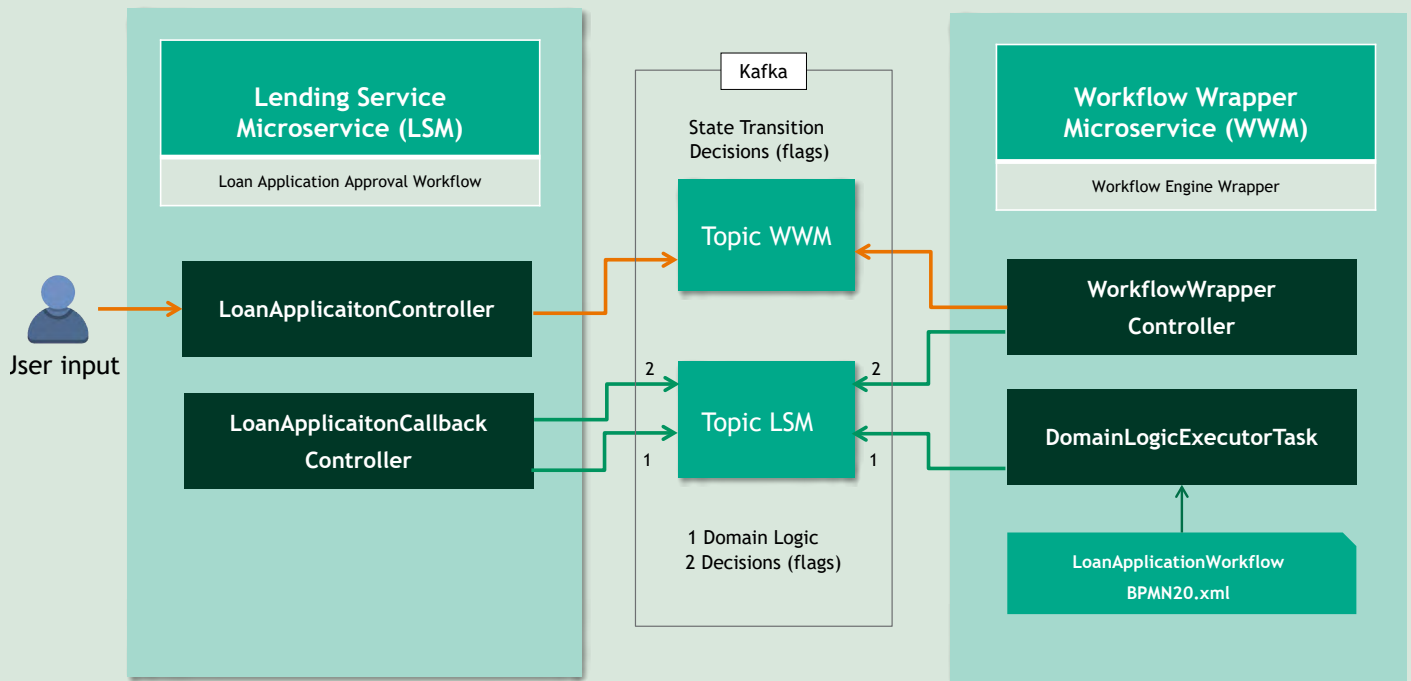


SCALABILITY OF WORKFLOWS

As we are following microservices architecture, each workflow microservice will be deployed separately in its own container. So, depending on metrics like CPU utilization and memory a particular domain microservice can be scaled up and down.

As Wrapper Microservice remains same for all the workflows and manages their state, it will also have to be auto scaled based on the metrics.

Another variation of the above design could be to decouple the wrapper microservice using an event based mechanism from the domain-microservices. Here the domain microservices would publish an event to a Kafka topic which would be consumed by the Wrapper Microservice to progress the workflow stages. Similarly the call back to the domain microservice would be another event from the wrapper microservice published to another topic to be consumed by the domain microservice for executing business rules/validation as shown below in the diagram. While event-driven microservices architecture provides a great level of decoupling and scaling, the architecture becomes complex and is warranted only if there are hundreds of domain microservices having to invoke the wrapper microservices back and forth.



BUSINESS AND STATE DATA SEGREGATION

Wrapper Microservice uses tables generated by the workflow engine (Flowable) and other workflow specific microservices use their own table so business logic (domain specific data) and transition/state data logic can be kept separate.

Also based on the requirement, the business data can be kept in a different database as there is no inter-relation between wrapper service data and workflow specific one.

CONCLUSION

Building business process execution workflows using open-source workflow engines as a micro-services-based architecture helps the development teams to author lightweight BPMN2.0 compliant models and wrap them with a domain microservice construct. This approach provides the above-mentioned benefits of scaling the individual domain microservices while decoupling from the underlying BPM product is the critical objective demonstrated in this paper given the rate of technology obsolescence and speed of innovation in the open source space.

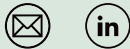
About the Author/Mentor



Author
Subramanian Radhakrishnan
Senior Principal Technology Architect



Co-Author
Anurag Sekhri
Specialist Programmer



Reviewer
Sachin Girijashankar Agrawal
Senior Principal Technology Architect



For more information, contact askus@infosys.com



© 2023 Infosys Limited, Bengaluru, India. All Rights Reserved. Infosys believes the information in this document is accurate as of its publication date; such information is subject to change without notice. Infosys acknowledges the proprietary rights of other companies to the trademarks, product names and such other intellectual property rights mentioned in this document. Except as expressly permitted, neither this documentation nor any part of it may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, printing, photocopying, recording or otherwise, without the prior permission of Infosys Limited and/ or any named intellectual property rights holders under this document.