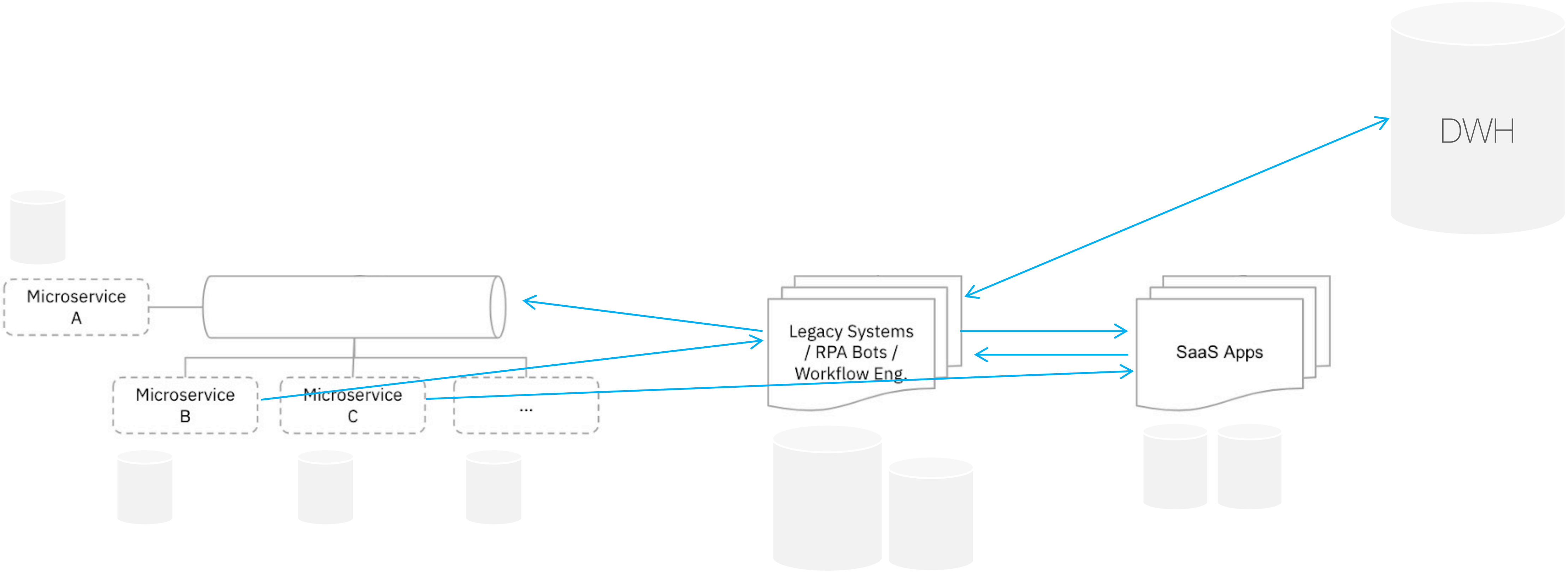
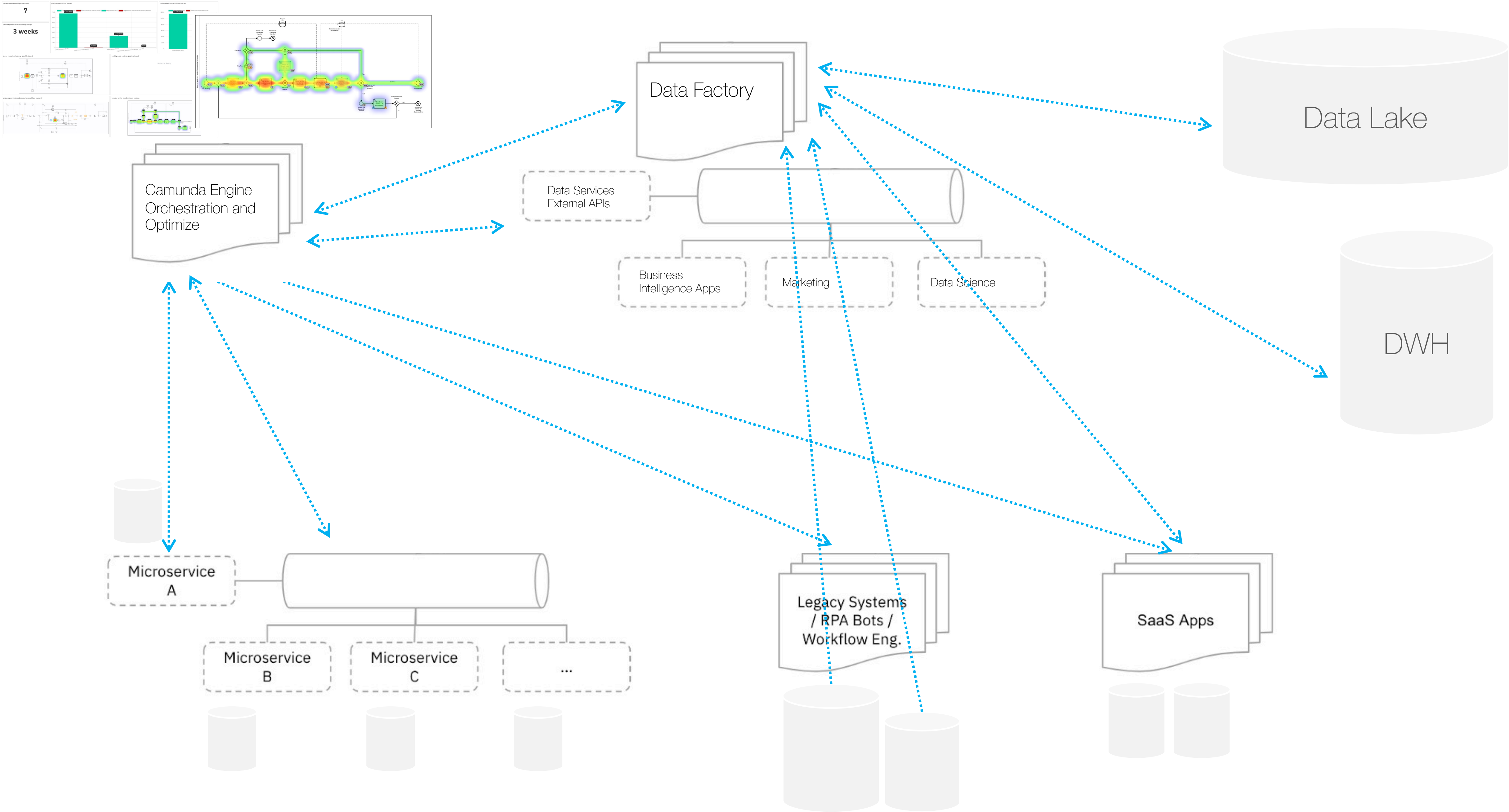
Controlled Transformation towards a **Command-Driven / Event-Driven Microservice framework**

- ✓ Avoiding Pinball-machines, synchronization problems, …
- ✓ Improve the understanding of event architecture
- ✓ Simplifying message/event/data security
- ✓ Improve change management
- ✓ Simplifying recovery from failures
- ✓ Defining boundaries (responsibility/accountability)
- ✓ Avoid distributed monoliths


DWH
Microservice A
Microservice B
Microservice C
...
Legacy Systems / RPA Bots / Workflow Eng.
SaaS Apps



Data Factory

Data Lake

Camunda Engine Orchestration and Optimize

Data Services External APIs

Business Intelligence Apps

Marketing

Data Science

DWH

Microservice A

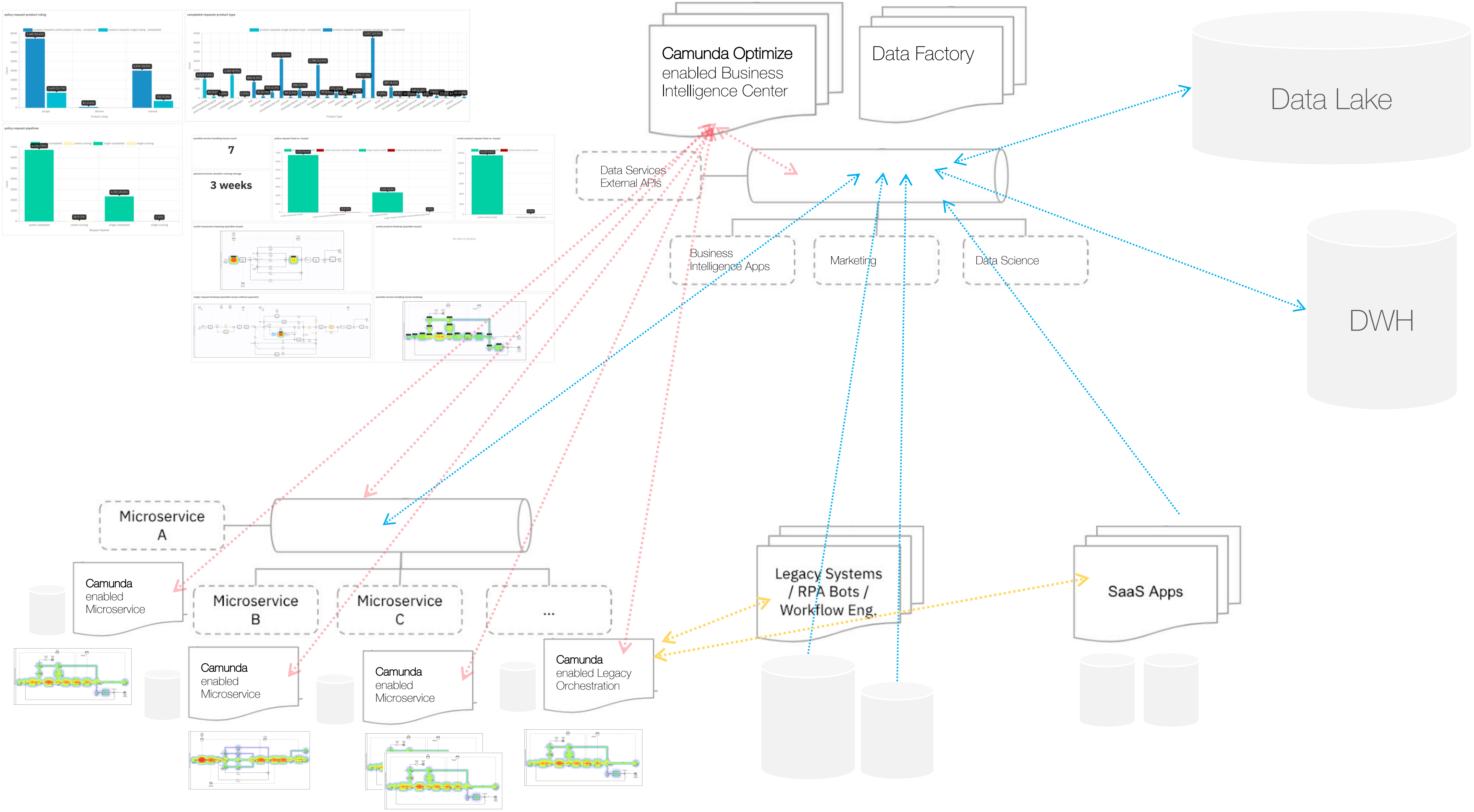
Microservice B

Microservice C

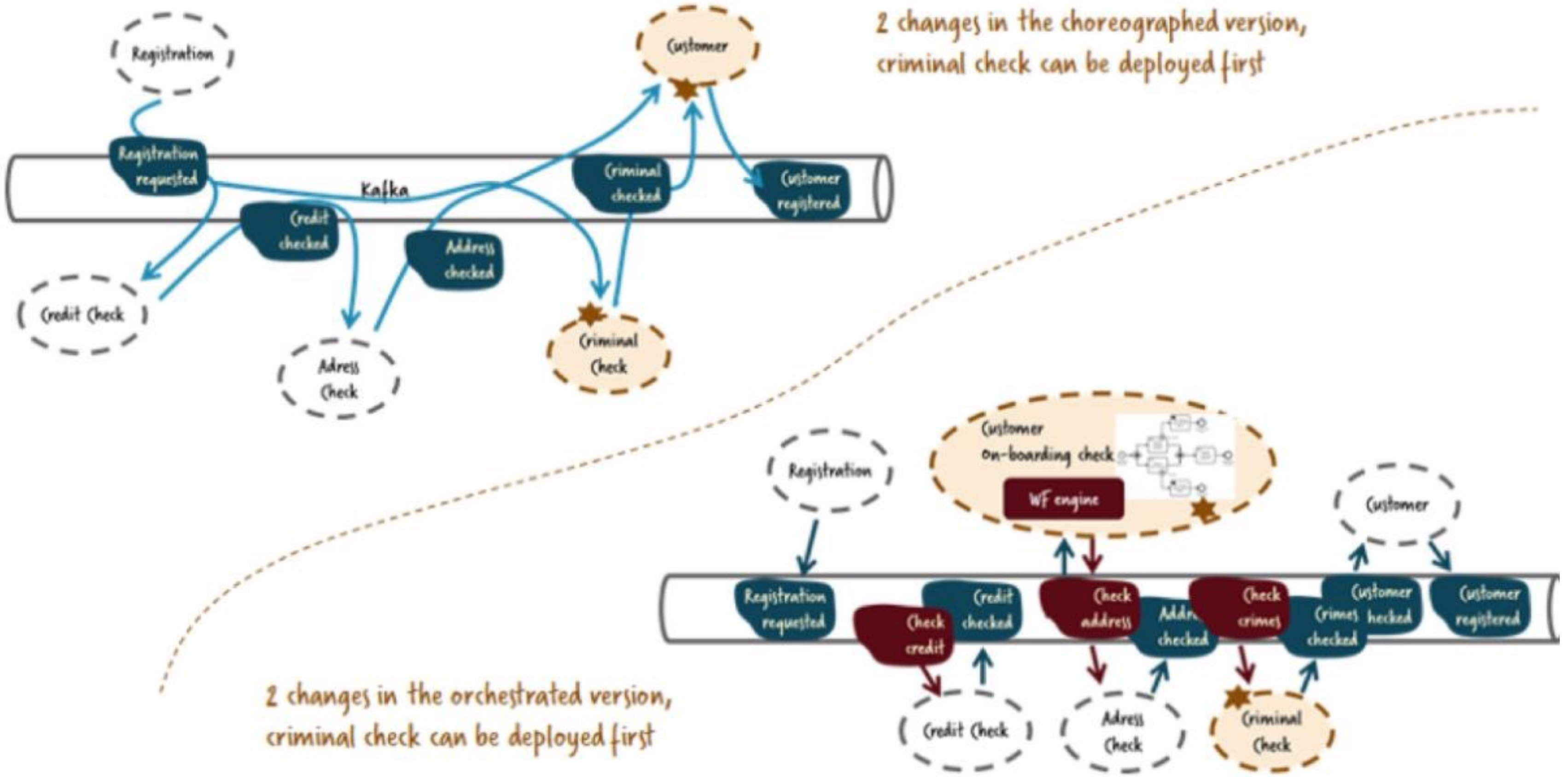
...

Legacy Systems / RPA Bots / Workflow Eng.

SaaS Apps

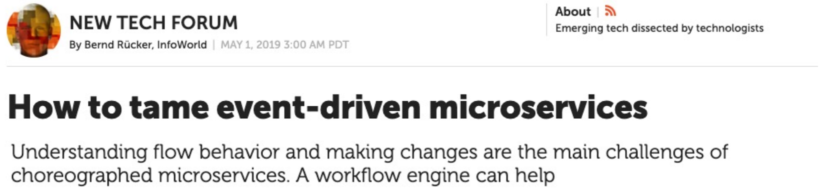

Camunda Optimize
enabled Business
Intelligence Center
Data Factory
Data Lake
Data Services
External APIs
Business
Intelligence Apps
Marketing
Data Science
DWH
7
3 weeks
Microservice
A
Camunda
enabled
Microservice
Microservice
B
Microservice
C
...
Camunda
enabled
Microservice
Camunda
enabled
Microservice
Camunda
enabled Legacy
Orchestration
Legacy Systems
/ RPA Bots /
Workflow Eng.
SaaS Apps

Orchestration within microservices



2 changes in the choreographed version, criminal check can be deployed first

2 changes in the orchestrated version, criminal check can be deployed first



NEW TECH FORUM
By Bernd Rücker, InfoWorld | MAY 1, 2019 3:00 AM PDT

About |
Emerging tech dissected by technologists

How to tame event-driven microservices

Understanding flow behavior and making changes are the main challenges of choreographed microservices. A workflow engine can help

Problems

- ✓ Reuse generic event workers that know nothing about Camunda
- ✓ Include orchestration in each microservice
- ✓ Do not depend on Java (allow cross platform components)

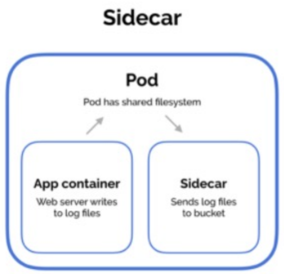
Gains

- ✓ Separating logic of dispatching Camunda workload from workers
- ✓ Single deployment of Camunda orchestration components Kubernetes (side-car)
- ✓ Allows Integration of enhanced security and RBAC without additional changes in Camunda (Java)
- ✓ Allows hyper scalability of workload
- ✓ Allows better control of workload (hence an intelligent service bus is used)

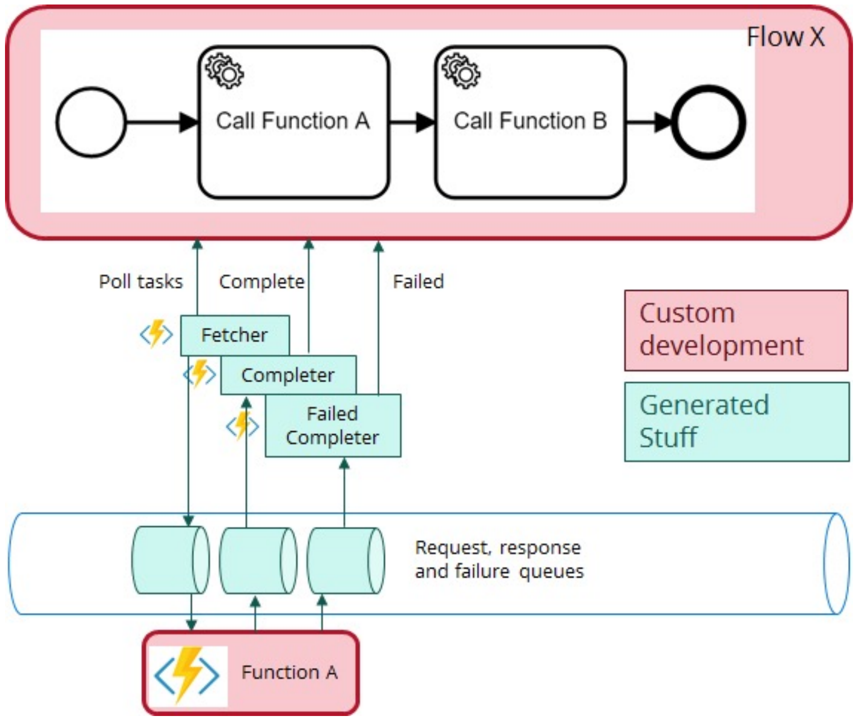
Trade-offs / Considerations

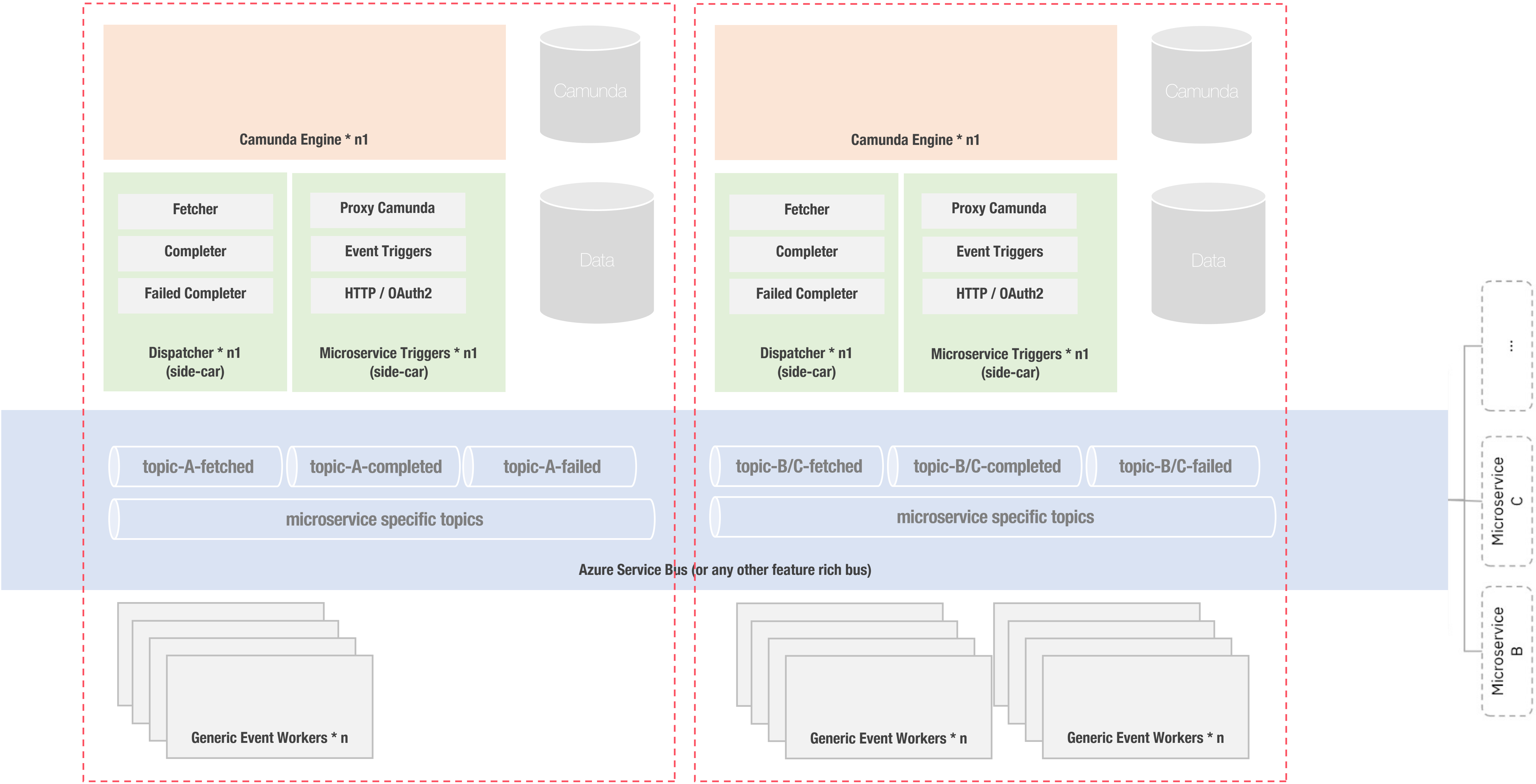
- ✓ Complex infrastructure setup
- ✓ Lock synchronization (message TTL and Camunda task TTL)
- ✓ Message duplication (consider intelligent service bus features to deduplicate)
- ✓ With External Service Architecture process execution is by default asynchronous

kubernetes


Sidecar
Pod
Pod has shared filesystem
App container
Web server writes to log files
Sidecar
Sends log files to bucket

HashiCorp Terraform





Camunda Engine * n1

Camunda

Fetcher

Completer

Failed Completer

Dispatcher * n1
(side-car)

Proxy Camunda

Event Triggers

HTTP / OAuth2

Microservice Triggers * n1
(side-car)

Data

Camunda Engine * n1

Camunda

Fetcher

Completer

Failed Completer

Dispatcher * n1
(side-car)

Proxy Camunda

Event Triggers

HTTP / OAuth2

Microservice Triggers * n1
(side-car)

Data

topic-A-fetched

topic-A-completed

topic-A-failed

microservice specific topics

topic-B/C-fetched

topic-B/C-completed

topic-B/C-failed

microservice specific topics

Azure Service Bus (or any other feature rich bus)

Generic Event Workers * n

Generic Event Workers * n

Generic Event Workers * n

...

Microservice C

Microservice B

Problems

- ✓ Create dynamically scalable microservices with integrated orchestration
- ✓ Include orchestration in each microservice
- ✓ Enable business to control the logic of microservice
- ✓ Do not depend on Java (allow cross platform components)

Gains

- ✓ Allows scalability of each component separately within the bounded context of the Microservice
- ✓ Plugin system to define executable workflows and business logic by business
- ✓ Allows Integration of enhanced security and RBAC without additional changes in Camunda (Java)
- ✓ Less complex infrastructure is needed
- ✓ Camunda acts as internal bus and is responsible for managing workload (locking, etc.)
- ✓ More integrated deployment = Deployment using Kubernetes Operators

Trade-offs and considerations

- ✓ Complex infrastructure setup
- ✓ Tide coupling of dependencies (worker / orchestration)
- ✓ Additional logic needed to control the workload (Back-pressure and Back-off policies)
- ✓ Consider a plugin system to keep the microservice component generic and ensure dynamic and flexible work executors.
- ✓ With External Service Architecture process execution is by default asynchronous

KUBERNETES OPERATORS

KUDO

RabbitMQ

LOW-CODE PLUGINS

Business Rule Engines

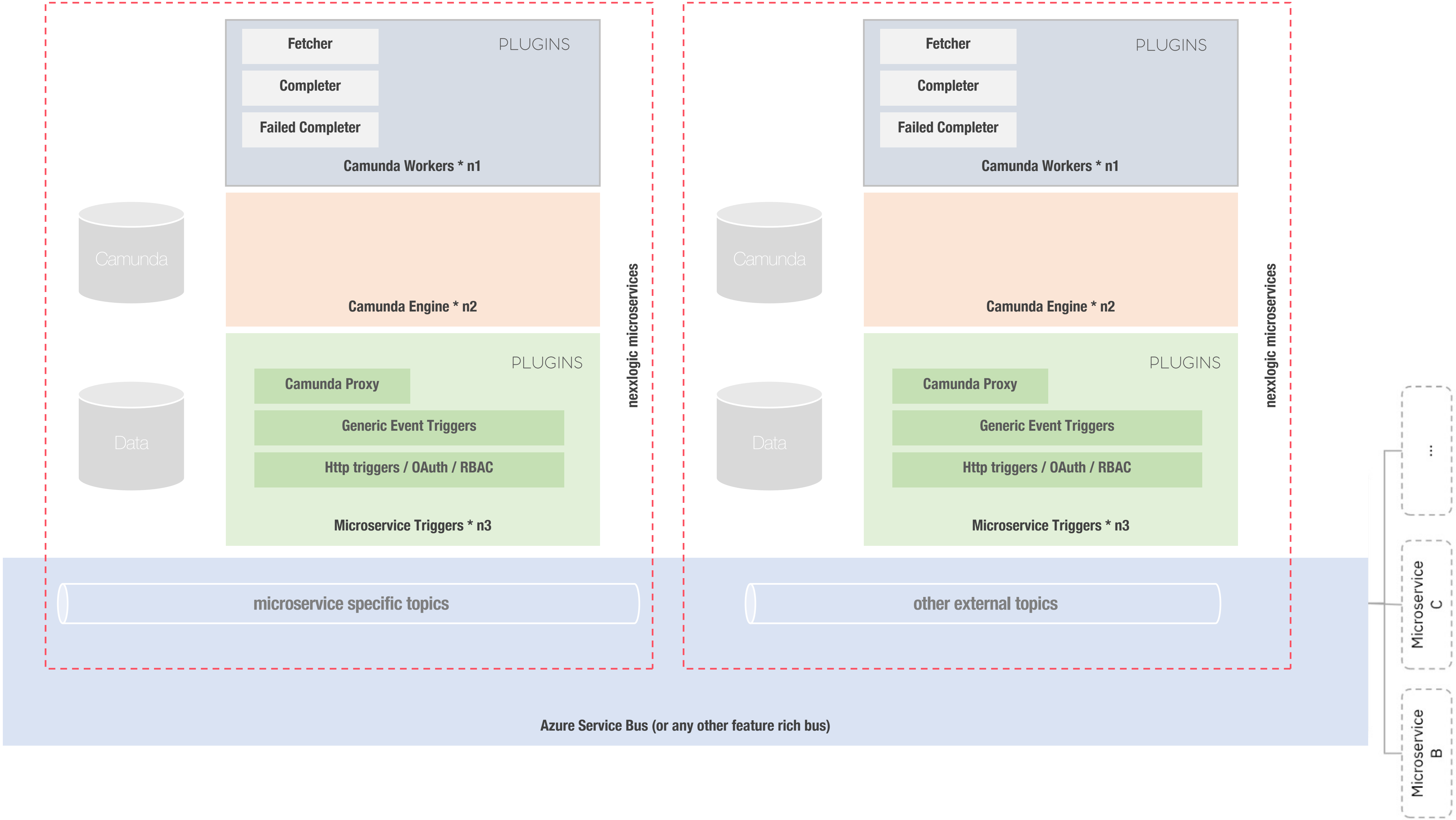
Data Transformers

Code Runners

Http Connectors

Micro Workflows

DMN Executors


Fetcher
Completer
Failed Completer
PLUGINS
Camunda Workers * n1
Camunda
Camunda Engine * n2
PLUGINS
Camunda Proxy
Generic Event Triggers
Http triggers / OAuth / RBAC
Microservice Triggers * n3
Data
nexxlogic microservices
Fetcher
Completer
Failed Completer
PLUGINS
Camunda Workers * n1
Camunda
Camunda Engine * n2
PLUGINS
Camunda Proxy
Generic Event Triggers
Http triggers / OAuth / RBAC
Microservice Triggers * n3
Data
nexxlogic microservices
microservice specific topics
other external topics
Azure Service Bus (or any other feature rich bus)
...
Microservice C
Microservice B

Specific Business Topics for External Service Tasks

Pros

- ✓ Clarity in reporting
- ✓ Allows more specific configuration of topic’s workload in the workers, e.g. variables to be fetched
- ✓ Execution handler changes in the workers can change without a need for new process definition versions.

Cons

- ✓ Large number of topics = more configuration = can cause performance issues
- ✓ Execution handler changes in the workers may have breaking impact in process, models are not tide.

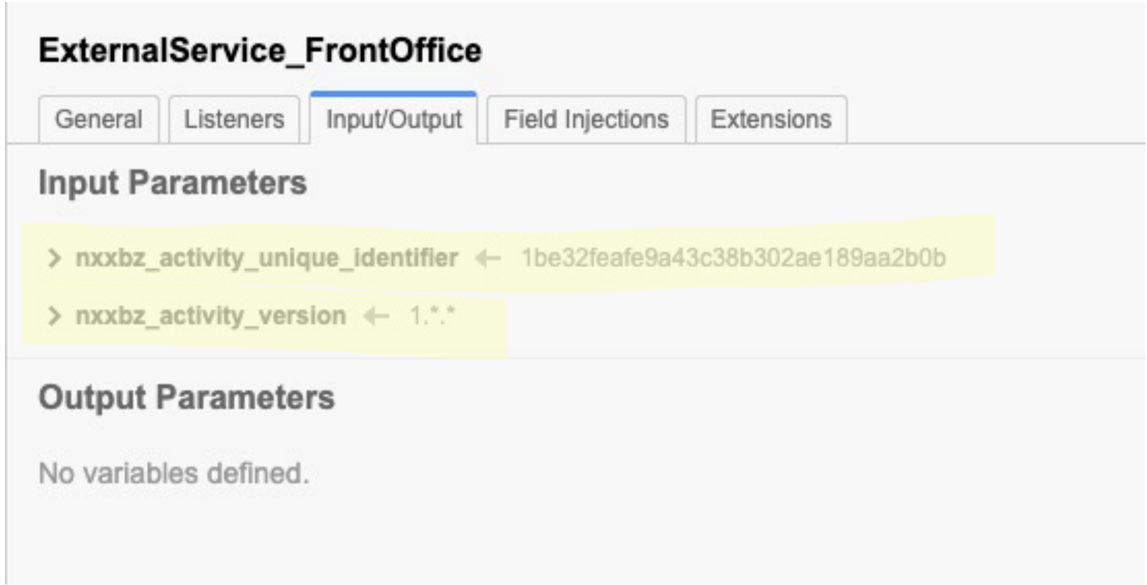
Generic Business Topics for External Service Tasks

Pros

- ✓ Simplifies configuration
- ✓ Allows fetching less topics = lower risk of performance issues in Camunda
- ✓ More control from BPMN model to specify which plugin/execution needs to be done = strict coupling of process definition versions and execution parameters

Cons

- ✓ Generic configuration = less control over which variables we want to fetch (usually fetch all variables and map them during the workload execution)
- ✓ More control from BPMN model to specify which plugin/execution needs to be done = more version of process definitions when execution parameters change



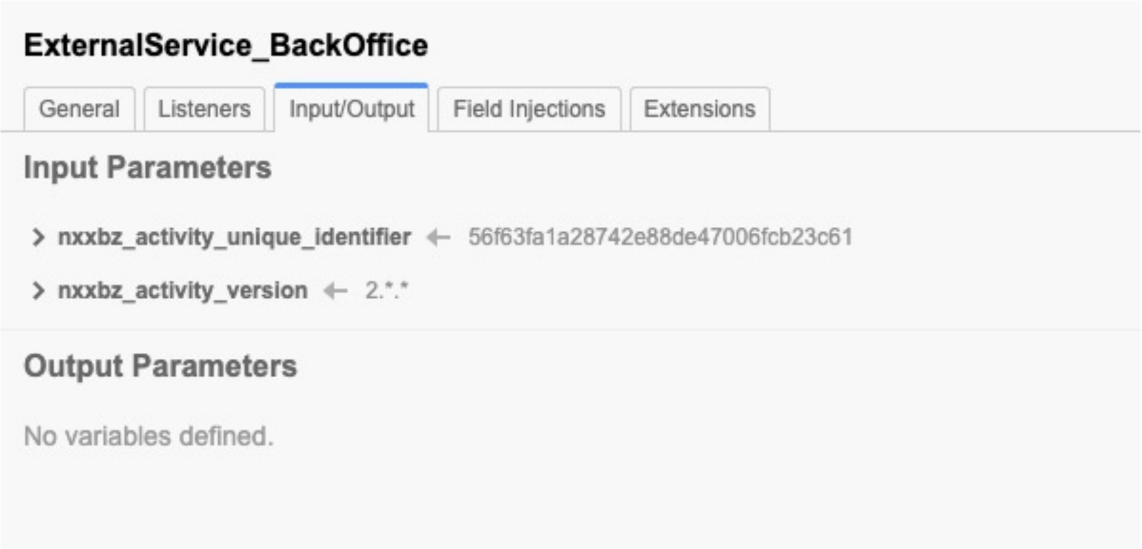
ExternalService_FrontOffice

Input Parameters

- nxxbz_activity_unique_identifier ← 1be32feafe9a43c38b302ae189aa2b0b
- nxxbz_activity_version ← 1.*.*

Output Parameters

No variables defined.



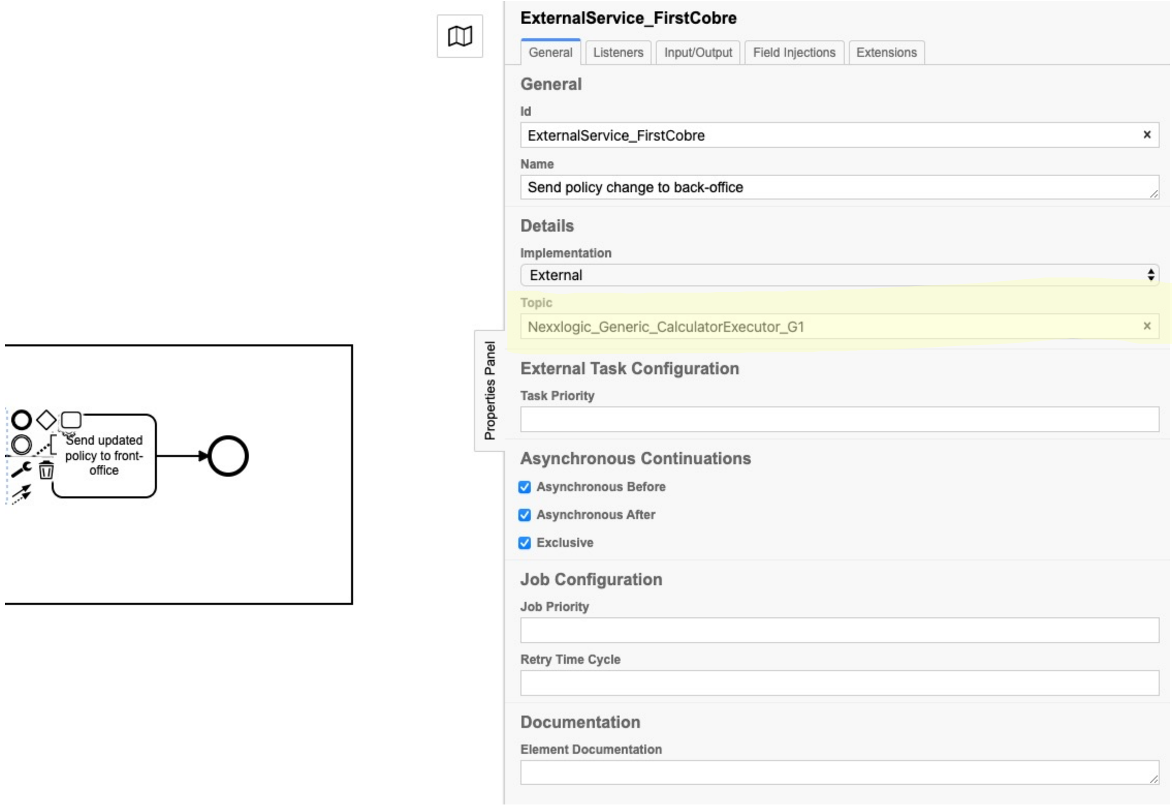
ExternalService_BackOffice

Input Parameters

- nxxbz_activity_unique_identifier ← 56f63fa1a28742e88de47006fcb23c61
- nxxbz_activity_version ← 2.*.*

Output Parameters

No variables defined.



ExternalService_FirstCobre

General

Id: ExternalService_FirstCobre

Name: Send policy change to back-office

Details

Implementation: External

Topic: Nexxlogic_Generic_CalculatorExecutor_G1

External Task Configuration

Task Priority:

Asynchronous Continuations

☑ Asynchronous Before

☑ Asynchronous After

☑ Exclusive

Job Configuration

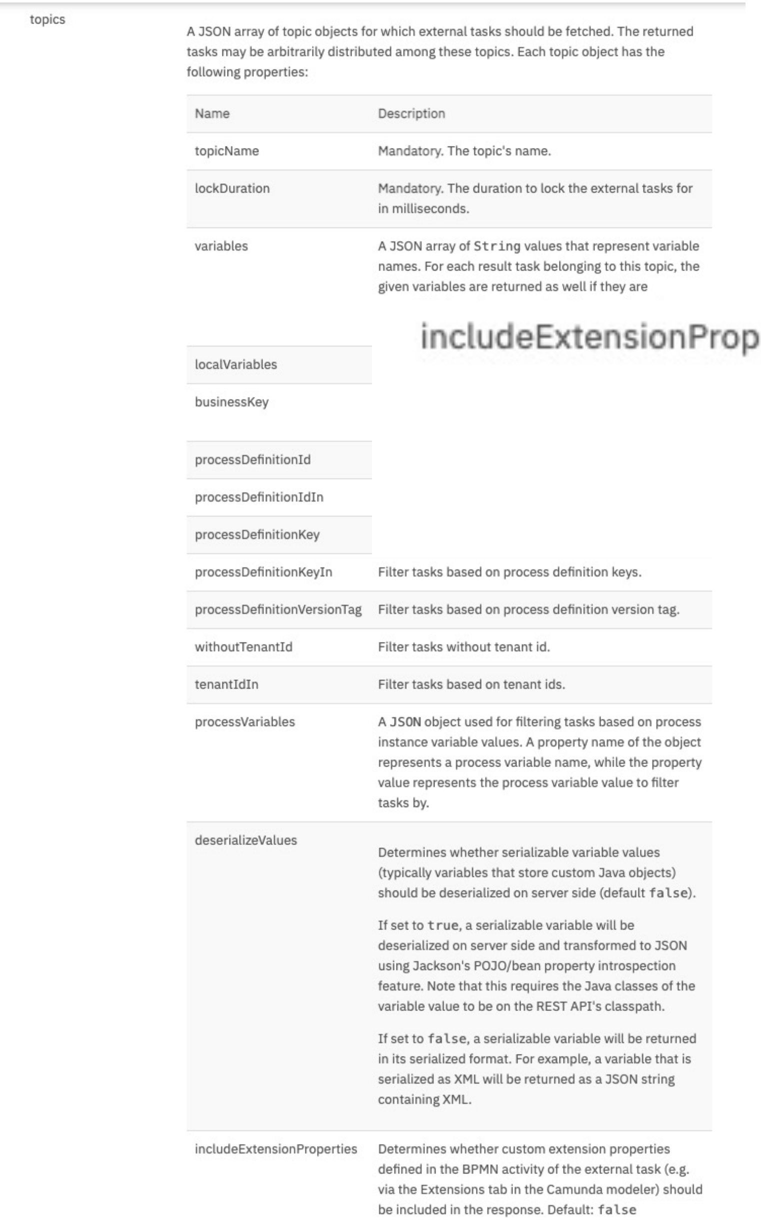
Job Priority:

Retry Time Cycle:

Documentation

Element Documentation:

Camunda 7.15.0

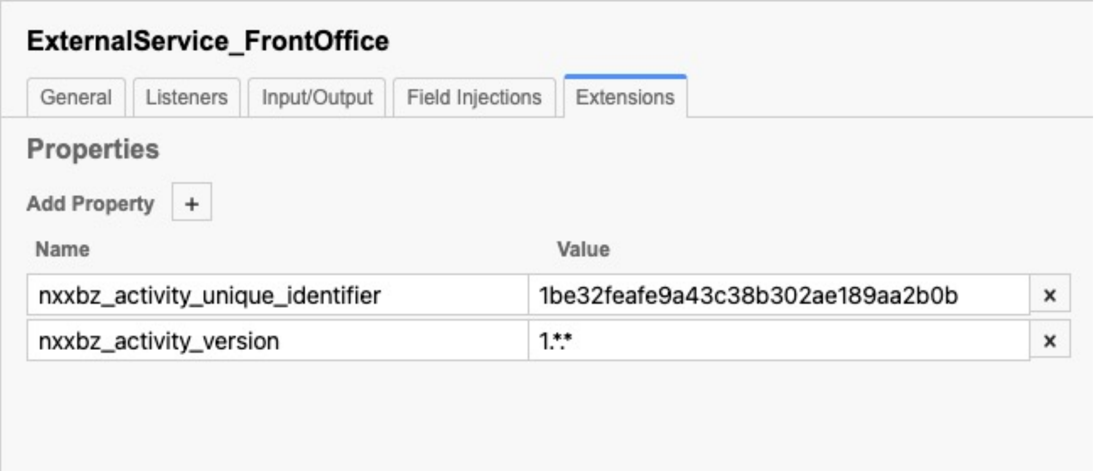


topics

A JSON array of topic objects for which external tasks should be fetched. The returned tasks may be arbitrarily distributed among these topics. Each topic object has the following properties:

Name	Description
topicName	Mandatory. The topic's name.
lockDuration	Mandatory. The duration to lock the external tasks for in milliseconds.
variables	A JSON array of `String` values that represent variable names. For each result task belonging to this topic, the given variables are returned as well if they are
localVariables	
businessKey	
processDefinitionId	
processDefinitionIdIn	
processDefinitionKey	
processDefinitionKeyIn	Filter tasks based on process definition keys.
processDefinitionVersionTag	Filter tasks based on process definition version tag.
withoutTenantId	Filter tasks without tenant id.
tenantIdIn	Filter tasks based on tenant ids.
processVariables	A `JSON` object used for filtering tasks based on process instance variable values. A property name of the object represents a process variable name, while the property value represents the process variable value to filter tasks by.
deserializeValues	Determines whether serializable variable values (typically variables that store custom Java objects) should be deserialized on server side (default `false`). If set to `true`, a serializable variable will be deserialized on server side and transformed to JSON using Jackson's POJO/bean property introspection feature. Note that this requires the Java classes of the variable value to be on the REST API's classpath. If set to `false`, a serializable variable will be returned in its serialized format. For example, a variable that is serialized as XML will be returned as a JSON string containing XML.
includeExtensionProperties	Determines whether custom extension properties defined in the BPMN activity of the external task (e.g. via the Extensions tab in the Camunda modeler) should be included in the response. Default: `false`

includeExtensionProperties Determines whether custom extension properties defined in the BPMN activity of the external task (e.g. via the Extensions tab in the Camunda modeler) should be included in the response. Default: `false`



ExternalService_FrontOffice

Name	Value
nxxbz_activity_unique_identifier	1be32feafe9a43c38b302ae189aa2b0b
nxxbz_activity_version	1.*.*