Sesambook

Morten Frellumstad
Erik Leven
Gabriel Constantin Vig
Geir Atle Hegsvold

Revision History

Revision 01	2021-03-01	GAH
	Initial version.	

Abstract

Lorem ipsum.

Table of Contents

1. Prologue

Introduction to the book

Who is this book for?

Who are we?

What is Sesam?

What problems do we seek to solve?

What should you expect to learn?

Innholdsfortegnelse og forklaring/justification for det.

2. Architecture and Concepts

2.1. Introduction

Sesam opens the door to running a data-driven business, but what is a data driven business and how does the journey towards it look?

In this chapter we will introduce you to Sesam as a way of not only developing but thinking. You will learn about overarching themes for designing integrations, the components you build with and the pitfalls to look out for along the way.

After reading you can expect to recognize and design a data driven Integration of your own.

2.2. Architecture

What do we mean about architecture

1. 1.

There are some integration architectures which have been used historically and today even. One is p2p, where systems create connections on-demand between them, as seen in Figure 1.1.1A: Point 2 Point.

* P2P

* ESB

Figure 1 .1.1A: Point 2 Point

1.

2.

What do we mean about datahub, and why is sesam a datahub

Pros/Cons

1.

2.

Terminology

Systems, pipes, datasets

In order to understand how Sesam works, it is important to understand the parts Sesam is made up of.

There are three central re-occurring concepts in Sesam which you will encounter in your everyday life working with the integration platform: systems, pipes and datasets. These are the fundamental parts which make up a Sesam integration pipeline. Figure 1.1.5A gives you an insight into how a standard Sesam integration pipeline would look.

- Systems:

  A system’s main feature is to import and export data into and out of the Sesam portal. They are therefore found in the beginning and end of the pipeline flows. A system could connect to a REST API, directly to a database of simply send data to a waiting http server. Sesam has several of these system types built into the product to simplify the workings inside the portal. In situations where the built-in system types are not enough for your requirements Sesam also supports connecting systems to a microservice which in turn can manipulate and delegate data according to your own specifications, making Sesam a very robust and comprehensive tool.

- Pipes:

  Pipes handles the transformation of the data and specifies where the data is supposed to be sent. Manipulation of the data is done through Sesam’s own Data Transformation Language (DTL) which allows you to add, remove, transform and combine data according to you own needs. A pipe generally acquires data from a system or from a dataset depending on where the pipe is located inside the integration pipeline.

- Datasets:

  Datasets are Sesam’s storage units. This is where pipes store the data after configuring them and in a pipeline flow, they are generally found between pipes. Sesam stores data in order to be able to perform tracking and indexing, but you will learn more about these functionalities later in this book (maybe a link?).

2.2.1. The Sesam portal

Show basics of portal

(Here also refer to a full chapter for portal or from the projects chapter?)

Integrations, connections and configurations can all be accessed inside the Sesam portal; the user interface of the Sesam product, The Sesam portal can be accessed at portal.sesam.io, and in this section you will learn the most commonly used parts of the portal such that you can orient yourself, as well as manage existing integrations. For a full explanation if the workings and functionality of the Sesam portal, please look [here (with a link)].

When logging in to the portal you will be met with a page like figure 1.1.6A

The cards on the Dashboard are often referred to as “subscriptions” or “nodes” and they represent separate instances of Sesam installations. Each node comes in different sizes (memory available) depending on the requirements of the customer/project/user. In this example you will be shown the portal inside the node called “Training Node”, but all nodes will have the same setup, only different set of systems, pipes and datasets.

When entering the “Training Node” you will be met with the page seen in figure 1.1.6B.

In this section we will only focus on the specific parts of the portal needed to start working with Sesam, namely the “Pipes” page and the “Systems” page.

When entering the “Pipes” page you will be met by figure 1.1.6C. This figure shows you all the available pipes in your subscription as well as some of their corresponding meta-data. There are also several search and filter options available, which are specially handy when trying to located one, or a set of pipes, in a subscription with many pipes.

If you now enter the pipe called “person-cmm” we can look into more of details regarding how you may use the portal to navigate, troubleshoot and configure you pipes.

Upon entering a pipe you by default be sent to the pipe’s “Graph” view, as seen in figure 1.1.6D.

The graph view shows you which pipes are upstream and downstream to your the specific pipe you have selected, and it also shows connections to related pipes (you will learn more about connected pipes later [link maybe?]). For now we will focus on three of the pipe’s subpages; Config, Input and Output.

- Config: The config subpage is where the actual coding takes place. This is where you define what this specific pipe is supposed to do.

  1.

Something general about JSON

JSON configuration of pipes and systems

DTL also validated as JSON?

2.2.2. Namegiving conventions

How (maybe a table) to give good names to the different parts in sesam

Why this is really important

2.2.3. Systems

Short about systems (where in the sesam-world-map)

Something more general about pipes maybe in context of pipes and datasets

Very low level but enough to set up an inputpipe after maybe?

and refer to systems chapter

Namegivingconventions ref. 1.1.8

Where to make new ref 1.1.6

2.2.4. Pipes

Something more general about pipes maybe in context of systems and datasets

Inbound(Input?)/Preparation/Outbound(Output?)

Very low level but enough to connect to system?

and refer to pipes chapter

Pump

Input & output(sink)

Namegivingconventions ref. 1.1.8

Where to make new ref 1.1.6

2.2.5. Datasets

Something more general about pipes maybe in context of systems and pipes

Very low level but enough to see entities?

and refer to entities subchapter ref. 1.1.12

Namegivingconventions ref. 1.1.8

Where to make new ref 1.1.6

2.2.6. Datasets vs. tables

Examples end ref to 1.1.13

1.

2.

Why globals

Golden records

Gjør data tilgjengelig

Ref. 1.2.19, 3.2.14

2.2.7. Special sesam attributes

Namespace

Rdf:type

_id

2.2.8. 1.1 Tasks for Architecture and Concepts: Beginner

1. *In what component is data stored in Sesam?*

---

2. *Which component moves data in Sesam?*

3. *What moves through Sesam?*

4. Name the input pipe for this system & table:
System name : IFS
Table name: workorder
Pipe name: ______

5. *in an entity representing a row, how would the column “personalid” with row value “123” look after it is read by a pipe named crm-person and stored inside an entity of the output dataset?*

6. *What is the difference between and entity stored as a row in a table vs in a Sesam Dataset?*

7. *What is the minimum required to define an entity?*

2. 16.

The value of joining data

Short overview of what data joining is

1-1, 1-n, n-m

16.

17.

All data from input ends up in output

2.2.9. Left Join (Hops)

Data is appended to the output

2.2.10. Global

Golden – the best truth about common attributes of a concept collected from multiple sources

Coalesce, prioritization of source data (master data)

2.2.11. Generic input pipes, custom output pipes

Write about where globals fit into the bigger picture of data flows, how do pipes going in look and how do pipes going out look?

2.2.12. Filter entities on the way out

Filter gives the ability to stop entities from being sent by providing a logical gate.
On the other hand, it can make sure you only send the entities you wish to receive in an endpoint.

Makes sure the endpoint only receives the entities they want.
Can stop entities from triggering events they shouldn’t trigger.

+ + many examples
filtering on source data
on target data (from hops f.ex) – typical example, hop to global-classification and map status, if cancelled then filter

2.2.13. Tag your entities - Categorization of sub-concepts

Extra:type

2.2.14. Customize data structure for endpoints

Sesam has transformative functions to add, remove,Copy the attributes you want the end system to receive.
All changes to attributes you add to the target will cause an entity update.

Referring to namespace 1.1.15 to know property origin, rename, add, copy

2.2.15. Change tracking & data delta

All entities stored inside sesam have a _hash value. This is a quantification of an entity and is calculated every time an entity is processed by a pipe. If the _hash value changes or is new, the entity will be stored as a new version in dataset. We call this change in _hash value a data-delta.

Any data-delta for an entity in a dataset causes downstream pipes to see this as a new sequence number they haven't yet read. This in turn makes the pipe process the entity. If the processed entity does not exist or gets a new _hash in the output of the pipe, it will cause an update to the output dataset.

2.

3. 26.

Dependency tracking

Stacking av hops (dvs flere datasett)

Indeksering

2.2.16. Incremental System queries

Dataset vs database-tabell (oppdatere data)

2.2.17. Subset

Grabbing the rdf:type or type of data you need from a global

2.2.18. Dynamic, Static & Timeseries Data

Dynamic data = frequent updates to the same object
Static data = rare/never update to the same object
Timeseries = Frequent new entities about the same object. (f.ex _id = meterpoint & timestamp and attribute attached is reading the last hour)

2.2.19. When to use a microservice

For everything Sesam is bad at or can't do.

2.2.20. 1.3 Tasks for Architecture and Concepts: Intermediate

2.3. 1.4 Architecture and Concepts: Advanced

2.3.1. Choosing a source dataset for your pipe

Do not create children based on a hop, rather read from the dataset you hop to.

2.3.2. Eventual Consistency

Dependency tracking causes reprocessing of source entity in the pipe with the hops.

Idempotency

2.3.3. Create Child & Emit children

Change-tracking

2.3.4. 1.4 Tasks for Architecture and Concepts: Advanced

2.4. Epilogue

Congratulations on making your way through this treacherous territory and finding your way out. Let's have a quick chat about the path you've walked before moving on to greener pastures.

3. Systems

3.1. Introduction

In this chapter you will learn about Systems; the component which connects Sesam to the external world. They are an integral component of Sesam and have a wide array of uses, so we will start off by introducing the general knowledge you should have at-hand while setting one up.

Onwards you will learn how Systems interface with Pipes and how to use customized solutions when Sesams built-in systems cannot.

1. 1.

Everything external to the node

1.

2.

Input, output (mention transform?)

1.

2.

How are secrets stored in the backend? – Discuss with product

How do systems read secrets? Encrypted and decrypted in transmission or passed as plain text?

$SECRET

$ENV

3.1.1. JSON Push & Pull protocol

Lots of info in docs.

3.1.2. Tasks for Systems: Beginner

2. 7.

General introduction to what systems are and what problems they solve.
Everything external to the node

3.1.3. Systems as a pipe source

System configuration (mostly) defines the possibilities pipes have to pull data.

We need to write about what a system is in the context of a pipe source, with not only configs but explanations. Keep it simple don’t go into too many system types (json & SQL?). Write more text than configurations, draw stuff. (1-N)

3.1.4. Systems as a pipe sink

Same as above only with system as a sink. What is a system in the context of a sink? What does the pipe see? What does the system see? (1-N)

3.1.5. [System?] Authentication methods

Default authentication methods built in for systems handling URLS $SECRET()
Basic, Oauth2, JWT, microservices

Authentication methods for specific systems: ?? worth mentioning
SQL, oracle

3.1.6. System Types

Mention all built in system types, is there a common denominator?
refer to appendix/documentation for more information

“Type”: “system_XXXX”

1.

2. 12.

How does Sesam look at microservices?

What is a microservice?

How do I use one?

Don’t go too deep, we have a whole module for these.

Probably want to wait with this subchapter until we’ve written the microservices module.

3.1.7. HTTP Transforms

When you need to transform or append information which Sesam isn’t good at handling, you’d use an http_transform.

When you don’t want all the data from a system, but need to append it to the data you’re processing, you’d typically do a http_transform.

Example: You want to get the current weather for a location, but you don’t want to read all the weather around the world constantly into sesam. What you’re interested in is the weather for a location specified by an entity at runtime. You can get this by querying an API per entity being processed.vor

Example: You need to convert UTM to LatLong coordinates. Sesam doesn’t have a function to do this built in, so you make a microservice to do the conversion and call this with an http_transform.

General Example: appending time-dependent datapoints to your output without reading absolutely all of the time-dependented data.

3.1.8. Chaining of Systems

Microservices are easily re-used if they do generic stuff.

The point of chaining microservices or API’s is to use multiple generic, simple services to solve a bigger complex problem.

Pros: Usually re-use of microservices makes development time shorter

Cons: Debugging can be complex and unforeseen issues hard to find & pinpoint. Can’t see it in the graph, need to search the whole node configuration to find the systems.

3.1.9. 2.3 Tasks for Systems: Intermediate

3.2. Epilogue

Summarize the topics the reader has gone through on a very high level.

In relation to the introduction, tell them what they’ve learned and what they should be capable of using this knowledge to do.

4. DTL – The language of Pipes

4.1. Introduction

4.1.1. What is DTL?

Data transformation language - Programming language, mix of JSON & func_programming etc

4.1.2. Why DTL?

What problems does it solve? Why did we need to make a new programming language?

4.1.3. Where is DTL used? – Can fit into 3.1.1 probably.

Where do you write it? Why only in pipes?

1. 1.

What happens when a pipe runs?

What is the relationship of pipes and DTL?

4.1.4. Entities, pipes and `_id` @Geir Atle

What is an `_id`? Why do we need it? Is it used for the same thing always? What is it good for? ~~Absolutely nothing~~ quite a bit!

4.1.4.1. The reserved property `_id`

Everything in Sesam must have a unique identity, whether it is a system configuration, a pipe configuration, a dataset, an entity within a dataset, etc.

The reserved property named `_id` is used as unique identity for components in Sesam.

This unique identity allows for precise references between configurations and precise connections between data entities.

See <ref to `_id` restrictions> for more information on how to create valid identifiers.

4.1.4.2. System `_id`

The identity (`_id`) of a system must be unique within a Sesam node instance.

Once a system configuration is saved, its identity cannot be changed. If you need to change a system’s identity, you can Duplicate the system configuration, save the duplicated configuration with the desired identity, and then delete the original configuration.

Remember to also update any other configurations that were referencing the original system to reference the new identity.

In the Sesam Management Studio, when you view the list of all systems in the Systems menu, the System column will by default show you the identity of all the defined systems in that Sesam node.

If the name property is also defined for a system configuration, then the System column will show that value instead of the identity.

Regardless, if you need to reference a system configuration from another configuration in Sesam, you reference the system’s identity.

See <ref to naming conventions> for more information on system naming conventions.

See <ref to system config> for more information on how to define systems in Sesam.

4.1.4.3. Pipe `_id`

The identity (`_id`) of a pipe must be unique within a Sesam node instance.

Once a pipe configuration is saved, its identity cannot be changed. If you need to change a pipe’s identity, you can Duplicate the pipe configuration, save the duplicated configuration with the desired identity, and then delete the original configuration.

In the Sesam Management Studio, when you view the list of all pipes in the Pipes menu, the Pipe column will by default show you the identity of all the defined pipes in that Sesam node.

If the name property is also defined for a pipe configuration, then the Pipe column will show that value instead of the identity.

Regardless, if you need to reference a pipe configuration from another configuration in Sesam, you reference the pipe’s identity.

See <ref to naming conventions> for more information on pipe naming conventions.

See <ref to system config> for more information on how to define pipes in Sesam.

4.1.4.4. Dataset `_id`

The identity (`_id`) of a dataset must be unique within a Sesam node instance.

By default, a dataset will have the same identity as the pipe it is generated from.

You can override the default dataset identity by defining the dataset property in the pipe's sink configuration. (reference to sink config).

Once a dataset is generated, its identity cannot be changed. If you need to change a dataset's identity, you can edit the dataset property in the pipe's sink configuration, delete the sink dataset, and restart the pipe. This will generate a new dataset with the new identity.

Remember to also update any other configurations that were referencing the original dataset to reference the new identity.

In the Sesam Management Studio, when you view the list of all datasets in the Datasets menu, the Dataset column will show you the identity of all the datasets in that Sesam node.

If you need to reference a dataset from another configuration in Sesam, you reference the pipe's identity.

4.1.4.5. Entity `_id`

The identity (`_id`) of an entity must be unique within the dataset in which it resides. The identity for an entity is similar to a primary key in a database table.

What makes an entity unique is usually dictated by the source system the entity is imported from. This can typically be the primary key(s) of a database table.

This means that you usually define the identity for entities in inbound pipes.

If the source system has multiple properties that combined makes the entity unique, you must combine all these properties into the `_id` property to ensure that uniqueness is preserved in Sesam.

In some cases, you can handle this in the source configuration part of the inbound pipe. SQL sources, as an example, allows you to specify multiple columns from the source database as primary keys. Sesam will then combine these columns automatically into the `_id` during import.

In other cases, you may have to explicitly add the `_id` property with DTL in a transform step in the inbound pipe. This may be relevant when the source configuration does not support specifying multiple properties as primary keys.

4.1.4.6. Entity `_id` and namespaces

By default, the pipe identity of the pipe where the entity originates is used as namespace for both the entity's identifier and the entity's properties.

Note that there is a slight, but significant, difference in the placement of the namespace for the entity's `_id` property compared to its other properties.

For the `_id` property, the namespace prefixes the property **value**:

```
"_id": "<namespace>:<value>"
```

For other properties, the namespace prefixes the property **name**:

```
"<namespace>:property1": "<value>"
```

The reason the namespace is put into the value of the `_id` is to ensure that all entities are unique across all source systems.

Example:

An entity imported from a system called "crm" with a "user" table consisting of a primary key "userId" with value "123", and a column "email" with value "john.doe@foo.no" would look something like this:

```
{
  "_id": "crm-user:123",
  "crm-user:userId": "123",
  "crm-user:email": "john.doe@foo.com"
}
```

Now imagine you have another source where one of the entities are also identified by “123”.

Unless the namespace is part of the property value of `_id`, both entities would have the same `_id`, namely “123”. So by prefixing this value with a namespace we ensure that these entities do not come into conflict with each other.

See <namespace ref> for more info on namespaces.

See <make-ni ref> for more info on namespaced identifiers and connecting data in Sesam.

4.1.4.7. The autogenerated property $ids

Should probably write something sensible about the connection between `_id` and $ids somewhere. Maybe related to merge pipes?

4.1.5. Entity Data model – Data Types @Gabriell

Give quick examples of each of these types.

Dict {}

Entity {`_id`}

Må inneholde en identifikator `_id`

List

String

Integer

Decimal

Float

Boolean

Null

4.1.6. Syntax

All configurations in JSON

[<func_name>, <key>, +<arg/value>]

- Transformative funksjoner, funksjoner for å endre Target
  - Kopierer value fra Source til Target og kan endre key
    - Copy kopierer hele key-value parret.
    - Rename setter ny key og kopierer value
  - Legger til ny Key som et datapunkt. Verdien kan være en transformasjon av et datapunkt eller et helt nytt datapunkt på Target.
    - Add, key, value : Her legger du på en ny key, hvor verdien ikke nødvendigvis finnes fra før. Derfor er det ikke implisitt om det er fra _T eller _S
- Utrykk/Expressions
  - Utregninger, funksjoner på verdier
    1.

Gå gjennom prosessen fra man trykker "New pipe" til "Save" til "Start" til "Restart"

- Sette `_id`
- Bruke templater
  - Source system "sesam:node" (refers to itself)
    - Provider: premade dataset

- "add DTL transform"
- ["add", "hello", "world"]
- Save
- Starte
- ["add", "key", "value"]
- Save
- Start - ikke noe nytt i output
- Referer "Proessser ny data" over, vis det også.

1.

Preview, Ctrl + Enter

Formatering alt + .

Save ctrl + s

Find/replace

Ctrl+space = Search/autocomplete

1.

2. 7.

Explain copy, based on ref 3.1.4 above

Wildcard * [namespace:*]

"Copy" whitelist, blacklist

4.1.7. “Add”

Explain the add, based on ref 3.1.4 above

4.1.8. “Concat” – Concatination

Concatenation of strings, examples etc

4.1.9. rdf:type

Resource Description Framework (?) explain what it means in Sesam context

4.1.10. Namespace

Explain namespace in `_id` (value) and keys.

EXAMPLESSS

4.1.11. “Make-ni”

Declaraiton of foreign key in Sesam, explain /reference Namespace

4.1.12. “Eq” – Equality

Equality for joins [n-n]

4.1.13. Merge as a Source

Examples, steal from PP training, show in tables vs json, everything coming in goes out.

- Strategy
- Identidy - `_id` etter merge

- datasets

7.

Explain in the context of reading from global pipes

4.1.14. Coalesce

ref 1.2.19

4.1.15. Nested dictionaries

As you can see in *Example 3.2.17A: Dotted Notation*, we can get attributes inside dictionaries by using “.”

Dotted notation

list of dicts can give you list of values from a single key.

A: [{“foo”:1},{“foo”:2}] -> _S.A.foo = [1,2]

1. [“add”, “some-nested-attribute”, “_S.somedict.some-nested-attribute”]

Example 3.2.17A: Dotted Notation,

4.1.16. Apply – Custom Functions

Basic, bare bruk på data fra _S, forklar det uten å bruke hops

4.1.17. Merge as a function

Source type Merge VS Transformation Merge

Merging dictionaries up to the root level of entities.

4.1.18. Hops

Basics, uten apply

4.1.19. _ Properties

(_deleted, filtered, `_id`, _previous, _updated, *_hash? REF 1.2.24*)

4.1.20. Type examples

Type eksempler:

• Datettime

• Dict {}

• List

# First

# Unique/Distinct

# Last

# Count

# nth

• String

• Integer

• Decimal

• Float

• Boolean

# And

Or

# Not

In

Eq

# If-null

# Is-empty

1.

2. 23.

• Pump

# Dead-letter…

• Metadata

• Reset-to-end

• Stop

• Enable/disable

4.1.21. Pipe Sink

Eksplisitt (vs implicit dataset) Sink - til system/fil

23.

24.

25.

4.1.22. 3.3.27 Dependency tracking in Hops

When does dependency tracking work? How does it work? When doesn’t it work (multiple transforms) Ref 1.3.25

4.1.23. "Apply-hops"

Apply a function to the entities retrieved by your hop

4.1.24. Source Subset

You don’t really need to filter :P

4.1.25. Tasks for DTL: Intermediate

4. 30.

VS as a transform (filter objects in list)

4.1.26. _. Syntax and Functions

_. : path, map, filter, what does it reference? How does it work?

4.1.27. Map

Map, map-values, map-dict

4.1.28. _P & _R – Parent & Root

How do I use _P. notation? Where does it point?

4.1.29. “Create-child”

1-N

dep. Tracking, $children, emit_child transform type (2 pipes necessary for all updates to propagate)

30.

31.

32.

Add ::hello

4.1.30. 3.4 Tasks for DTL: Advanced

4.2. Epilogue

Summarize the topics the reader has gone through on a very high level.

In relation to the introduction, tell them what they’ve learned and what they should be capable of using this knowledge to do.

5. 4. Projects & Infrastructure

5.1. Introduction

@todo Gabriell

5.2. 4.1 Projects & Infrastructure Beginner

5.2.1. Portal GUI

Bli kjent med gui

Datasets

previous version etc.

Task på å sette opp ting som gjøres når man er i prosjekt

Laste opp/ned node i tools

Legge til brukere

Legge til env-vars/secrets (system secrets vs secrets)

Lage JWT

se på Execution logs/system dataset

system:config-dataset

Lage grupp/tilganger

5.2.2. sesam-CLI

NB!! IKKE BRUK SYNCCONFIG TIL Å LASTE OPP/NED TIL AKTIVE NODER (PROD)

pre-requisite lære seg hvordan man installerer det.

lag en sesam-init <- feature request

setup

expected folder

test.conf.json

whitelist/blacklist

test.json

entiteter

env-var-folder

set up vars for different environments

test-env

.syncconfig

jwt, node

kommandoer

sesam upload/download

test

update

-print-scheduler-log

-vv

-use-internal-scheduler

wipe

restart

verify

run

-version

Hvordan funker expected output

5.2.3. Testing & Testdata

testing

Manuell testing med sesam-cli før opplasting til versjonskontroll

Manuell testing med config-group på live node

Automatisk testing med ci-node

Testdata

Bør lage data som reflekterer virkelige koblinger mellom data i systemer

Bør være nok for å beskrive de caser man kan møte i virkeligheten

Bør ikke være all data i prod

Bør være anonymisert

Bør reflektere *innkommende* data

Bør utvidet behov legges til data, ikke endre eksisterende

Bør gis navn utfra det case du vil teste, f.eks gi entiteten navn utfra casen

Dokumenter testdata

\\oppdater prosjekt i docs utfra hva vi skriver\\

Hvordan funker expected output

5.2.4. Dokumentasjon

Hvordan bruke docs.sesam.io

developer guide!!

ctrl + f "hva du tror funksjon heter"

Hvordan dokumentere

Schema definition

hva mener vi er dokumentasjon

Generell dokumentasjon

DTL dokumentasjon(comments)

clean code

5.2.5. JWT/Authentisering

Hvordan fungerer JWT’er?

NB: Skal snake mer om API I sesam-in-the-wild

5.2.6. Groups & Permissions

Hvordan virker det

Får man satt opp tilgangsstyring i Sesam?

1.

2. 7.

Mappestruktur

System

Pipes

Node-metadata.conf.json

+expected

Global metadata

Namespaces

Tag for å inkludere c++ ext.

Dependency tracking hops limit

signalling

Referer mye til dokumentasjon

5.2.7. Deployment

Når trenger man å resette pipes?/Når trenger man ikke det

Update last seen

reset to end

reset

Disable/enable pipes (spesifik endpoint)

Indexering

5.2.8. Monitorering

microservices

pipes

ekstern monitorering

Execution logs/system dataset

5.2.9. Arbeidsmetodikk

1. Dokumentere source-data og sink-data før en flyt

formater

datamengde

frekvens

2. Analysere innkommende data for globala dataset

3. Lage testdata

4. Drøfte behov & Løsninger

5. Velge løsning

6. Lag løsning

Mer?

1.

2. 8.

dev = personlig node der utvikling foregår

test = node som kjører samme config som prod med testdata for å finne bugs

CI = do tests for pull requests /deployments before deploying.

prod = live node som kjører live integrasjoner

Tagging av brancher for deployment

5.2.10. CI/CD/TDD?

ci-node

kjører med test-data

embedded data

NB!! nye cli fra 1.18.1(separat testdata-fodler)

NB!! ikke koblet til live systemer, ikke legg inn secrets som ikke skal være der

node-env

conditional source

embedded data

NB!! nye cli fra 1.18.1(separat testdata-folder)

conditional transform

Hvordan bruker vi versjonskontroll(git, vcs, svn)

initiere repo (se docs)

protected branches

merge regler

byggserver

Autodeploy/vs ikke

5.2.11. Workflow in Projects

Get task

[Document task]

Pull repo

Create branch

Do changes

Test changes

[Create more test cases]

Update expected data

Push changes

Document solution

Deploy to test

Test changes in test – go back to create branch if necessary.

Deploy to prod

5.2.12. Tasks for Projects & Infrastructure: Intermediate

5.3. Epilogue

Summarize the topics the reader has gone through on a very high level.

In relation to the introduction, tell them what they’ve learned and what they should be capable of using this knowledge to do.

6. Microservices

6.1. Introduction

1. 1.

Nevn bruksområder

språk

Docker

6.1.1. Why use Microservices in Sesam?

System som gjør ting andre systemer ikke kan

6.1.2. How are Microservices used in Sesam?

Ekte Use caser

6.1.3. Microservice hosting

Sesamcommunity Git & Docker

Intro til Hosting

6.1.4. Running a microservice in Sesam

Intro til Running I sesam

Forklare GUI

Pull & Restart

    Status

    Refresh

Forklare Config

Pipe source/sink/http

6.1.5. Types of Microservices

    Interne

    http-transform

    Source, sink (begge i 1?)

    Eksterne

    Monitorering av Sesam

6.1.6. Naming Convention

_id standard system naming convention (source/sink system name)

Repo/microservice naming convention recommendation: sesam-<system>[-<special-functionality>]

6.1.7. Tasks for Microservices: Beginner – Tasks

Run a microservice in Sesam [could be sink, http, source]

1. 8.

Sesam push/pull protocol

Error handling/logging

Pipes

Statuslogg [Hvordan bruke & Lese]

Log-level

Trigger/Call

Endepunkt i ms (/<paramater>)

Hvordan sender man info til ms fra sesam

url-i pipe

url-parameter

6.1.8. Microservice Development Prerequisites

Docker

User

Program

GitHub

User

CLI/Desktop

6.1.9. Changing a Microservice

Workflow

Fork [Vi må lage et repo]

Change

Test

Teste lokalt

Bygge docker konteiner

Pushe docker konteiner

Explanation of Bare Bones DockerFile

How DockerFiles run [Sequentally, cache]

6.1.10. Authentication with microservices

Docker hosting site login

Passing Env-vars/Secrets i Sesam

Oauth2 standard – Salesforce microservice

6.1.11. Sesam I/O

Common for sesams input & output

Sesam push/pull protocol

Sesam-json (formattering)

Lister av entiteter

query-parameter

url-parameter

is-first

is-last

6.1.12. Using a Microservice as Input in Sesam

Inside sesam

Best practise:

Delta/last seen

request-params

is-first

is-last

6.1.13. Looking inside an Input Microservice

Inside the microservice

Transparens (minst mulig transformasjon i microservice)

Måter å returnere entiteter på (Transform i MS vs transform i pipe)

Streaming

Logging

Gi gode feilmeldinger på http, catch spesifikke exceptions

6.1.14. Tasks for Microservices: Novice

Run a microservice in Sesam [could be sink, http, source]

2. 15.

Ukjent: Business logikk

Eventual Consistency 1.4.30

NB!! _properties blir med ut!! NB!!

Filter

_filtered - blir ikke sendt videre

_deleted - blir sendt videre

Endpoints fjerner namespaces

Batching/streaming

NB! siste batch sendt fra sesam er alltid en tom liste

6.1.15. Looking inside an Output Microservice

Create vs update

Formattering

Transparens (minst mulig transformasjon i microservice)

transit-encoding fra sesam

Logging

Gi gode feilmeldinger på http, catch spesifikke exceptions

Batching/streaming

NB! siste batch sendt fra sesam er alltid en tom liste

6.1.16. Guidelines for Microservice Development

Check if it already exists

Documentation: Readme

Define Scope

Gjenbrukbarhet

Sesamutils

Templates

Env var for dynamiske MS’er

Videreutvikling

Release/tagging

Ref. Optimalisering 5.3.17

Requirements.txt

6.1.17. Microservices and GitHub [VCS]

Lisenser

For community bruk

For privat bruk

Community github/slack/stackoverflow

Krav til microservices i sesam-community

Videreutvikling

Release/tagging

Byggserver – Travis, Azure ContainerRegistry

For community bruk

For privat bruk

Reference the “5.2.8 Changing a Microservice” for workflow

Ref appendix with complete microservice workflow (create a sequence diagram Gabriell/Daniel?)

6.1.18. Optimizing a Microservice

Minnebruk

Streaming / Yield

Delta/last seen

Transparens (minst mulig transformasjon i microservice)

6.1.19. Microservice System types

Lots of examples!

How should microservices which read or write to/from these types work? What have we learned?

Source & Sink

Apier

    Paging

    Update VS Create

Filer

sFtp

SOAP

6.1.20. Tasks for Microservices: Intermediate

Run a microservice in Sesam [could be sink, http, source]

Create a microservice

1. 21.

Env vars

Lokal testing

Returnerer riktig format (json som sesam kan lese)

Unit testing

[Experimental] Undersøke:

[Experimental] Morten? (docker-compose att: Gabriell)

[Experimental] !!NB!! Definer testing i ms // Lag en test-ms-template // Implementer MVP testing på sesam-community [great expectations python-lib Daniel har info]!!

6.1.21. Proxy Endpoint [Jonas]

Kan lage en ms med frontend f.eks og eksponere den fra sesam

6.1.22. Chaining

Ref advanced system 2.4.13.

6.1.23. Tasks for Microservices: Intermediate

Run a microservice in Sesam [could be sink, http, source]

Create a microservice

6.2. Epilogue

Summarize the topics the reader has gone through on a very high level.

In relation to the introduction, tell them what they’ve learned and what they should be capable of using this knowledge to do.

7. Sesam in the Wild [WIP]

7.1. Introduction

7.2. Sesam in the Wild: Beginner

7.2.1. 6.1.1 Beginner topic

7.2.2. 6.1 Tasks for Sesam in the Wild: Beginner

7.3. Sesam in the Wild: Novice

7.3.1. 6.2.X: Novice topic

7.3.2. 6.2 Tasks for Sesam in the Wild: Novice

7.4. Sesam in the Wild: Intermediate

7.4.1. 6.3.X: Intermediate topic

7.4.2. 6.3 Tasks for Sesam in the Wild: Intermediate

7.5. Sesam in the Wild: Advanced

7.5.1. 6.4.X: Advanced topic

7.5.2. 6.4 Tasks for Sesam in the Wild: Advanced

7.6. Epilouge

8. Appendix

1.X.X

2.X.X

3.X.X

4.X.X

5.X.X