I will start by describing components and the relevant processes.

* Web site – angular.
* Web API - .NET.
* MSSQL - Keep file metadata.
* AWS S3 – Media files storage.
* Grafana, Influx DB – Monitoring and alert system.
* Message bus – Kafka, will be used to pass data to monitoring.

**Web site**

The web sit will contain 3 major pages:

1. Manage stream data – create topic, upload files.
2. Stream page – Stream data viewer according to selected topics.
3. Manage permissions.

**Web API**

Will be written in .NET (latest stable version)

2 controllers:

FileStorage

Upload file

Delete File

MetaData

CRUD on meta data

**Monitoring Service**

Use to monitor and alert data according to definition.

A screenshot of a computer

Description automatically generated

Components:

1. Monitor service.
   1. Register to Kafka for specified topic.
   2. Add data to influx DB – time series DB.

We will use influx DB for add relevant monitoring data:

* + - * Users who updated data,
      * Upload file to s3 latency,
      * Upload errors,

Create alerts for unusual use according to definition.

<https://docs.influxdata.com/platform/getting-started/>

* 1. Grafana – Show relevant dashboard. For example,
     1. Who upload files and how much time it takes.
     2. Errors on any update contents.

Why different service?

Because of memory and CPU. We don’t want to share main system memory and CPU usage.

With monitor service.

Why Kafka?

In case service is stop working for any reason we will keep write the monitoring data on start

From the place we stopped.

In addition, we can work in a group mode if we decide to add more monitoring services.

**MSSQL/MySQL**

Table: FileMetadata – keep metadata on media files

|  |  |
| --- | --- |
| **ID** | **Int** |
| **Topic** | **Varchar (100)** |
| **Description** | **Varchar (1000)** |
| **Url** | **Varchar (500)** |
|  |  |

Table: history – who did what. In addition to influx this data will be kept on SQL.

It will allow us to present history on a different page, and it will show who did what.

We cannot show these values with influx.

|  |  |
| --- | --- |
| **ID** | **Int** |
| **User** | **Varchar (100)** |
| **Action** | **int** |
| **Old value** | **Varchar (max)** |
| **New value** | **Varchar (max)** |

**AWS S3**

The data will be saved in a bucket.

The bucket will contain folders according to topic.

For example

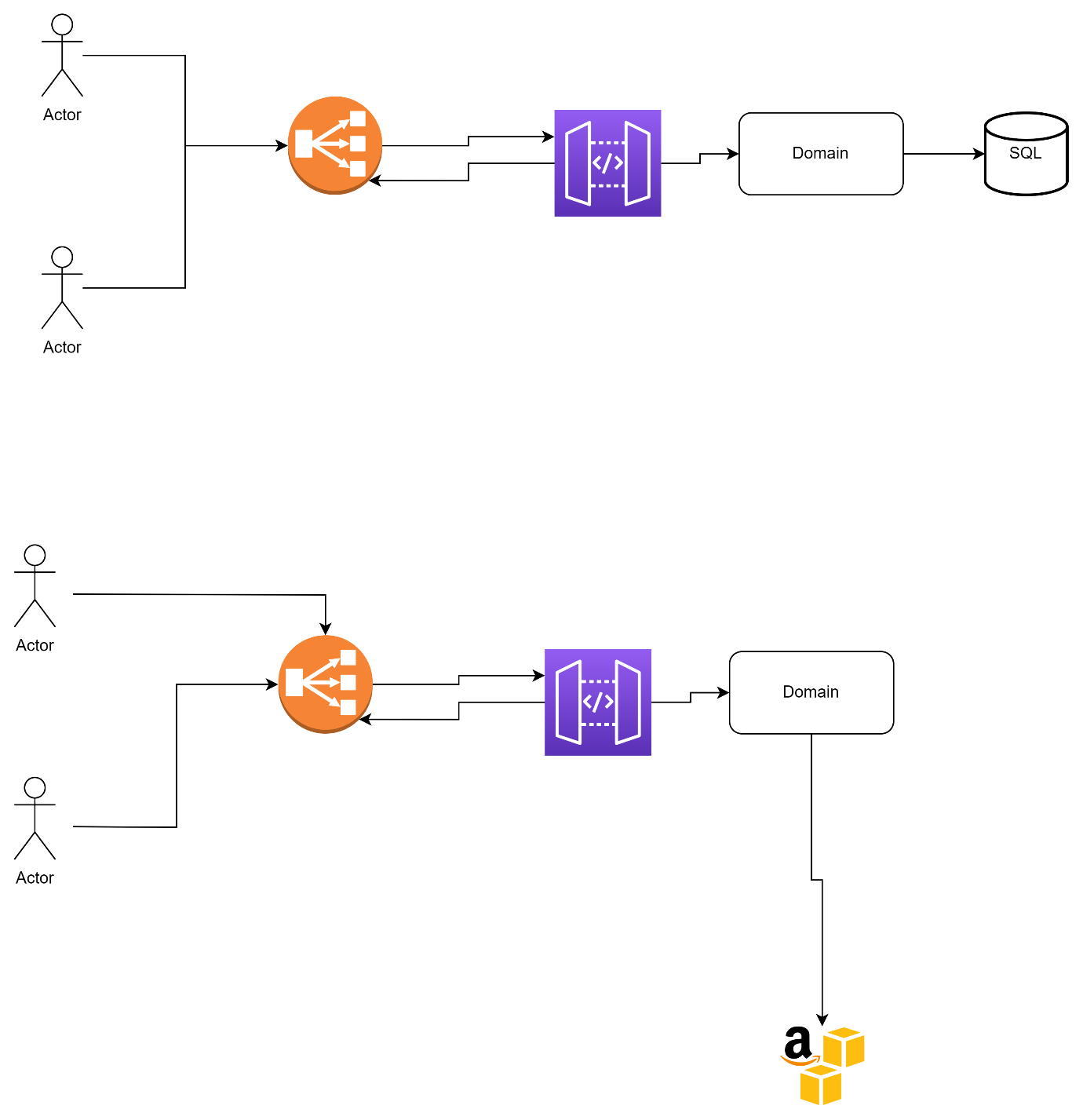
<https://keshet-test.s3.amazonaws.com/topic-1/VID20230716220532.mp4>

Why SQ and not DB?

* Scalability: With S3, you can store and retrieve any amount of data at any time, without having to worry about running out of space on your server.
* Durability: S3 stores your data across multiple devices and locations, so you don't have to worry about data loss due to hardware failure.
* Cost-effectiveness: S3 is often more cost-effective than storing large amounts of data locally, especially if you're using cloud-based hosting.
* We can have access from multiple places using permission on URL.

We can use Azure as well using cloud file storage or Google cloud using cloud storage.

CRUD – Load meta data and files

****

**Tasks - assumption – developers are full stack.**

Tasks:

1. Build web pages according to definitions.
2. Build monitoring using Kafka, influx DB and Grafana.
3. Keep metadata in MSSQL using entity framework.
4. Store files in S3, build s3 bucket, create dev environments and prod environment.
5. Crete Permission mechanism for managing and viewing files.
   1. Admin – can do all including change permissions.
   2. Editor – can manage stream files and view stream data.
   3. Viewer – can view stream data.

We need to enable plural permission (on group of employees) and singular (1 employee).

\*\*We will use domain architecture for all process using .Net

\*\* Business logic (domain) will include unit test and integration test

\*\* code review – will be done by senior developers / team leader. At list 2 different people and only after unit test are pass.

Git will be used as source code.

User stories will be managed in Jira and will be open under relevant epics.

1. **Monday user management:**

**Solution 1**: Set password policy that matches LDAP policy.

**Solution 2**: Add notification on active directory when user is blocked / deleted.

On notification – run script that deletes user from Monday.

**Solution 3**: Using azure AD with Monday to manage users.

<https://support.monday.com/hc/en-us/articles/360018360819-SCIM-Provisioning-of-Users-and-Teams-with-Azure-AD>