POLITECNICO
MILANO 1863

Software Engineering 2

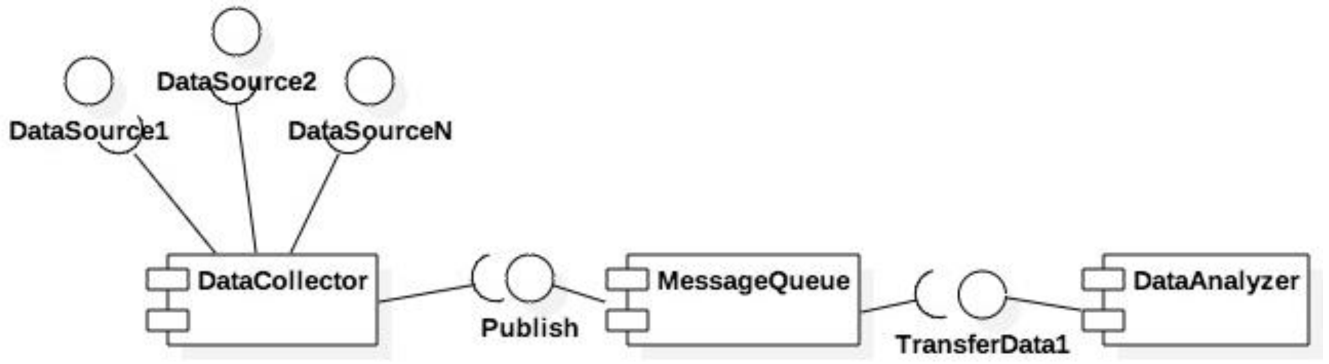
Software Design Exercises

Data Analysis Architecture

Exercise on Alloy and architectures

(exam of July 13th, 2018)

- Consider the following UML component diagram.



This diagram describes a software system that acquires and elaborates information from a number of different sources by polling them periodically. The DataCollector component is exploiting the interfaces offered by some data sources to acquire data and the interface of the MessageQueue to pass collected data to the other components. The MessageQueue exploits the interface offered by the DataAnalyzer to pass the data to this component.

Exercise (cont.)

- Write in Alloy the signatures that model a DataSource, a DataCollector, a MessageQueue and a DataAnalyzer. Make sure that you represent in the model the connections between components that are highlighted in the UML component diagram.
- Assume that we decide to replicate the DataCollector component. Model in Alloy the following possible configurations of the system:
  - **Configuration 1**: Each DataCollector replica is connected to a disjoint subset of DataSource components.
  - **Configuration 2**: All DataCollector replicas are connected to all DataSource components.
  - **Configuration 3**: DataCollector components are classified in master and slaves. There is always one DataCollector that acts as *master*.

A possible solution

```
sig DataSource {}

sig DataCollector {
  sources: set DataSource,
  queue : MessageQueue
}

sig MessageQueue {
  analyzer: DataAnalyzer

}


sig DataAnalyzer {}
```

A possible solution (cont.)

```
sig Configuration {
  sources: set DataSource,
  collectors: set DataCollector,

  queue: MessageQueue,
  analyzer: DataAnalyzer
}
{ // all DataCollector components are connected to the
  // same MessageQueue, which is connected to the
  // DataAnalyzer of the configuration
  all coll : collectors | coll.queue = queue
  queue.analyzer = analyzer

  // also, the DataSource components used by the
  // DataCollector ones are exactly
  // those of the configuration

  collectors.sources = sources
}
```

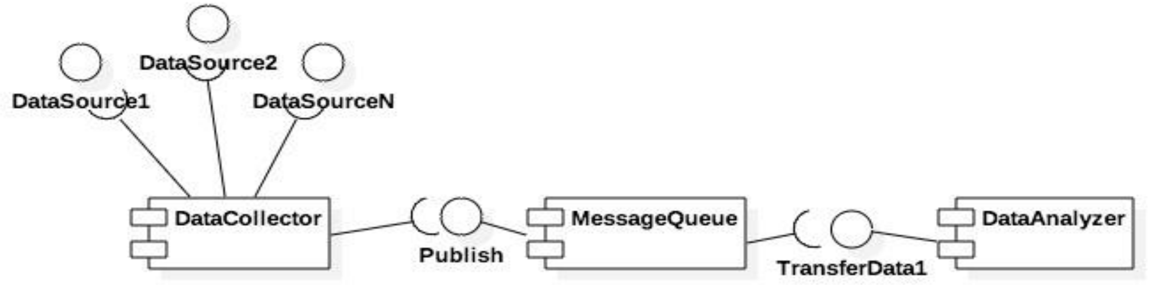
A possible solution (cont.)

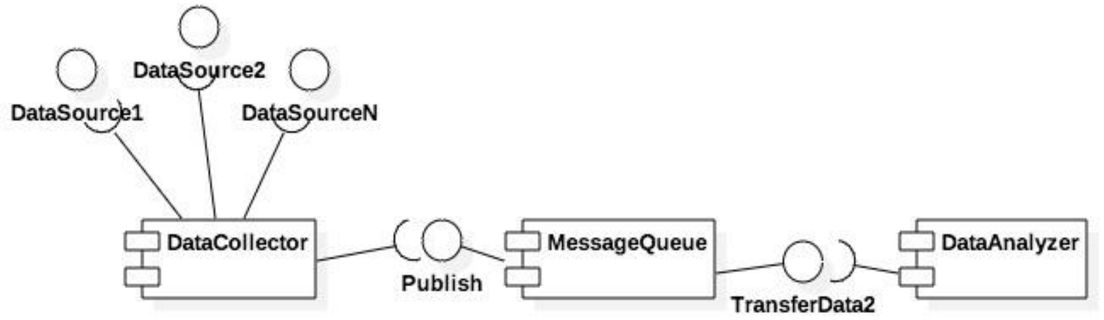
```
// We capture the different configurations through
// extensions of the Configuration
// signature above; they add the necessary constraints

sig Configuration1 extends Configuration{}
{ all disj coll1, coll2 : collectors |
        coll1.sources & coll2.sources = none }

sig Configuration2 extends Configuration{}
{ all coll : collectors | coll.sources = sources }

sig MasterDataCollector extends DataCollector {}
sig SlaveDataCollector extends DataCollector {}

sig Configuration3 extends Configuration{}
{ all coll : collectors |
   coll in (MasterDataCollector | SlaveDataCollector)
  one coll : collectors | coll in MasterDataCollector
}
```

More on the data analysis example

(from the exam of June, 27th 2018)

- Consider the following two versions of the same system





- Q1: What is the difference between the two?

More on the data analysis example

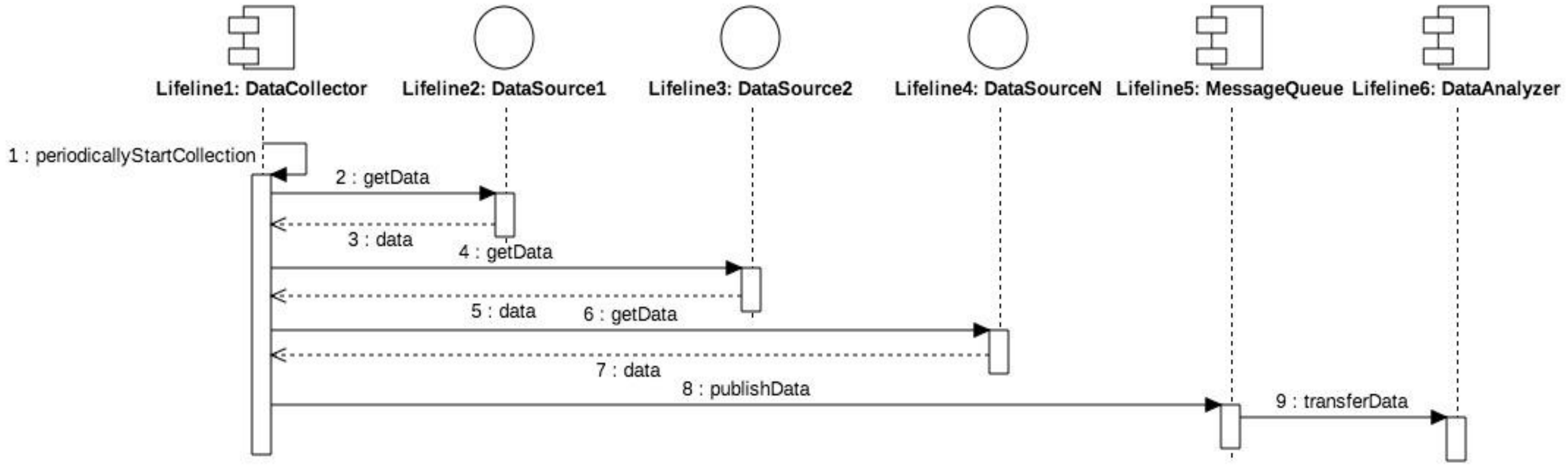
(from the exam of June, 27th 2018)

- Solution
  - In the second case, the MessageQueue does not actively push the data to the DataAnalyzer, but it offers interface TransferData2 so that the DataAnalyzer can pull data as soon as it is ready to process them. Also in this case, both a batch or a per data approach is possible. The rest of the system behaves as first one.
- Q2: Define two sequence diagrams that describe how data flow through the system in the two versions of the architecture

More on the data analysis example

(from the exam of June, 27th 2018)

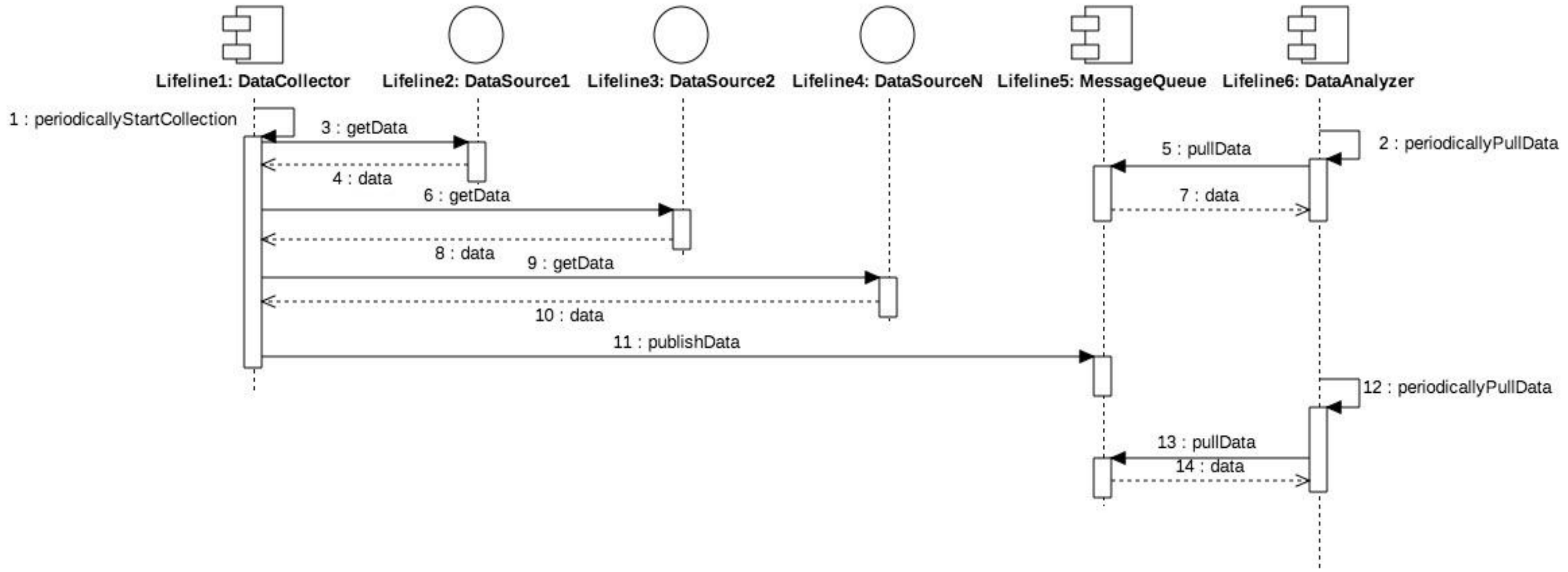
- Solution to Q2
  - Sequence diagram compatible with the first component diagram



More on the data analysis example

(from the exam of June, 27th 2018)

- Solution to Q2
  - Sequence diagram compatible with the second component diagram



More on the data analysis example

(from the exam of June, 27th 2018)

- Assume that the components of your system offer the following availability:
    - DataCollector: 99%
    - MessageQueue: 99.99%
    - DataAnalyzer: 99.5%
- Provide an estimation of the total availability of your system (you can provide a raw estimation of the availability without computing it completely).

More on the data analysis example

(from the exam of June, 27th 2018)

- Data flow through the whole chain of components to be processed => series of component.
- The total availability of the system is determined by the weakest element, that is, the DataCollector.
  - $A_{Total} = 0.99*0.9999*0.995 = 0.985$
- Assuming that you wanted to improve this total availability by exploiting replication, which component(s) would you replicate? Please provide an argument for your answer.

More on the data analysis example

(from the exam of June, 27th 2018)

- If we parallelize the data collector adding a new replica, we can achieve the following availability:
  - $(1-(1-0.99)^2) *0.9999*0.995 = 0.995$
- if we increase the number of DataCollector replica, we do not achieve an improvement as the weakest component becomes the DataAnalyzer.
- We can parallelize this component as well to further improve the availability of our system.

- How would such replication impact on the way the system works and is designed?

More on the data analysis example

(from the exam of June, 27th 2018)

- Let’s consider the impact of the DataCollector parallelization on the rest of the system.
- If both replicas acquire information from the same sources in order to guarantee that all data are offered to the rest of the system, then the other components will see all data duplicated and will have to be developed considering this situation. For instance, the MessageQueue could discard all duplicates.
- Another aspect to be considered is that both DataSources and MessageQueue have to implement mutual exclusion mechanisms that ensure the communication between them and the two DataCollector replicas does not raise concurrency issues.
- Another option could be that only one DataCollector replica at a time is available and the other is activated only when needed (for instance, if the first one does not send feedback within a certain timeout).

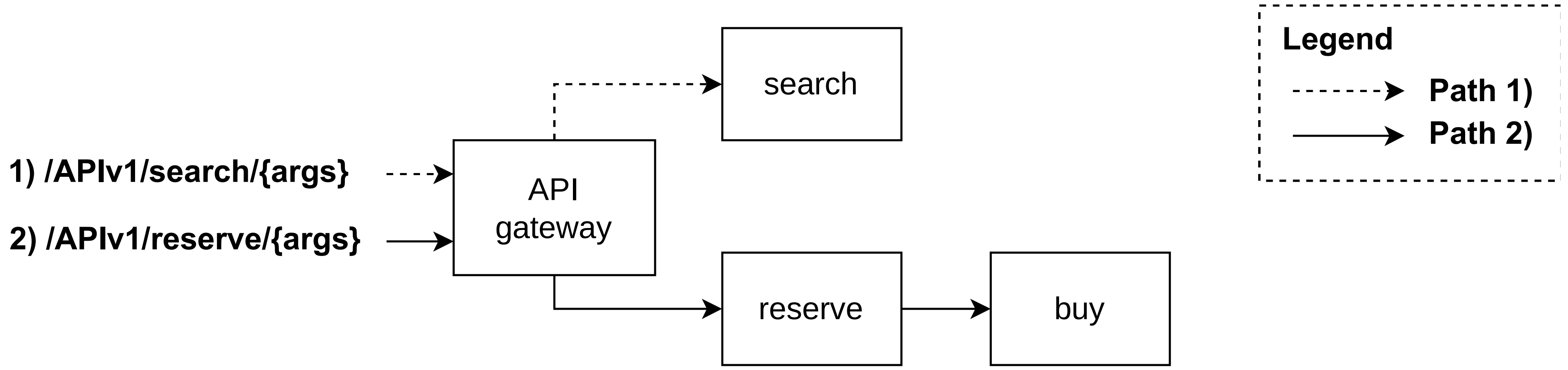
TrainTicket

TrainTicket (Sept 7th 2023 - WE1 exam)

- Consider a microservices application called TrainTicket composed of 3 domain microservices (search, reserve, buy) and 1 additional microservice that acts as the API gateway. TrainTicket supports two basic operations invoked using the exposed RESTFul APIs:
    1. search: /APIv1/search/{args}
    2. reserve: /APIv1/reserve/{args}
- Requests (both search and reserve) are received and then dispatched by the API gateway.

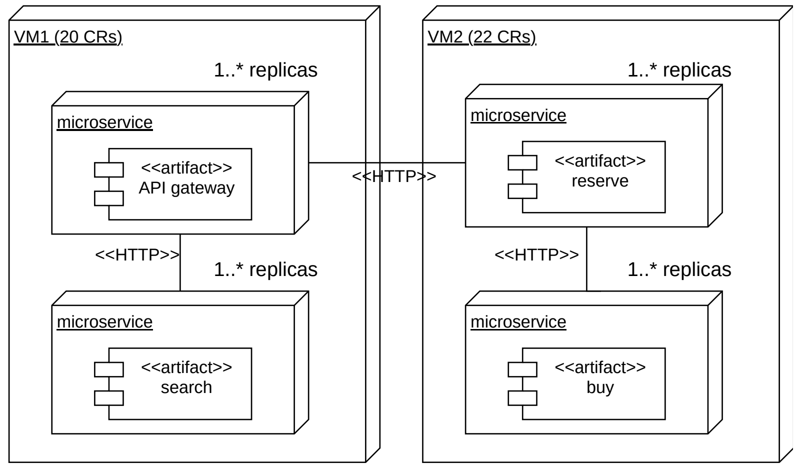
TrainTicket (Sept 7th 2023 - WE1 exam)

- In particular, the following high-level schema shows how requests propagate from the gateway to internal microservices. Note that in this example reserve includes also the purchase of reserved items.



TrainTicket (Sept 7th 2023 - WE1 exam)

- Microservices run in units deployed onto 2 different Virtual Machines, VM1 and VM2 as shown in the following UML deployment diagram.
- The available VMs have Computational Resources (CRs) that can be allocated to run microservices. Each VM has a maximum number of CRs and each microservice requires a certain number of CRs, according to the executed artifact. As shown in the schema, available CRs are as follows:
  - VM1: 20 CRs
  - VM2: 22 CRs



TrainTicket (Sept 7th 2023 - WE1 exam)

- The mapping between microservices and required CRs is as follows:
  - API gateway: 2 CRs, search: 5 CRs, reserve: 4 CRs, buy: 5 CRs
- The deployment diagram shows that each microservice can be replicated to have redundant business-critical components. In the latter case, requests are directed to all the replicas rather than to an individual instance and the first answer received from a replica is returned to the caller, while the others are simply ignored. The number of replicas for each microservice shall be defined so that the following nonfunctional requirement is satisfied and the deployment constraints defined in the deployment diagram and above are fulfilled.
- R1: “Both search and reserve services exposed through API gateway shall have availability greater than or equal to 0.99”.

TrainTicket (Sept 7th 2023 - WE1 exam)

- **Availability_1 (3 points)** Considering the constraints of the execution environment represented above, determine whether requirement R1 can be satisfied or not assuming the following availability estimates for each microservice:
  - API gateway: 0.99
  - search: 0.98
  - reserve: 0.95
  - buy: 0.91
- Justify your answer.

TrainTicket (Sept 7th 2023 - WE1 exam)

- **Answer**: Considering the execution environment, we can derive the following inequations constraining the number of replicas:
  - Constraints extracted from environment:
    - $2x + 5y <= 20$
    - $4u + 5z <= 22$
  - Constraints extracted from requirement R1:
    - $(1 - (1 - 0.99)^x) * (1 - (1 - 0.98)^y) >= 0.99$
    - $(1 - (1 - 0.99)^x) * (1 - (1 - 0.95)^u) * (1 - (1 - 0.91)^z) >= 0.99$
- Where variables x, y, u, and z represent the number of replicas for the microservices *API gateway*, *search*, *reserve*, and *buy*, respectively.
- The requirement R1 can be satisfied since there exists a valid assignment to variables that satisfies all constraints. For instance:
  - $x = 2$, $y = 2$, $u = 3$, $z = 2$

TrainTicket (Sept 7th 2023 - WE1 exam)

- **Availability_2 (2 points)** Consider the problem of resource allocation taking into account the operational profile, that is, the behavior of the users. Assume the following workload in terms of average number of concurrent users for each request:
  - search: 50 users
  - reserve: 90 users
- Assume also that for reserve, only 20% of users complete the purchase at reservation time. This means that 20% of reserve requests get through and reach the buy microservice, while 80% of them terminate the execution without calling buy.

TrainTicket (Sept 7th 2023 - WE1 exam)

- After a preliminary analysis, we realize that availability depends on the workload according to the following new estimates:

LOW workload: 0-60 concurrent users		HIGH workload: 60-150 concurrent users	
Microservice	Availability	Microservice	Availability
API gateway	0.99	API gateway	0.98
search	0.98	search	0.95
reserve	0.95	reserve	0.93
buy	0.91	buy	0.90

- Does the execution environment have enough computational resources to support the workload defined above still fulfilling requirement R1 and the defined constraints? Justify your answer.

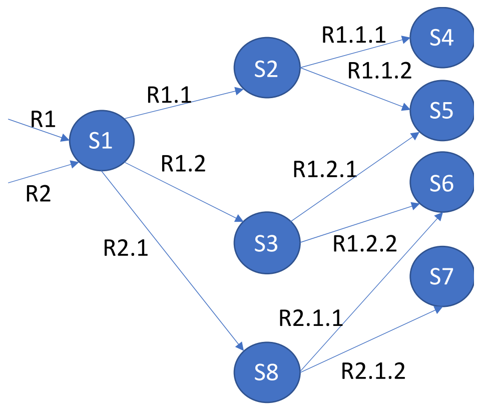
TrainTicket (Sept 7th 2023 - WE1 exam)

- The expected workload for each microservice is as follows:
  - API gateway: 140 users (HIGH)
  - search: 50 users (LOW)
  - reserve: 90 users (HIGH)
  - buy: 18 users (LOW)
- The constraints extracted from requirement R1 become as follows:
  - $(1 - (1 - 0.98)^x) * (1 - (1 - 0.98)^y) >= 0.99$
  - $(1 - (1 - 0.98)^x) * (1 - (1 - 0.93)^u) * (1 - (1 - 0.91)^z) >= 0.99$
- An optimal resource allocation is represented by the assignment x = 2, y = 2, u = 3, z = 2 that is again feasible according to environment constraints.

Microservices and availability

Sept 7th, 2022 exam

- Consider the microservice-based architecture shown in the figure below. The architecture is organized in eight stateless microservices that collaborate to fulfill requests R1 and R2. S1 is the front-end service that receives both requests. The fulfillment of request R1 requires the interaction with services S2 and S3 (through sub-requests R1.1 and R1.2, respectively), which, in turn, need to interact with other services. In particular, S2 interacts with S4 and S5 and S3 with S5 and S6.
- The fulfillment of R2 requires that S1 interacts with S8 which, in turn, interacts with S6 and S7.



Service	Avail
S1	0.99
S2	0.9
S3	0.9
S4	0.95
S5	0.999
S6	0.99
S7	0.99
S8	0.95

Sept 7th, 2022 exam

- Assuming that the availability of services S1-S8 is the one reported in the table above, what is the availability of the system when answering to request R1?
- If each of the services S1-S8 is duplicated, what is the new value of the availability computed at point 1?

Sept 7th, 2022 exam (Solution)

- Since we have no information that suggests the opposite, we can assume that the availability of each services can be associated with any of the requests the service is able to answer. This implies that, for instance, the availability associated with the request R1.1.2 is the same associated with the request R1.2.1 and corresponds to the availability of service S5.
- Considering that the fulfillment of R1 requires, besides the execution of S1, also the delegation of the corresponding sub-requests to the other services, we can write:

$A_{R1} = A_{S1} * A_{R1.1} * A_{R1.2} = A_{S1} * (A_{S2}*A_{S4}*A_{S5}) * (A_{S3}*A_{S5}*A_{S6}) = 0.753$

- Notice that S5 is involved in two sub-requests and, therefore, its availability occurs twice in the expression above.

Sept 7th, 2022 exam (Solution, cont.)

- If all services are duplicated, we can replace in the formula above the availability of each service with the one obtained by duplicating it. So:

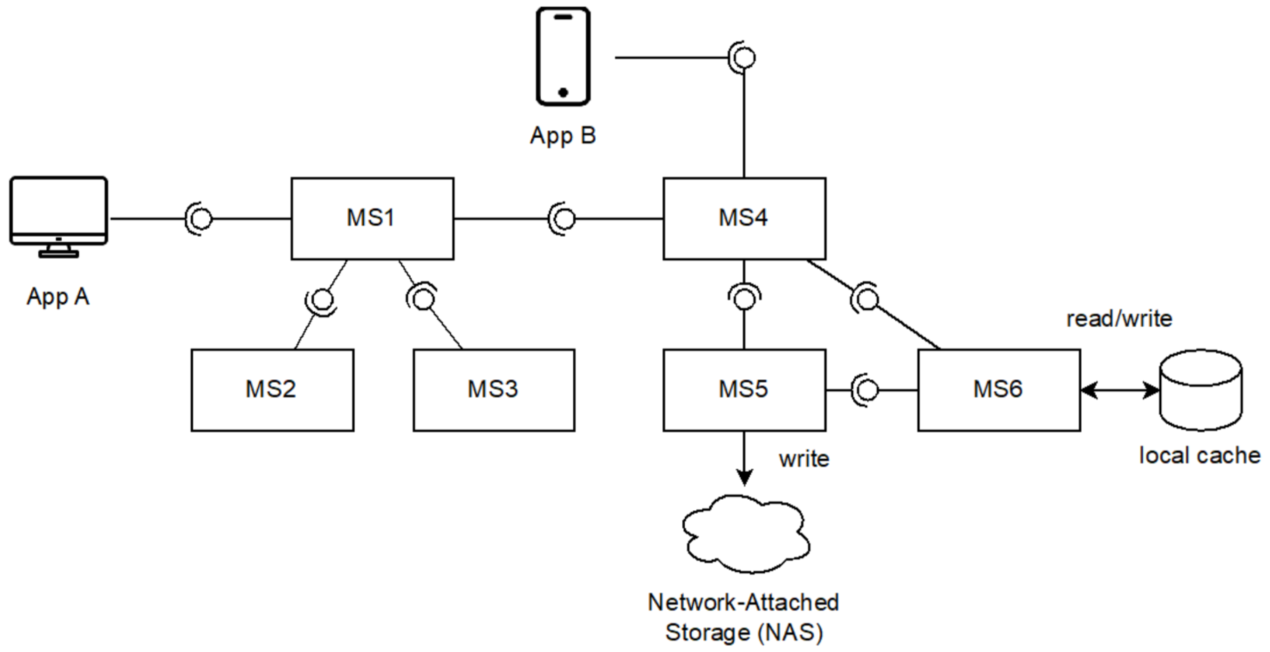
$A_{Si\_dup} = 1 – (1 - A_{Si})^2$

$A_{R1} = A_{S1\_dup} * (A_{S2\_dup}*A_{S4\_dup}*A_{S5\_dup}) * (A_{S3\_dup}*A_{S5\_dup}*A_{S6\_dup}) = 0.977$

Resiliency Patterns

Feb 5th, 2024 exam

- Consider the following high-level component structure that is a static description of a possible microservices-based system implementing the business logic for two applications (App A, and App B). The UML Component Diagram below highlights the interfaces (both provided and required ones) of microservices MS1, …, MS6. Moreover, the diagram shows that MS5 writes data on a Network-Attached Storage (NAS) device and MS6 reads and write data on a local cache.



Feb 5th, 2024 exam: Q1

- Based exclusively on the static information presented in the Component Diagram, enumerate all possible execution paths (in terms of sequences of microservices) triggered by requests generated by App A and App B, respectively. Notice that we are interested only in paths that terminate with a microservice that does not require any further interface.
- Then, describe the ripple effect possibly generated by a sudden slowdown of the NAS on each of the identified paths. Assume every request to MS5 causes interactions with the NAS. Identify the application(s) affected by this disruptive event. Justify your answer.

Feb 5th, 2024 exam: Q1

- According to static information only, the possible paths triggered by either App A or App B are as follows.

  - App A -> MS1 -> MS3
  - App A -> MS1 -> MS4 -> MS5 -> MS6
  - App A -> MS1 -> MS4 -> MS6
  - App B -> MS4 -> MS5 -> MS6
  - App B -> MS4 -> MS6

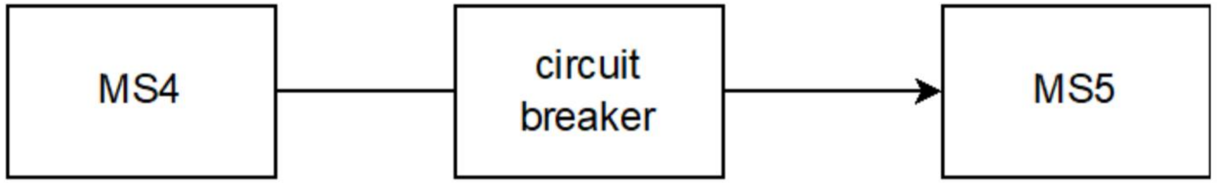
- A sudden slowdown of the NAS could cause a growing number of pending requests to MS5 due to high number of concurrent (slow) operations of the NAS. This could cause a saturation of the resources of MS5. Therefore, pending requests to MS4 also grow potentially saturating its available resources. Essentially the ripple effect follows in a backward manner all the paths listed above that include MS5 or MS4. Thus, according to the paths the effect potentially propagates back to both App A and App B.

Feb 5th, 2024 exam: Q2

- Assume you have the following additional information about the runtime behavior of the system. Microservice MS5 writes data into the NAS and then propagates the same pieces of data to MS6, which writes them on the local cache. Remember that MS5 does not read data from the NAS. All reads are carried out by MS6 using the local cache only.
- Describe how you can use the circuit breaker pattern to mitigate the disruption described above (SA_Q1), taking into account the following constraints:
  - NAS must be avoided during the disruption.
  - All write operations shall eventually take place, either on the NAS or on the local cache, or on both. Do not consider possible data reconciliation issues.
- In your description, explicitly state:
  - how many circuit breakers you would use;
  - where you would place them;
  - their behavior.

Feb 5th, 2024 exam: Q2

- Only one circuit breaker is necessary to avoid interactions with the slow NAS. We can place the circuit breaker in between MS4 and MS5.



- As soon as the number of failed requests grows above the threshold, the circuit breaker goes into state open. At this point the mechanism drops all requests directed to MS5 and runs a fallback procedure that interacts directly with MS6, thus writing the fresh data into the local cache (rather than the NAS). When the circuit breaker changes from half-open to closed, write requests can follow again the nominal flow (… -> MS4 -> MS5 -> MS6).

Feb 5th, 2024 exam: Q3

- Assume now that the expected number of requests per second (workload) for the two applications is as follows:

Application	Workload
App A	100
App B	150

- You can also assume the following behavior when all circuit breakers are closed:
    - Requests directed to MS1 are always distributed evenly among all microservices used by MS1.
    - 25% of the requests to MS4 are write operations and are transferred to MS5, which, in turn, executes them into the NAS and transfers them to MS6 for local cache updates.
    - 75% of the requests to MS4 are read-only operations and are transferred to MS6.
    - The number of requests generated by MS2 is always negligible.

Feb 5th, 2024 exam: Q3

- Address the following points:
  - Determine the number of requests per second that reach each microservice.
  - Use the numbers you have computed to determine the availability of each individual microservice replica according to the following rules: if the total number of requests a microservice receives is less than 60, the availability estimate is 0.99, if the number of requests is between 60 and 80 the estimate is 0.98, 0.97 otherwise. These estimates apply to each of the considered microservices.
  - Focus now on the write operations generated by App B and on the corresponding path. What is the availability shown by App B for write operations when a single replica of each microservice is used?
  - If we want to satisfy the following nonfunctional requirement: "The total availability of App B for write operations, shall be at least 0.999, under the condition that any circuit breaker is closed", what is the required minimum number of replicas for each of the involved microservices?

Feb 5th, 2024 exam: Q3

- When the circuit is closed, given the defined workload, we have the following concurrent requests per each microservice:
  - MS1 = 100 requests
  - MS2 negligible
  - MS3 = 50 requests
  - MS4 = 150 + 50 = 200 requests
  - MS5 = 200 * 0.25 = 50 requests
  - MS6 = 200 * 0.75 + 50 = 200 requests

Feb 5th, 2024 exam: Q3

- Under such conditions, individual availability estimates are as follows:
  - MS1 = 0.97
  - MS2 = 0.99
  - MS3 = 0.99
  - MS4 = 0.97
  - MS5 = 0.99
  - MS6 = 0.97

Feb 5th, 2024 exam: Q3

- The path followed by write operations incoming from App B is MS4 -> MS5 -> MS6, which, for this purpose, are organized in a series. The availability is then: 0.97*0.99*0.97 = 0.93
- To satisfy the non-functional requirement we need to introduce additional replicas for all the three involved microservices. In general, we should satisfy the following constraint:

  $(1 – (1 – 0.97)x) * (1 – (1 – 0.99)y) * (1 – (1 – 0.97)z) > 0.999$
- With x, y, z, number of replicas for MS4, MS5, MS6, respectively.
- Assignments satisfying the previous condition are:
  - x=2, y=3, z=3 and x=3, y=3, z=2 that lead to Avail = 0,9991
  - x=3, y=2, z=3 that leads to Avail = 0,9998
- Instead, instantiating only two replicas per service leads to Avail = 0,9981, while two replicas for MS5 and MS6 or MS4 and three replicas for the remaining service leads to Avail = 0,99897312, which is still slightly lower than 0,999.