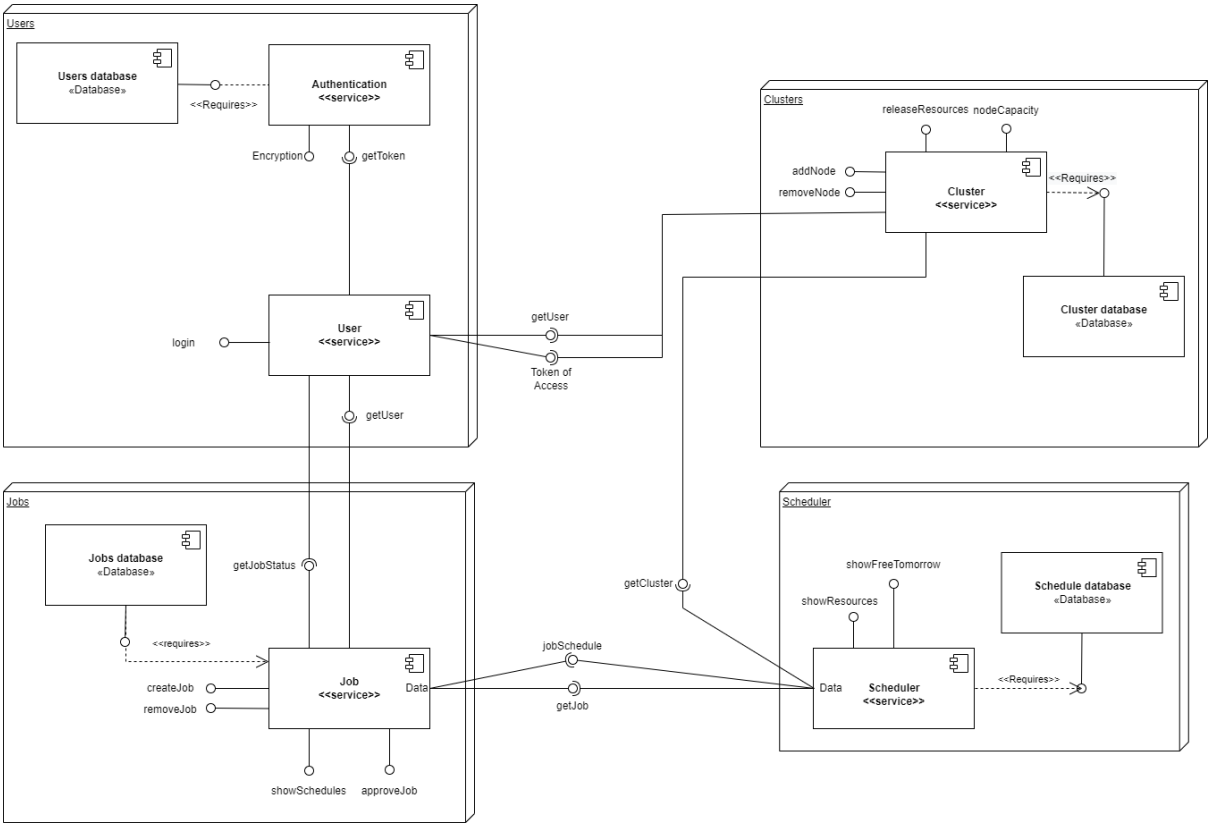
Group-DelftBlue-9b-Assignment-1



UML Component Diagram

Bounded contexts

In the DelftBlue scenario a scheduling system should be implemented that gives university employees the possibility to request resources for their jobs to be done from the DelftBlue supercomputer. This suggests that employees represent one of the subdomains. Each employer is assigned to a faculty and each faculty has a faculty account. So faculties and faculty accounts are also subdomains. Next to the employees and faculty accounts, there are also sysadmins. So users can be distinguished between employees, faculty accounts and sysadmins. Before any of these users can make a request they have to be authenticated, such that authentication is another subdomain. The jobs that need to be done can be classified as a subdomain. Then we also have clusters of nodes as a subdomain and each of the clusters is assigned to one faculty and each faculty will have a cluster of nodes and then we also have a free pool which is another cluster. The main job of the application is the resource allocation so the scheduler is the final subdomain.

The found subdomains are: jobs, users, employees, faculty account, sysadmins, faculty, authentication, scheduler and clusters.

When it comes to the bounded contexts, then employees, faculty account and sysadmins can be summarized as a user. Moreover, authentication will be added to the user context. Clusters and faculty can be merged to one bounded context and jobs will be a bounded context by itself. Thus, the bounded contexts are: Users&Authentication, Jobs, Clusters and the Scheduler.

Microservices

How are bounded contexts mapped into microservices?

We have decided that every bounded context will be a microservice on its own. Bounded contexts were constructed in such a way that every bounded context had the most common functionality. We wanted to keep it that way so we opted for this solution of having four microservice: Users&Authentication, Jobs, Clusters and the Scheduler.

Security and token validation

When a microservice receives an API request with an authorization token, then the token needs to be validated, in order to check if the initial request is accepted. If there is no authorization token, the request will be immediately declined. The validation check will be done by an additional request to the Users&Authentication microservice which will validate the token and send their response. Afterwards, if the token is valid, the request can be processed by the microservice. The token contains the information about the netId and the role of the user. Finally, after the token has been verified, the microservice which handles the request still needs to check if the user with his role can make such a request.

Microservice: Users&Authentication

This microservice is responsible for storing all users in the database. They will be assigned roles that will allow them to interact with methods from other microservices. Their authentication is handled by the Authentication module that is built into this microservice. After providing a correct password, each user receives their private token which will be provided as a parameter to microservices that will determine whether or not perform a requested action.

Users with proper roles will be able to create new jobs with the Jobs microservice, where each job must be assigned to a user. Besides this relationship, users are also allowed to add new nodes to the Nodes microservice, where each node must be assigned to a user. Added nodes are automatically assigned to its creator's faculty. Users will not interact with the Scheduler, as this is the role of the Jobs microservice after a job gets approved.

This microservice was chosen by us, because we identified Users as a bounded context in our scenario. It makes sense for it to be a separate microservice – it is only responsible for storing and authenticating users, which can then interact with other functionality.

Microservice: Clusters

The purpose of the Clusters microservice is to handle and store nodes belonging to faculties, and a free pool. Each node will have a specified amount of resources that can be later used by the Scheduler to schedule the jobs. If the faculty decides to release some nodes from their own pool, they go to the free pool and they can be used for jobs from any

faculty. Cluster stores the information about the number of nodes and the amount of resources a certain node adds. Employee and faculty accounts can add new nodes for their faculty. If an admin adds a new node, resources are split equally between all the faculties.

The Clusters microservice does not interact with other microservices, but provides services to be used by the Scheduler the Users&Authentication. Its role is to maintain a database of nodes and allow scheduling based on that information.

We have decided to choose this microservice because there are many nodes to be managed that belong to certain faculties (where one faculty corresponds to one Cluster). We merge all the functionality of the nodes under one microservice to decrease coupling.

Microservice: Scheduler

The scheduler interacts with Clusters microservice to obtain information required to scheduled jobs. For each job it checks to which faculty it has to be assigned to. Then it takes the cluster of nodes that constitutes the faculty and checks its available resources. If the available resources are enough to suffice the job requirements, the job is assigned to this particular faculty. In the event that the faculty is short of resources, the scheduler checks whether the free pool has enough resources to make up for the shortage and if so, the job is split between the faculty and the free pool. Otherwise, the job is rejected.

The reason behind choosing scheduler as a microservice is that it acts as a logical connection between jobs and clusters. It provides an interface for the Jobs microservice so that it can easily request a job to be scheduled, while not being concerned about the scheduling logic itself. The Scheduler will perform this action and check with the Clusters microservice if there are resources available needed to schedule the job. This isolated the scheduling logic well from the other bounded contexts.

Microservice: Jobs

The jobs microservice stores requests with their name, description, the amount of resources it needs to use and the preferred day to have it run on or before. Every job is made by an Employee (User). This microservice provides an interface for the Users to schedule a job. Once a request for a job to be scheduled arrives, it makes a request to the Scheduler. If the Scheduler is able to allocate resources for the job, it sends back a proper response, which will be used to denote the job as accepted or rejected. This information will later be used by the Users microservice, so that each user can check the status of their jobs.

The reason behind choosing jobs as a microservice is that it isolates the logic of making a request and scheduling it. Moreover, following the Decompose by Subdomain decomposition strategy, it seems like a reasonable choice as we have a subdomain Jobs.

Structure of Code & Database

Every microservice will have a controller that handles incoming requests. Next to the controller there are services, which are used by the controller to execute the actual service of the microservice. For example, the controller of the Schedule microservice will receive a request from the Jobs microservice and after it verifies its access, the controller will call the scheduleService to schedule the job and sends it's response back to the controller, which will then send it's request response back to the Job microservice. This structure is applied to all microservices.

All important information is stored within a database, thus that each microservice will have its own database. User&Authentication database will store all data about the users: netId, password, authorization token and a role(employee, faculty account, admin). Jobs database will store all data about the jobs: netID, jobID, resourceType, CPUusage, GPUusage, MemoryUsage. Clusters database will store all data about the nodes: nodeID, faculty, netID, address, token, CPUprovided, GPUprovided, memoryProvided. Scheduler database will store all data about the scheduled resources: faculty, freeCPU, freeGPU, freeMemory, date.