Virtual Machine Scheduler – Team 14

# Problem Statement

**Build an efficient machine scheduling algorithm to place Virtual Machines on Esxi hosts.**

Scope

**Version 1.**

a. User Management.

1. Define roles like Admin, User and restrict the operations accordingly

2. Ensure data encryption while storing credentials in Database

b. Define Objects (**DataCenter, Cluster, Host (Esxi), and VirtualMachine**) with their relationships and provide RBAC.

**Properties**:

**Datacenter**: Name

**Cluster**: Name

**Host (Esxi)**: Name, ResourceCapacity -> Cpu and Memory

**Virtual Machine**: Name, ResourceCapacity -> Cpu and Memory

c. Decide on Database and design DB Schemas and have ER models for the demo.

d. Define the Scheduling algorithm and provide an option to deploy virtual machine on Cluster and Esxi Host.

**Version 2:**

a. Make sure APIs implemented for all the operations.

b. Consider cases for rebalancing VMs between the hosts in the cluster to accommodate new VMs.

c. UI portal to demo supported operations.

d. Define Test scenarios (Positive and Negative) for the demo.

**Version 3 (Stretch Goal):**

a. Extend the scheduling logic to consider more properties ex: Disk Usage and Network

b. Define initial and maximum capacity for resources ex: should be able to define for CPU-> Min: 1cpu and Max 2cpu and consider the change in scheduling logic

# Requirements Gathering and Planning

* Business Processes and workflows:

o User logs in the system.

o If Admin, then can create, update, and delete datacenter, cluster, ESXi host, and VM. He can see the resources allocated to each of the elements on the screen. If he chooses to deploy a new VM, the scheduling algorithm allocates appropriate ESXi host to it. This allocation is decided on availability of resources like CPU, memory, etc.

o If a regular user, then he can view the details of different elements. He can also perform operations on the VM and deploy a new VM.

* Data and Integration
* Data Migration & Conversion
* Administrative functions:

o If the user has admin privileges, then he is allowed to perform creation, update, and deletion of datacenter, cluster, ESXi, and VM along with deployment of the VM.

o Otherwise, the user has restricted access to the DataCenter, which permits only creation, update, deletion, and deployment of VM and viewing of the properties of the datacenter, cluster, and ESXi host.

* Authentication: Only authenticated users will be allowed in the system.
* Interfaces: REST APIs are used for data transfer.

**Non - Functional Requirements**

* Performance – for example Response Time, Throughput, Utilization, Static Volumetric
* Scalability
* Availability
* Reliability
* Maintainability
* Security
* Manageability
* Data Integrity
* Usability

# Designing and Modelling

## Entity Relationship Diagram:

Below is the ER diagram of the VM Scheduler Project. As depicted, the datacenter contains many clusters, a cluster contains many ESXi Hosts, on which multiple Virtual Machines are deployed. The user entity may be an administrator or local user. and based on the role they can perform operations

A picture containing text

Description automatically generated

*Figure 1 Entity Relationship Diagram*

Use Case Diagram:

The use case diagram depicts the actors and actions they perform. Here, the user of the system logs in, and based on the role type, he is allowed to perform a specific set of actions on the entities. All these actions require the user to login first.

If the user has admin privileges, then he is allowed to perform creation, update, and deletion of datacenter, cluster, ESXi, and VM along with deployment of the VM. Otherwise, the user has restricted access to the DataCenter, which permits only creation, update, deletion, and deployment of VM and viewing of the properties of the datacenter, cluster, and ESXi host.

A close up of text on a whiteboard

Description automatically generated

*Figure 2 Use Case Diagram*

## Database Schema:

A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied to the data.

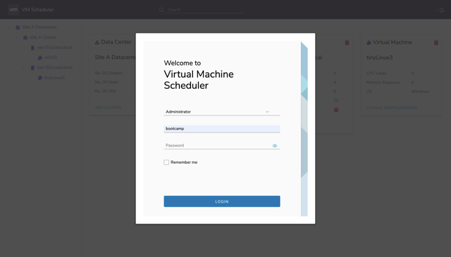
In our project, the entity has ID as its primary key that is used as a foreign key in the entity which references it. As described in the ERD before, the Many-to-One relationship between the entities is shown.

For protecting the password of the users, SHA - 256 is used. Spring security using bcrypt is also implemented. SHA-256 is a novel hash function computed with 32-bit words.

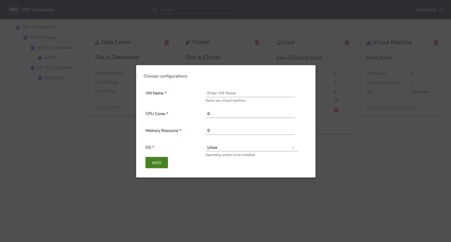
*Figure 3 Class Diagram*

# Implementation Details

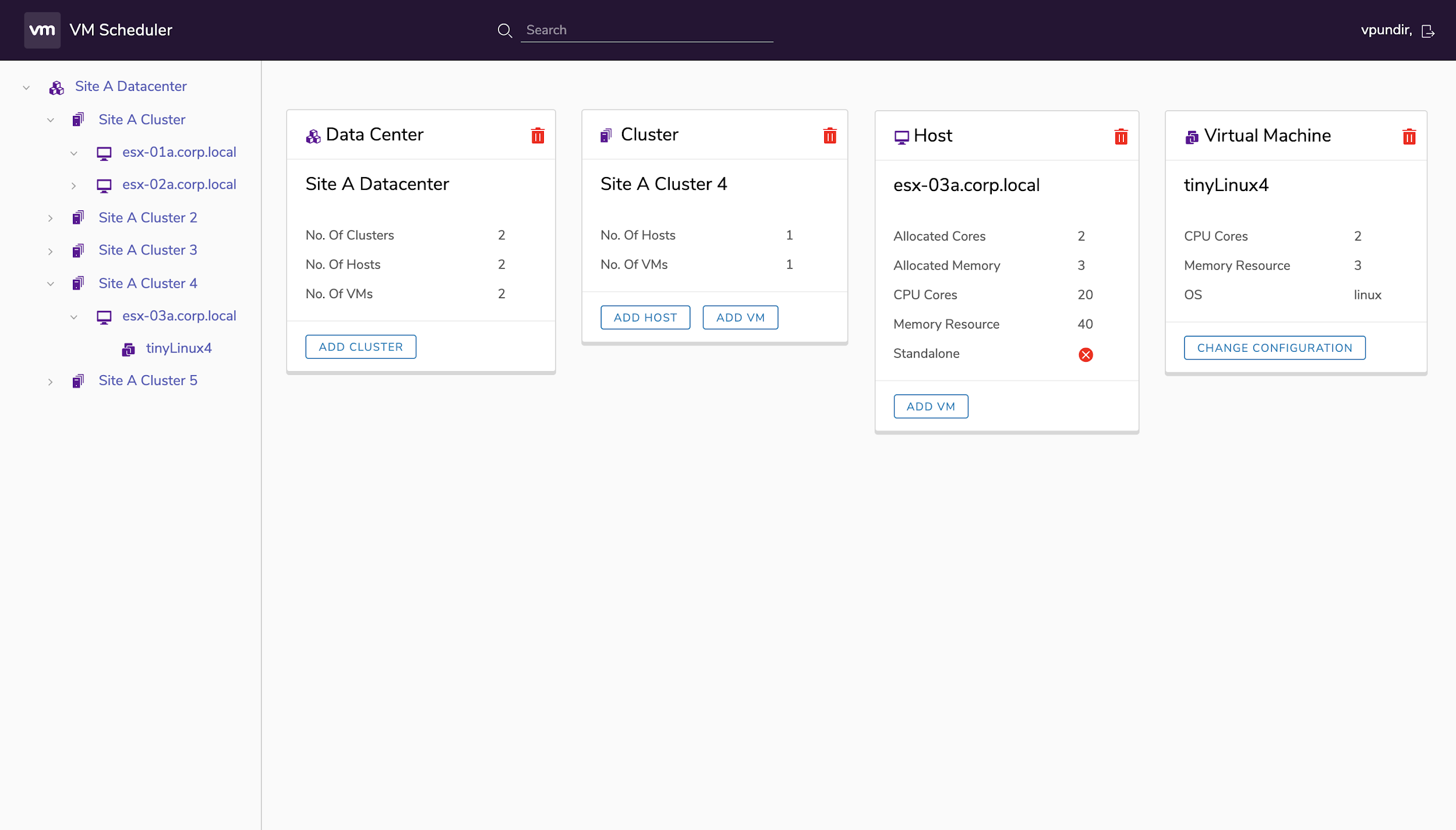
**User Interface**



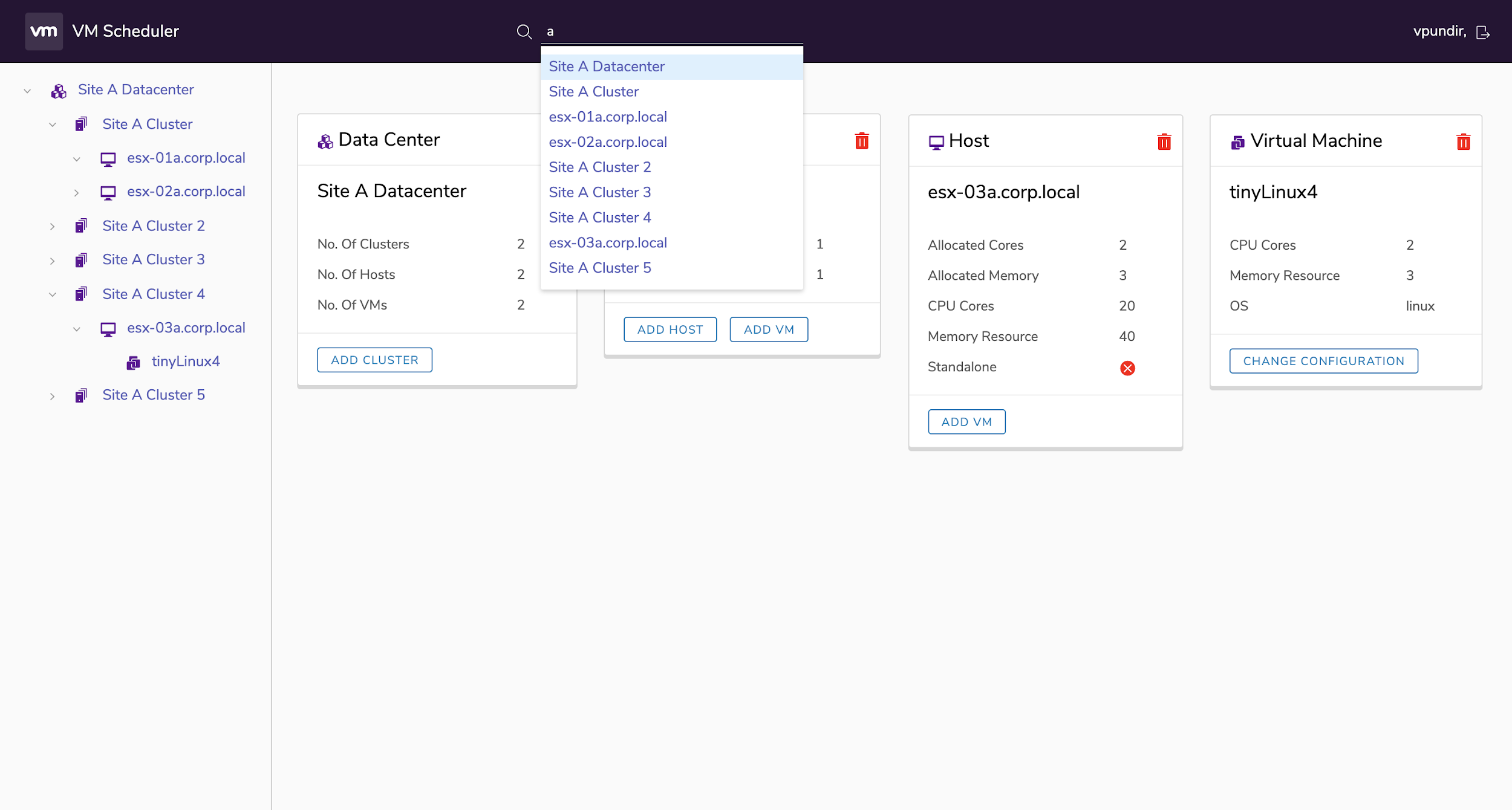
*Figure 4 User Interface 1*



*Figure 5 User Interface 2*



*Figure 6 User Interface showing resource details*

**

*Figure 7 User Interface showing search bar*

**Scheduler Algorithm**

**A close up of a map

Description automatically generated**

Identified the problem as **Bin Packing Problem**, where hosts are bin and VMs are items

Inputs are number of CPU required and amount of memory resource required for the virtual machine. Then all the requirements are normalized according to the min-max normalization scheme and then all such requests are sorted based on the normalized value, which help us in allotting the VMs to each host. We can map the Virtual Machine Scheduler problem into the famous Bin packing problem. The algo can be divided into 2 approaches

1. **Online approach**

In the online version of the scheduling problem, the VM arrives one after another and the (irreversible) decision where to place an item has to be made before knowing the next item or even if there will be another one.

We use First Fit approach to place VMs into host in the online approach.

A screenshot of a cell phone

Description automatically generated

2. **Offline approach**

We are using the FIRST FIT DECREASING algorithm (FFD for short), defined as follows. First, we order the items so that S(x1) >= S(x2) >= S(x3) ……. >= S(xn). We then proceed to pack the items in order, starting with x1, which we place in the first bin x1 . In general, item xi is placed into the first bin that has room for it, i.e., we find the minimum i such that the total size of items currently in xj is no more than 1 - s (xi) and place Xi in Xj.

This algorithm is appealing for both its simplicity and the efficiency with which it can be implemented.

Complexity = O( n logn )

First-Fit is an AF-algorithm that processes the items in a given arbitrary order L. For each item in L, it attempts to place the item in the first bin that can accommodate the item. If no bin is found, it opens a new bin and puts the item within the new bin.

First Fit Decreasing Algorithm works analogous to First-Fit. However, before starting to place the items, they are sorted in non-increasing order of their sizes. this algorithm can be implemented to have a running time of at most O(nlog(n))

**Approximation Guarantee**

FFD(I) <= (11/9) (OPT(I)) + (6/9)

For a given list of items I the number A(I) denotes the number of bins used when algorithm A is applied to list I, while OPT(I) denotes the optimum number for this list.



**Advantage**

Using First Fit Decreasing instead of First Fit, results in reducing the number of hosts required for the same number of VMs and with same requirements.

As shown in the example, in offline mode lesser number of hosts are required to accommodate the same VMs.

**A screenshot of a cell phone

Description automatically generated**A screenshot of a cell phone

Description automatically generated

**API Calls Details**

File – UserController.java

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **API Call** | **Parameters Passed** | **Return** |
| 1 | getLeftPanSummary | None | String |
| 2 | datacenterById | Int - datacenterID | DataCenter – returns Datacenter object of the given datacenterID |
| 3 | clustersInDatacenter | Int – datacenterID | List - of Cluster objects of that Datacenter |
| 4 | Aggregation | Int – datacenterID | Summary - |
| 5 | getAllDatacenter | None | List – of all Datacenter objects |
| 6 | clusterByID | Int – clusterID | Cluster – returns cluster object of the given clusterID |
| 7 | hostsinCluster | Int – clusterID | List – of hosts belonging to the given cluster |
| 8 | clusterAggregation | Int – clusterID | Summary - |
| 9 | getAllCluster | None | List – of all clusters |
| 10 | hostByID | Int – hostID | Host – return Host object of the given hostID |
| 11 | vmsInHost | Int – hostID | List – of all VM objects belonging to the host |
| 12 | vmByID | Int – vmID | VM – returns VM object of given vmID |

File – AdminController.java

|  |  |  |  |
| --- | --- | --- | --- |
| **S.  No.** | **API Call** | **Parameters Passed** | **Return** |
| 1 | addNewUser | User - user | User – returns newly added user object |
| 2 | userById | Int – userID | User – returns object for the given userID |
| 3 | createDatacenter | Datacenter – datacenter | Datacenter – returns created Datacenter object |
| 4 | updateDatacenter | Datacenter – datacenter | Datacenter – returns updated Datacenter object |
| 5 | deleteDatacenterByID | Int – datacenterID | Int – returns number of clusters deleted under the given Datacenter |
| 6 | createCluster | Cluster – cluster | Cluster – returns created Cluster object |
| 7 | updateCluster | Cluster – cluster | Cluster – returns updated Cluster object |
| 8 | deleteClusterbyID | Int – clusterID | Int – returns number of hosts deleted under the cluster |
| 9 | createHost | Host – host | Host – returns created Host object |
| 10 | updateHost | Host – host | Host – returns updated Host object |
| 11 | deleteHostByID | Int – hostID | Int – returns number of VMs deleted under the host |
| 12 | deleteVmByID | Int – vmID | Void – returns nothing, deletes the VM for given vmID |
| 13 | createVM | VirtualMachine – vm | ResponseEntity <VirtualMachine> – throws “Resource not available” exception, if the resources requested are not available, otherwise returns status “ok” |
| 14 | createVmAtCluster | VirtualMachine – vm  Int – clusterID | ResponseEntity <VirtualMachine> - |
| 15 | updateVM | VirtualMachine – vm | VirtualMachine – returns updated VirtualMachine object |

# Future Work

If an existing VM’s requirement is increased and if the VM’s requirement cannot be handled by the existing host in which the VM resides then we will be using the scheduling algorithm to rearrange to check if the VM can be accommodated in a different host.