


## Article

# Bespoke Virtual Machine Orchestrator: An Approach for Constructing and Reconfiguring Bespoke Virtual Machine in Private Cloud Environment

Joonseok Park <sup>1</sup>, Sumin Jeong <sup>2</sup>  and Keunhyuk Yeom <sup>3,\*</sup>

<sup>1</sup> Research Institute of Intelligent Logistics Big Data, Pusan National University, Busan 43241, Republic of Korea; pjs50@pusan.ac.kr

<sup>2</sup> Department of Information Convergence Engineering, Pusan National University, Busan 43241, Republic of Korea; sumin2708@gmail.com

<sup>3</sup> School of Computer Science and Engineering, Pusan National University, Busan 43241, Republic of Korea

\* Correspondence: yeom@pusan.ac.kr; Tel.: +82-51-510-2475

**Abstract:** A cloud-computing company or user must create a virtual machine to build and operate a cloud environment. With the growth of cloud computing, it is necessary to build virtual machines that reflect the needs of both companies and users. In this study, we propose a bespoke virtual machine orchestrator (BVMO) as a method for constructing a virtual machine. The BVMO builds resource volumes as core assets to meet user requirements and builds virtual machines by reusing and combining these resource volumes. This can increase the reusability and flexibility of virtual-machine construction. A case study was conducted to build a virtual machine by applying the proposed BVMO to an actual OpenStack cloud platform, and it was confirmed that the construction time of the virtual machine was reduced compared with that of the existing method.

**Keywords:** cloud computing; private cloud; virtual machine; cloud platform



**Citation:** Park, J.; Jeong, S.; Yeom, K. Bespoke Virtual Machine Orchestrator: An Approach for Constructing and Reconfiguring Bespoke Virtual Machine in Private Cloud Environment. *Appl. Sci.* **2023**, *13*, 9161. <https://doi.org/10.3390/app13169161>

Academic Editor: Rong Gu

Received: 10 June 2023

Revised: 26 July 2023

Accepted: 8 August 2023

Published: 11 August 2023



**Copyright:** © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Cloud computing [1–3] provides a service environment in which IT resources can be virtualized and used. This computing environment has evolved with the advent of public cloud services [4] such as AWS [5] and Google Cloud [6] and the emergence of open-source platforms that provide private cloud service construction, such as OpenStack [7] and CloudStack [8]. In the case of a public cloud, the user pays a certain amount and receives a prebuilt cloud service. A private cloud [9] is used by a company or user to directly build and use a cloud-service-provision environment. “A Hybrid cloud”, which refers to the use of a public cloud and a private cloud together [10], is also provided.

In the case of public cloud-computing services, companies or users receive services in the form provided by cloud vendors and pay according to service usage. A private cloud environment must be established to reduce the cost of cloud computing and build a cloud-service environment in the form desired by companies or users.

In the case of private clouds, dependence on cloud vendors can be reduced; however, it is difficult to build a desired cloud-service-delivery environment without expert knowledge of the cloud. The most basic technical application for building and receiving cloud services is building a virtual machine. Enterprises or users who build private clouds have different purposes for using the cloud and have different requirements for different operating environments. Therefore, it is important to provide a customized virtual-machine environment suitable for various companies or users who wish to operate a private cloud [11–15].

Recently, infrastructure as code (IaC) [16] has emerged as a concept for automating and providing virtual-machine provision and cloud-infrastructure construction. Tools such as Ansible [17] and Cloud-Init [18], which are used to automate virtual-machine construction

as support tools that reflect the concept of IaC, are being researched and developed [19]. In this study, we propose a method for automating the creation and management of custom virtual machines by borrowing the concept of IaC. In addition, to ensure reusability and flexibility in building user virtual machines, we applied the concept of software product lines to build resource volumes used for virtual machines as core assets and presented an approach to utilize them. We proposed a bespoke virtual machine orchestrator (BVMO) to build a user-customized virtual machine by applying these concepts.

The contributions of the BVMO proposed in this study are as follows: (1) providing a foundation to support user-customized virtual machine construction and management in a multi-private cloud environment by combining volumes provided from provisioning resource pools; (2) establishing a reuse virtuous cycle system that enables new reuse by re-registering virtual-machine resources created by users for the provisioning resource pool; and (3) solving technical difficulties in building virtual machines by applying BVMO without using command line interface (CLI) or application programming interface (API).

The remainder of this paper is organized as follows. Section 2 introduces previous studies related to the creation and management of virtual machines. Section 3 describes the architecture, process and application of BVMO. Section 4 presents a case study and an evaluation of the process of creating a custom virtual machine using BVMO, and Section 5 concludes the paper.

## 2. Related Works

### 2.1. Private Cloud Platform

OpenStack [7] and CloudStack [8] are representative open-source platforms that support private cloud computing. OpenStack has numerous components that can provide major cloud functions such as computing, networking, storage and identity. In addition, a cloud-orchestration component called heat was provided to support virtual-machine creation and cloud-environment construction using a heat-orchestration template. Similar to OpenStack, CloudStack is an open-source cloud platform that supports deployment and management of cloud-service resources. This supports various hypervisors [20], such as Hyper-V, XenServer and vSphere. Most modules in CloudStack are bundled into a single binary, whereas OpenStack can build a cloud by combining multiple components.

### 2.2. Software-Product-Line Engineering

Among software development methodologies, product-line engineering [21] involves developing a core asset, a unit to be reused in developing a similar software-product family, in advance. Assets used for commonality (required) and assets used for variability (optional) were identified in advance and reused to develop new software products flexibly.

Even in a cloud environment, there is a concept of resource provisioning [22], in which various cloud resources existing on a cloud platform are built, prepared and utilized in advance before being combined. This can be linked to the product-line-engineering approach. Therefore, in this study, we propose a method that combines product-line engineering with cloud-resource provisioning for virtual-machine creation in a cloud environment. In this method, a volume pool that can be combined to build a virtual machine is built in advance as a core asset (resource provisioning) and a new virtual machine is built by binding these assets.

### 2.3. Provisioning Cloud Resources

Rajan et al. [23] proposed a method for distributing cloud resources by applying a map-reduce-merge method based on cloud workloads. After the cloud resources were classified into task units and the divided units were identified as one workload, the map-reduce-merge technique was applied to allocate the workload to a work queue.

By contrast, in the proposed approach, apart from separating cloud resources into workload units and then dividing and conquering them, common and reusable cloud resources were identified in advance and built. The difference is that a virtual machine is

created by combining prebuilt cloud resources according to the requirements, similar to a building with Lego blocks.

Wei et al. [24] formalized the decision system in the configuration of a cloud-platform-management system according to the workflow analysis of the cloud. This method focuses on configuring a system from a management viewpoint rather than configuring a customized system by reflecting requirements. This involves merging instances by analyzing the quality of service (QoS). Therefore, this study presents a case in which the QoS is considered in the process of combination. The proposed method differs from the method that considers simple QoS in the part where combining is performed according to requirements and merging is performed by reusing pre-established resources.

Nadeem et al. [25] recommended the proactive resource provisioning of service level agreement (SLA) in cloud computing (PRP-RM-SLA), which can provide cloud-computing resources based on SLA. This method operates by delivering the task of provisioning resources to the cloud based on the SLA manager, which interprets and negotiates SLA requests from cloud users, and the resource manager, which manages the resources based on the SLA manager. However, this study does not suggest systemic elements that reflect multi-clouds, and it is different from our method because no support is available for the components constituting the resources.

#### 2.4. Cloud-Service Brokerage

Alwada'n et al. [26] presented a dynamic congestion-management system that provides and recommends services suitable for cloud consumers according to cloud size, classified as IaaS, PaaS or SaaS. The proposed system configures the service and transmits it to the user by performing scheduled provisioning requests based on the situation in the data center. The system receives inputs by prioritizing the user requests. However, this study differs from that of Alwada'n et al. in that it does not describe how the provisioning method is performed but only presents a scheduling methodology.

Li et al. [27] presented a management method that used online scheduling for cloud services provided by public clouds. This method classifies virtual machines (VMs) provided as containers into large, medium and small based on their size and proposes a method for provisioning and management by determining the suitability of resource preparation according to each size. However, this study presents an intensive scheduling technique for containers, which is another virtualization method. Moreover, VM-level resources, which constitute a large category in current cloud computing, are ignored, which differs from the results of this study.

#### 2.5. Infrastructure as Code

Terraform [28] is an IaC tool written to create and manage resources for multi-cloud components. This corresponds to tools that perform infrastructure deployment and management in addition to the Hashicorp Software Stack [29], a software stack that manages and controls multi-cloud platforms. The Terraform Engine, which is mapped to each cloud platform, interprets it based on a common template, converts it into commands or a representational state transfer API (RESTful API), and delivers the platform commands.

Heat [30] is an OpenStack-specific IaC tool that was developed to control the components and resources of OpenStack, an open-source private cloud platform. It is used to control and manage various components that exist in OpenStack and performs operations in the form of transmitting commands to each component of OpenStack by interpreting templates created by users in the Heat Engine.

IaC tools are commonly used as software for deploying infrastructure elements. However, the BVMO presented in this study is different in that it increases reusability and utilization by focusing on VM resources.

#### 2.6. Discussion

Table 1 compares the BVMO proposed in this study with those of related studies.

**Table 1.** Literature Review of Related Studies.

Study	Resource Provisioning Domain?	Resource Reuse?	Methodology System Architecture?	Multi-Cloud System Support?	Customized VM?
Rajan et al. [23]	✓		✓		
Wei et al. [24]	✓		✓		✓
Nadeem et al. [25]	✓	✓	✓		
Alwada'n et al. [26]			✓	✓	
Li et al. [27]		✓			✓
Our method	✓	✓	✓	✓	✓

The categories adopted for comparison listed in Table 1 are as follows: (1) whether the resource-provisioning domain is introduced and utilized in the study; (2) whether there is a method or mechanism for reusing cloud resources; (3) whether there is a description of the technique through the system architecture; (4) whether there is a plan for multi-cloud support; and (5) support measures for providing VMs tailored to user needs.

Table 2 presents the comparison of the IaC tools, cloud-platform-management tools and BVMO.

**Table 2.** Review of Related Technologies.

Software Set	Specific Software	Software Category	Platform Boundary	Implement Method	Coverage
Hashicorp Software Stack [29]	Terraform [28]	IaC	Multi cloud domain where automation engine exists	Template	Cloud platform elements deployment
	Vault [31]	Security		Template	Cloud platform authority
	Consul [32]	Networking System		Template	Cloud platform networking strategy
	Nomad [33]	Application		Template	Cloud platform workload automation
OpenStack Components	Heat [30]	IaC	OpenStack cloud platform	Template	OpenStack platform elements management
	Keystone [34]	Authentication		API, CLI	OpenStack platform authority and authentication
	Neutron [35]	Networking		Dashboard, API, CLI	OpenStack platform virtual network strategy
	Nova [36]	Instantiation		Dashboard, API, CLI	OpenStack platform instance management
Our Study	BVMO	VM resource orchestration based on IaC	Multi-cloud platform where IaC can be applied	Dashboard	Cloud platform instance management for more reusability and flexibility

As listed in Table 2, the subjects of comparison with the BVMO presented in this study are each components of OpenStack, which is used as an open-source private cloud platform, and the main element of the Hasicorp Software Stack, which performs automation and distributed control of multi-cloud. The vault of the Hashicorp Software Stack performs security related to authentication processing scattered across multiple clouds, Consul controls cloud networking resources through service discovery and Nomad performs service utilization and control through a Remote Procedure Call (RPC). In this stack, the platform deployment and execution of each software package are implemented as templates based on Terraform.

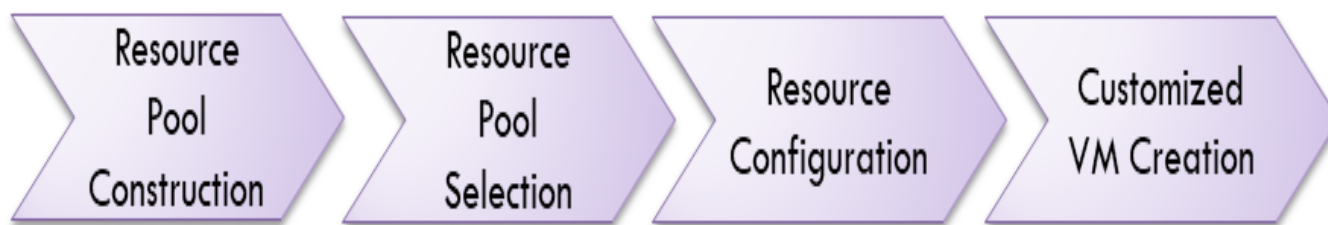
Additionally, on the OpenStack Components side, Keystone creates and assigns permission to access the platform, configures and controls instance networks through Neutron, and creates and controls cloud instances through Nova. The distribution and flow control of these components can be achieved using heat templates.

In contrast, the BVMO in this study is a system that creates and reconfigures VMs by combining cloud resources extracted from multiple private clouds using IaC and supports deployment to specific locations in multiple clouds. In addition, the various tools compared above require prior knowledge of APIs or templates rather than system control through a dashboard for smooth use. However, because the BVMO presented in this study provides the basic flow of combining pre-established resources with the user's choice, compared to other tools, the BVMO is technically accessible and simple to use.

### 3. Bespoke Virtual Machine Orchestrator

#### 3.1. Conceptual Process for Bespoke Virtual Machine Orchestrator

The BVMO allows the creation and management of user-customized virtual machines in a private cloud environment. Figure 1 shows the conceptual steps for building a virtual environment by applying the proposed provisioning resource pool.



**Figure 1.** Conceptual Process of BVMO.

First, in the resource-pool-construction step, provisioning resources are divided into operating system, software and data-resource items. Operating system resources include CentOS and Ubuntu, which operate the virtual machines. Software resources refer to various software programs, such as GCC [37], Java [38] and Python [39], that users require in a virtual environment. Data resources represent various structured and unstructured data, such as text, documents, images and videos.

Next, in the resource-pool-selection step, the user selects the desired resources from a prebuilt resource pool. Because prebuilt resources are provided, users can easily combine the desired components of a virtual environment through selection.

The setting and combination of the selected resource items are performed in the resource-configuration step.

Finally, in the customized-VM-creation step, a user-customized virtual machine reflecting the settings and combinations is created. Figure 2 shows the application of the process described above.

As shown in Figure 2, the proposed method creates various provisioning resource pools (data and software) and provides flexibility in building a virtual environment by combining resource pools that satisfy the needs of each user.

#### 3.2. Analysis of Cloud Resource Features and Cloud Functional Flow

Currently used public and private cloud platforms configure and provide independent virtual machines; therefore, virtual machines are configured based on platform-dependent virtual-machine specifications [40–44]. Table 3 lists the classifications of common specifications for supporting various virtual machines.



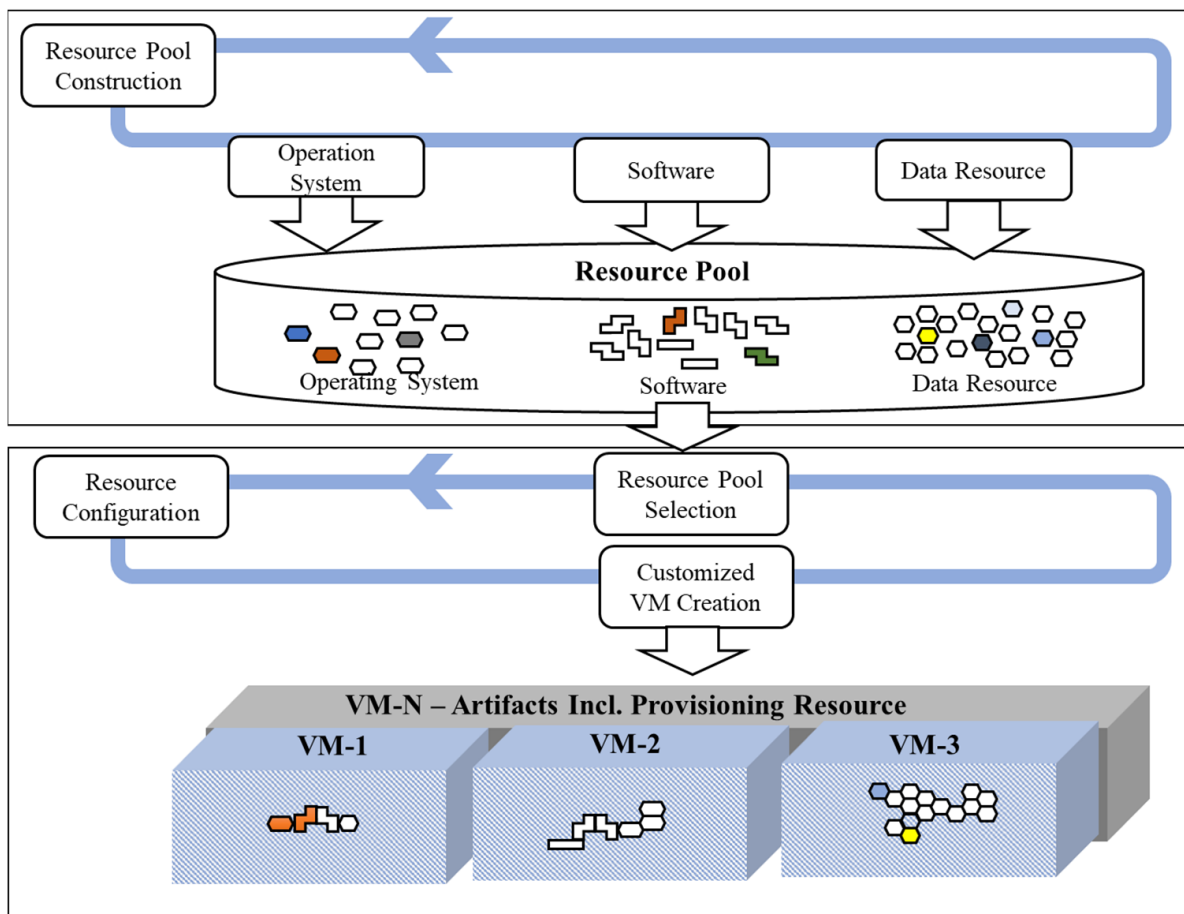


Figure 2. Conceptual Application of BVMO.

Table 3. Common Specifications of Cloud Platforms.

Category	Google Cloud Platform	Amazon Web Service	Microsoft Azure	OpenStack	CloudStack
VM specification	Name	Virtual machine name	Name	Instance name	Instance name
	Instance type	Machine configuration	Size	Flavor	Instance type
Resource specification	Bootimg disk	Application and OS image	Image	Image	Template
	Disk	Storage config	Disk	Volume	Volume
Interface specification	Network	VPC	Virtual network	Network	Network
	Network tag	Security group	Inbound port rule	Security group	Security group

Among the categories listed in Table 3, the VM specifications contain the metadata required to specify the VM and the resource-size information required to drive the VM. Resource specifications include the information necessary for operating software, such as an OS, and the details of the storage device on which the software is loaded. The interface specification contains the network information and policy through which the VM communicates.

The aforementioned specifications are interpreted according to the schedule flow of the cloud platform and reflected in the platform. The basic feature request flows of OpenStack and CloudStack are as follows.

For OpenStack:

1. Issuance of tokens (temporal data for authority execution) according to user identification;
2. Authorization to compare issued tokens;
3. API specification and request that the user wants to perform;
4. Execution of functions within the platform of the specified API.

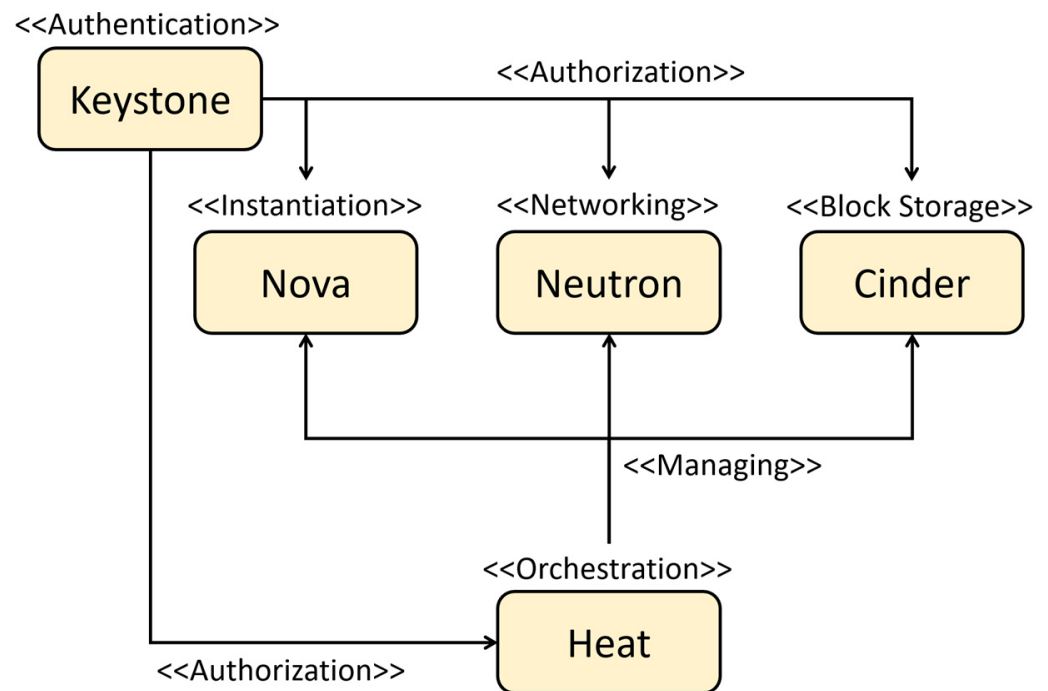
For CloudStack:

1. Create a secret key that combines the API key existing in the user's identification and the request function;
2. Authorization of secret key;
3. Request for functions that the user wants to perform;
4. Execution of functions within the platform of the specified API.

From the flow above, a common execution flow can be extracted as follows.

1. Authority information issued based on user-identification information;
2. Authorization based on authority information;
3. Execute request with authority information based on request interface;
4. Performance and processing of API functions suitable for the purpose.

Figure 3 shows the dependencies of the basic components, including the IaC of the OpenStack platform, which is currently used as a private cloud.

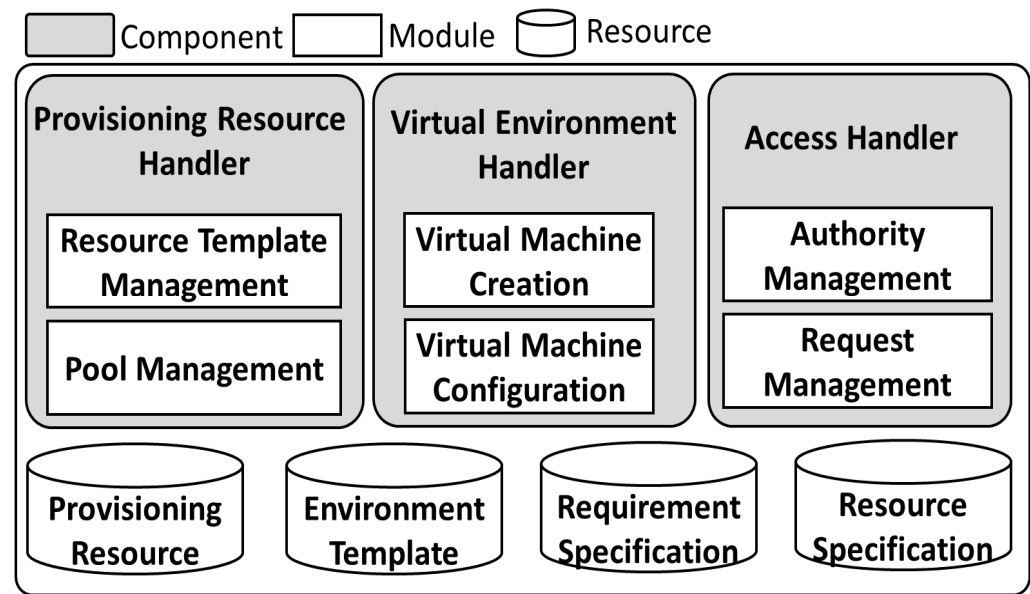


**Figure 3.** OpenStack-Component Functional-Dependency Diagram.

The components shown in Figure 3 consist of a keystone that implements authentication and authorization; Nova, Neutron and Cinder, which perform direct resource management such as instance, network and block storage in the cloud platform; and Heat, an IaC tool that supports management.

### 3.3. Architecture of Bespoke Virtual Machine Orchestrator

A virtual-environment-construction architecture using provisioning resources is employed to realize the construction process presented in Section 3.1, as shown in Figure 4.



**Figure 4.** Architecture.

Table 4 describes the main components of the architecture presented in Figure 4.

**Table 4.** Main Component in Architecture.

Name	Description
Provisioning-Resource Handler	A component that manages resources scattered in the cloud environment to create a virtual environment
Virtual-Environment Handler	A component that builds, reconfigures and manages virtual environments
Access Handler	A component that handles resource-related requests for user authentication, resource-access-permission setting and requirements

The main modules of each component implement the following roles:

- **Resource Template Management:** this manages templates that specify the provisioning of resource-binding information that reflects user requirements;
- **Pool Management:** this supports creation, reading, updating and deletion (CRUD) of provisioned resources and the allocation of provisioned resources according to interpretation of user requirements;
- **Virtual Machine Creation:** this interprets template information and creates a virtual environment in which the provisioning resources are bound;
- **Virtual Machine Configuration:** this supports the creation, reconfiguration and assembly of orchestration templates to combine provisioned resources;
- **Authority Management:** this authenticates users and provides access to provisioning resources;
- **Request Management:** this invokes the application-programming interface (API) of the cloud platform that performs provisioning resource integration and requests resource combinations.

Table 5 lists the resources used in the architecture.

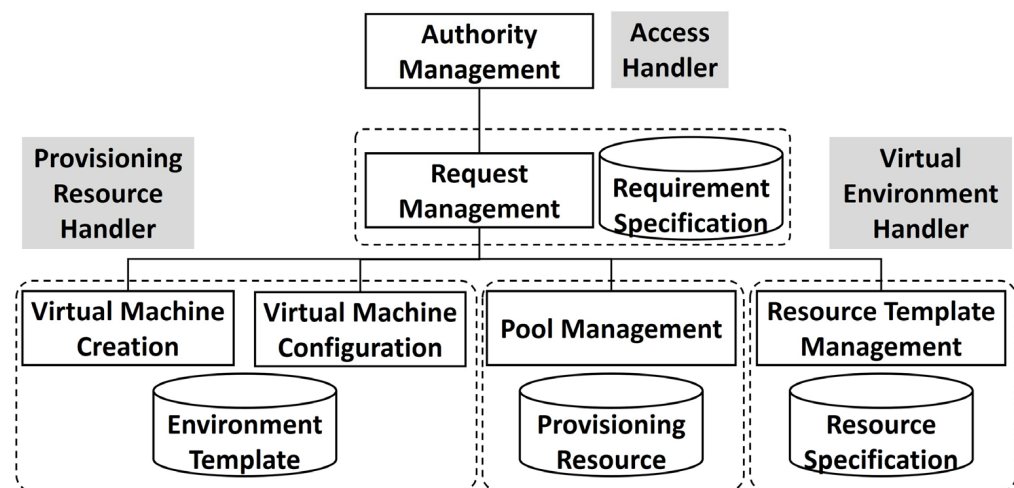


**Table 5.** Resource Information.

Name	Description
Provisioning Resources	Pre-created provisioning resource pools on multiple target platforms
Environment Template	A template pool that specifies the settings of various virtual environments currently being utilized
Requirement Specification	User requirements pool
Resource Specification	Detailed specifications of the provisioned resource pool (available amount, maximum usage, etc.)

As presented in the architecture, the provisioning-resource-handler component manages the provisioning resources and resource templates. The delivery of provisioned resources and API calls are delivered by an access-handler component. The virtual-environment-handler component combines resource items and creates templates based on them. In addition, it creates, separates and combines provisioned resources using templates. Therefore, a virtual environment reflecting these resources was built. The corresponding application scenarios are described in Section 3.4.

Figure 5 shows the inclusion relationship based on the functional flow analyzed in Section 3.2 for the function and related resources of the proposed architecture.

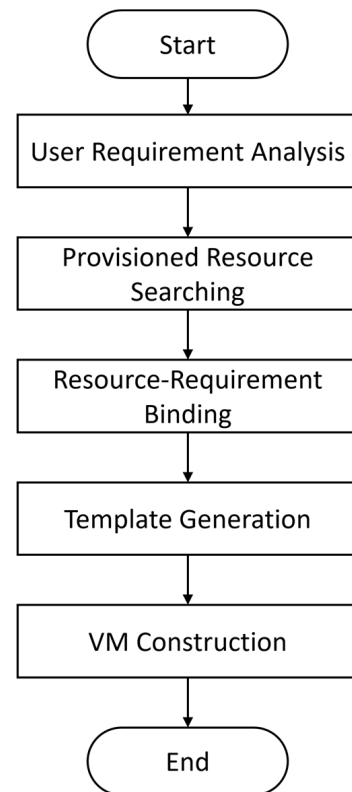
**Figure 5.** Architecture Hierarchy.

The hierarchy shown in Figure 5 is based on the dependencies of the OpenStack components shown in Figure 3. As shown in Figure 5, the authority management of the access handler is executed first. After the authority is normally executed, resource pool management, resource template management, VM creation and VM management are performed according to the given functions in the request management.

### 3.4. Application Scenario for BVMO

The BVMO allows the creation and management of user-customized virtual machines in a private cloud environment. Figure 6 shows an application scenario involving a virtual-machine system based on the architecture presented in Section 3.3, in which a virtual environment is built by applying the provisioning resource pool proposed in this study. When requesting the creation of a virtual environment, the user requirements were collected and analyzed. Subsequently, provisioning resources suitable for the requirements were delivered. The virtual-environment-creation management system combines resource items by reflecting requirements (selected resource information). It also creates a template that

describes the resource-combination information that can be interpreted and executed on a cloud platform. The created template is used to combine the necessary resources, and a user-customized virtual environment is created by applying additional information such as the network settings required when creating a virtual environment.



**Figure 6.** Application Scenario.

## 4. Case Study and Evaluation

### 4.1. Case Study

In this section, a case study that implements the proposed method is presented. This case study presents the process of building a virtual machine that conducts web classes on a cloud platform based on a BVMO dashboard.

The environment and software applied to the BVMO development are presented as a dashboard system, as in Table 6, and OpenStack was used as the target platform to which the VM was applied.

**Table 6.** Case Study Environment.

Label		Requirement
Dashboard System	OS	Windows 10
	Software	Node.JS v.18.16.1
	Hardware	CPU: 8 core RAM: 16 GB Disk: 1.8 TB
OpenStack Platform	OS	Ubuntu 20.04 LTS
	Software	OpenStack Xena, Terraform Engine v.1.5.3
	Hardware	CPU: 4 core RAM: 8 GB Disk: 80 GB

## (1) BVMO dashboard

Figure 7 shows the BVMO dashboard implemented by applying the concept presented in this study.

The screenshot displays the BVMO Manager dashboard with a sidebar on the left containing navigation links: CORE, Dashboard, ADDONS, and BVMO Manager. The main content area is divided into four sections, each highlighted with a red border and a numbered label:

- 1. Exist Virtual Machine View:** A table titled 'Virtual Machine List' with columns: Name, Id, Platform, Region, Owner, and Detail. It lists three VMs (VM3, VM2, VM1) with their respective IDs and details. A 'Delete VM' button is present.
- 2. Exist Virtual Machine Resource View:** A tree view titled 'VM Resource View' showing the configuration of a selected VM. It includes sections for OS (Image, Type), Flavor (VCpu, Ram, Disk), Network (Type, Zone, Subnet, IP Address Range, IP Version), and KeyPair (set to None). Buttons for 'Initialize' and 'Create VM' are at the top right.
- 3. Provisioning Resource Pool View:** A section titled 'Provisioning Resource' with a 'Resource Type' dropdown set to 'Empty'. Below is a table with columns: Volume Name, Size, and Select. It lists 'default Vol 2' and 'default Vol 1'.
- 4. Selected Resources View:** A section titled 'Selected Resource' with a table showing selected resources. It includes sections for 'Selected VM Resource', 'Selected Provisioning Resource', and 'Selected KeyPair'.

At the bottom left, it says 'Created By: Pusan National University Software Engineering Lab.'

Figure 7. BVMO Dashboard.

The dashboard is divided into four views. No. 1 'Exist Virtual Machine View' is the basic information of the previously created VM, providing information to check the VM's deployed ID and deployment location. No. 2 'Exist Virtual Machine Resource View' can check resources specified by the selected VM, and representative information of VM configuration, such as VM hardware information and OS, is displayed. In No. 3, 'Provisioning Resource Pool View' can check the pre-established resources that exist in the provisioning resource pool presented in this study. Number 4 'Selected Resources View' is for the final confirmation before providing the resources selected in views No. 2 and No. 3 as templates.

Each resource type in view No. 3 is enlarged as shown in Figure 8.

As shown in Figure 8, the resource type was created by dividing it into empty, which provides empty storage space; OS, which provides an operating system; software, which provides software against the operating system; and data resources, which provide data for specific purposes. Because data should be provided most fundamentally in the software stack constituting the service, a provisioning pool was built by classifying them as above.

## (2) A case study on building a virtual machine for web classes using BVMO

Figure 9 shows the results of selecting provisioning resources and VM specifications when building a virtual machine for web classes.

Provisioning Resource				
Resource Type: <input type="text" value="All"/>				
Volume Name	Size	Select		
default Vol 2	1	<input type="checkbox"/>		
default Vol 1	1	<input type="checkbox"/>		
Volume Name	Size	OS Type	Version	Select
Ubuntu20.04 Vol	1	Ubuntu	20.04	<input type="checkbox"/>
CentOS7 Vol	1	CentOS	7	<input type="checkbox"/>
Volume Name	Size	Installed software	Select	
MySQL-CentOS Vol	1	Mysql 5.7.0	<input type="checkbox"/>	
Nodejs-CentOS Vol	1	NodeJS 20.04	<input type="checkbox"/>	
MySQL-Ubuntu Vol	1	Mysql 5.7.0	<input type="checkbox"/>	
Nodejs-Ubuntu Vol	1	NodeJS 20.04	<input type="checkbox"/>	
Volume Name	Size	Contents	Select	
DB Class Video Vol	3	db class video	<input type="checkbox"/>	
DB Class Data Vol	1	db class data	<input type="checkbox"/>	
Web Class Video Vol	3	web class video	<input type="checkbox"/>	
Web Class Data Vol	1	web class data	<input type="checkbox"/>	

- Empty

- OS

- Software

- Data

Figure 8. Provisioning Resource Pool View.

**Virtual Machine List**

Name	Id	Platform	Region	Owner	Detail
VM3	8a4b727f-9a81-400d-86e8-ba0d076f0539	OpenStack	RegionOne	admin	<a href="#">Show</a>
VM2	fee73a05-00da-4bb9-8a7f-7c1970e39032	OpenStack	RegionOne	admin	<a href="#">Show</a>
VM1	d232fbc6-a04e-45ab-babc-0b6be7fd47d3	OpenStack	RegionOne	admin	<a href="#">Show</a>

**VM Resource View**

- OS
  - ☒ Image: ubuntu 20.04
  - ☐ Type: qcow2
- Flavor
  - ☒ Vcpu: 2
  - ☒ Ram: 2048
  - ☒ Disk: 10
- Network
  - ☒ Type: RegionOne
  - ☐ Zone: flat
  - Subnet
    - ☒ IP Address Range: 172.24.4.0/24
    - ☐ IP\_Version: 4

Keypair:

**Provisioning Resource**

Resource Type:

Volume Name	Size	Contents	Select
DB Class Video Vol	3	db class video	<input type="checkbox"/>
DB Class Data Vol	1	db class data	<input type="checkbox"/>
Web Class Video Vol	3	web class video	<input checked="" type="checkbox"/>
Web Class Data Vol	1	web class data	<input checked="" type="checkbox"/>

**Selected Resource**

Category	Selected
Selected VM Resource	Image: ubuntu 20.04 Ram: 2048 Vcpu: 2 Disk: 10 IP Address Range: 172.24.4.0/24 Type: RegionOne
Selected Provisioning Resource	> Nodejs Vol > Web Class Video Vol > Web Class Data Vol
Selected KeyPair	

**Virtual Machine Specification Mapping**

**Provisioning Resource Mapping**

Figure 9. Web-Class Case—Selected Resources.

As shown in Figure 9, if the selected virtual machine specifications and provisioning resources are selected, they are reflected in the selected resource view, and by applying these details, BVMO internally creates a terraform template for IaC utilization. Figure 10 shows the created terraform template.

```

1  [
2    "resource": {
3      "openstack_compute_volume_attach_v2": {
4        "volume_attach_1": {
5          "instance_id": "${openstack_compute_instance_v2.instance.id}",
6          "volume_id": "35cac393-0dbe-4ce5-86bc-7b3b6a53bb0e",
7          "device": "/dev/vdb"
8        },
9        "volume_attach_2": {
10         "instance_id": "${openstack_compute_instance_v2.instance.id}",
11         "volume_id": "8985e9ae-d658-475d-956b-4ee2fb7314f1",
12         "device": "/dev/vdc"
13       },
14       "volume_attach_3": {
15         "instance_id": "${openstack_compute_instance_v2.instance.id}",
16         "volume_id": "0bbe2516-d551-4242-83c3-3eb96fa1e58b",
17         "device": "/dev/vdd"
18       }
19     }
20   }
21 ]

```

Figure 10. Web-Class Case—Terraform Template Example.

Figure 10 shows a created terraform template that maps the provisioning resources to virtual machines. Figure 11 shows the VM volume tree created by BVMO during the integration process.

```

root@bvmo-vm:~# tree -L 2 /mnt
/mnt
├── vdb
│   ├── lost+found
│   └── nvm
├── vdc
│   ├── Lect01-Coure-Intro.pdf
│   ├── Lect02-Introduction-to-Internet.pdf
│   ├── Lect03-Introduction-to-WWW.pdf
│   ├── Lect04-Introduction-to-HTML.pdf
│   ├── Lect05-Introduction-to-CSS.pdf
│   ├── Lect06-Introduction-to-NodeJS.pdf
│   ├── Lect07-Introduction-to-Nginx.pdf
│   └── lost+found
└── vdd
    ├── Lect01-Coure-Intro.mp4
    ├── Lect02-Introduction-to-Internet.mp4
    ├── Lect03-Introduction-to-WWW.mp4
    ├── Lect04-Introduction-to-HTML.mp4
    ├── Lect05-Introduction-to-CSS.mp4
    ├── Lect06-Introduction-to-NodeJS.mp4
    ├── Lect07-Introduction-to-Nginx.mp4
    └── lost+found

7 directories, 14 files

```

Figure 11. Web-Class Case—Virtual-Machine Volume Tree.

As shown in Figure 11, it can be confirmed that node version manager (NVM v0.39.4) software (in/mnt/vdb) for web classes, pdf files of lecture materials (in/mnt/vdc) and video lecture materials (in/mnt/vdd) imported from existing data resources are loaded into the virtual machine and that the software is normally executed, as shown in Figure 12.

```
root@bvmo-vm:/mnt/vdb# nvm --version
0.38.0
root@bvmo-vm:/mnt/vdb# node --version
v20.4.0
root@bvmo-vm:/mnt/vdb#
```

Figure 12. Web-Class Case—Software Check.

#### 4.2. Qualitative Evaluation

A qualitative evaluation suggests a method to show flexibility and a method to show the reusability of the proposed method. To demonstrate whether the system was flexible, the combinability of the provisioning resources used in this study was reviewed.

Assuming that all coupling dependencies between provisioned resources are resolved, the number of coupling cases can be extracted as follows:

$$(\text{The number of Instances}) = \sum_{k=1}^N C(N, k) \quad (1)$$

$$C(N, k) = \frac{N!}{(k!(N-k)!)}, \quad N = \text{total number of provisioning resources}$$

If this was applied to the 12 provisioning resources used in the case study and calculated, a total of 4095 virtual-machine combinations were derived. The above results show the flexibility corresponding to the case in which provisioning resources are independent.

Dependencies may also exist between provisioned resources. For example, if the OS is Windows, software that can operate on Windows should be selected. This type of relationship can be resolved by creating a combination table (Table 7) because the cases differ depending on the OS, software and data that must be selected. Table 7 lists some of the combined cases based on the web-class case study and the case study system presented in this study.

As listed in Table 7, meaningful cases indicate cases in which a virtual machine combination is configured to meet the user's requirements in consideration of dependencies (for example, in Case 1 (web-class case study), selecting software for web classes and OS, and data compatible with the corresponding software), and unmeaningful cases indicate cases in which dependencies are not resolved or are unclear.

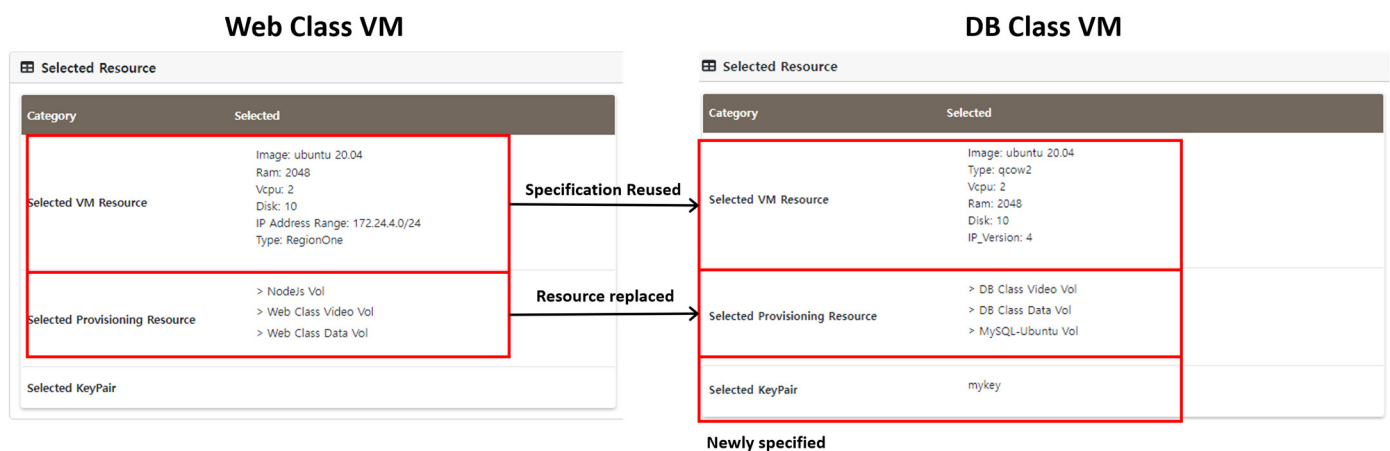
To demonstrate the reusability of BVMO, the application of a virtual machine built into the web class introduced above is presented. If a DB class is newly created and only the applied software and data are different, the VM components of the OS of the existing web class can be reused and applied to create a new VM. Figure 13 shows the change from web-class VM to DB-class VM through reuse.

Figure 13 shows that the VM's OS and other specific elements were reused, and the provisioned resources were changed to support the type of class to be performed. In addition, it can be confirmed that keypairs were newly specified to support additional security performance.



**Table 7.** Resource Coupling Table.

Category	OS	Software	Data	Empty
Resources	Ubuntu20.04 CentOS7	MySQL-CentOS	DB Class Video	Default Vol 1 Default Vol 2
		Node.Js-CentOS	DB Class Data	
		MySQL-Ubuntu	Web Class Video	
		Node.Js-Ubuntu	Web Class Data	
Meaningful Cases				
Case 1 (Web-class case study)	Ubuntu20.04	Node.Js-Ubuntu	Web Class Video Web Class Data	-
Case 2 (Web-class case with CentOS)	CentOS7	Node.Js-CentOS	Web Class Video Web Class Data	-
Case 3 (Ubuntu System with additional storage)	Ubuntu20.04	-	-	Default Vol 1
...				
Unmeaningful Cases				
Case n-1	Ubuntu20.04	MySQL-CentOS	DB Class Video	Default Vol 1 Default Vol 2
Case n	CentOS7	MySQL-Ubuntu Node.Js-Ubuntu	DB Class Video Web Class Data	Default Vol 1 Default Vol 2

**Figure 13.** Web-Class VM to DB-Class VM.

#### 4.3. Quantitative Evaluation

A comparative analysis of the method of applying BVMO, the system proposed in this study, and the method of creating a VM using only existing IaC tools was performed. OpenStack, a private cloud, was used for the experiment. In addition, to establish time measurement standards for each case, the log analysis process that occurred when creating a VM in OpenStack was performed.

Figure 14 shows an excerpt of the parts that can be utilized among the logs generated when creating a VM in OpenStack.

As shown in Figure 14, the OpenStack Nova component undergoes a two-step process to create an instance. The instance-creation process starts at the instance-create start time and provides the information received for VM creation to the hypervisor. Instance-spawning time measures the time taken to spawn an instance that Nova can control by

receiving the VM created by the hypervisor. The Instance-build time refers to the time required to build a runnable VM during the verification process of the spawned instance.

<pre>Jul 19 08:02:22 jae nova-compute[434290]: DEBUG nova.compute.manager [None req-31c1ff60-2010-44ce-ab51-4b7c8a4fba64 admin admin] [instance: f1499fa5-8a03-4201-b9cc-88bc3d0ce09c] Starting instance... {{(pid=434290) _do_build_and_run_instance /opt/stack/nova/nova/compute/manager.py:2211}}</pre>	Instance Create Start Time
<pre>Jul 19 08:02:24 jae nova-compute[434290]: INFO nova.compute.manager [None req-31c1ff60-2010-44ce-ab51-4b7c8a4fba64 admin admin] [instance: f1499fa5-8a03-4201-b9cc-88bc3d0ce09c] Took 1.93 seconds to spawn the instance on the hypervisor.</pre>	Instance Spawn Time
<pre>Jul 19 08:02:24 jae nova-compute[434290]: INFO nova.compute.manager [None req-31c1ff60-2010-44ce-ab51-4b7c8a4fba64 admin admin] [instance: f1499fa5-8a03-4201-b9cc-88bc3d0ce09c] Took 2.20 seconds to build instance.</pre>	Instance Build Time

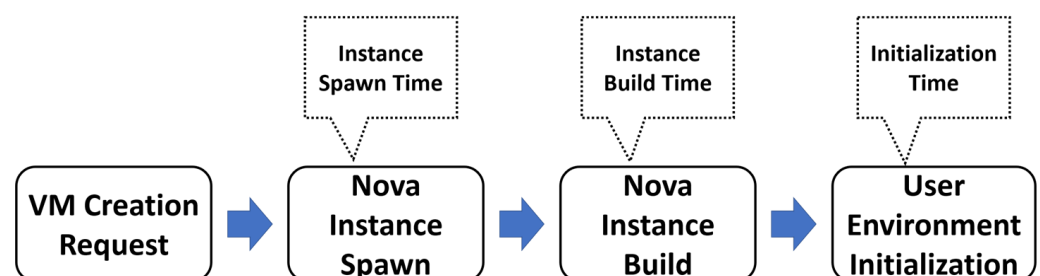
**Figure 14.** OpenStack Logs for Quantitative Evaluation.

The VM created above performs an initialization to build an environment that users can utilize. Cloud-init is currently used as a cloud-instance-initialization tool. Figure 15 shows an excerpt from the cloud-init logs that can be utilized.

<pre>[ 0.000000] Linux version 5.4.0-131-generic (buildd@lcy02-amd64-108) (gcc version 9.4.0 (Ubuntu 9.4.0-1ubuntu1~20.04.1)) #147-Ubuntu SMP Fri Oct 14 17:07:22 UTC 2022 (Ubuntu 5.4.0-131.147-generic 5.4.210)</pre>	Initialization Start
<pre>[ 111.056844] cloud-init[1296]: Cloud-init v. 22.3.4-0ubuntu1~20.04.1 running 'modules:final' at Tue, 18 Jul 2023 10:56:02 +0000. Up 109.18 seconds.</pre>	Initialization End

**Figure 15.** Cloud-Init Logs for Quantitative Evaluation.

In the case of cloud-init-execution logs, the data that can be utilized are the start and end times. The process of measuring the VM creation time based on the log-analysis results derived from Figures 14 and 15 is shown in Figure 16.



**Figure 16.** VM-Creation Flow for Quantitative Evaluation.

Based on the time-measurement elements shown in Figure 16, the configuration of the experiment was divided into two cases when creating a VM: (1) when creating an empty instance with only the OS installed and (2) when performing software installation in the user-environment-initialization stage.

In the case of the method using IaC listed in Table 8, the spawn time was 1.94 s on average, and the build time was 2.216 s on average. When using the BVMO presented in this study, the average spawn time was 3.154 s, and the average build time was 5.384 s. The

execution time, according to the specification interpretation, occurred in the process of the BVMO reusing existing resources.

**Table 8.** Test 1—VM with OS.

Method	Iteration (Count)	VM-Creation Phase			Total (Seconds)
		Instance Spawn (Seconds)	Instance Build (Seconds)	Initialization (Seconds)	
IaC	1st	1.89	2.16	112.73	120
	2nd	1.86	2.09	106.9	114
	3rd	1.93	2.2	111.75	118
	4th	2.06	2.32	109.14	117
	5th	1.96	2.31	110.76	117
BVMO	1st	3.06	5.2	81.12	91
	2nd	3.11	5.29	82.98	93
	3rd	3.13	5.27	84.24	95
	4th	3.29	5.63	83.6	95
	5th	3.18	5.53	92.76	104

In addition, in the case of the initialization time, the method using IaC averaged 110.256 s, and the method using BVMO averaged 84.94 s, resulting in a reduction of approximately 25.316 s in the proposed BVMO. The initialization time of the OS, which is a major element of the computing system among existing resources, affects the initial creation speed of the VM. Therefore, it can be confirmed that the approach of pre-creating the combined OS resources proposed by the BVMO is effective.

For the total execution time, the results include the spawn time, build time, initialization time and idle time according to the progress of the VM-creation flow. An average of 117.2 s for the IaC method and 95.6 s for the BVMO method were measured. It was confirmed that the proposed BVMO method could perform as fast as 21.8 s.

Table 9 lists the experimental results for performing the software installation required by the user during the user-environment-initialization step. The software used in the experiment was performed by adopting npm, which can be installed from a Linux package repository.

**Table 9.** Test 2—VM with Software.

Method	Iteration (Count)	VM Creation Phase			Total (Seconds)
		Instance Spawn (Seconds)	Instance Build (Seconds)	Initialization (Seconds)	
IaC	1st	1.95	2.2	278.01	305
	2nd	1.89	2.13	286.37	314
	3rd	1.78	2.06	280.65	312
	4th	1.73	1.99	275.57	304
	5th	1.81	2.12	282.66	311
BVMO	1st	3.05	5.39	90.33	101
	2nd	3.12	5.39	88.67	99
	3rd	3.08	5.22	89.41	100
	4th	3.05	5.24	96.71	107
	5th	3.14	5.35	95.56	106

In the case of the method using IaC, as listed in Table 9, the spawn time was 1.832 s on average, and the build time was 2.1 s on average. When using the BVMO presented in this study, the average spawn time was 3.088 s, and the average build time was 5.318 s. This result was similar to the experimental results derived from Test 1.

In addition, in the case of initialization time, the average execution time of the method using IaC was 280.652 s, and that of the method using BVMO was 92.136 s. It was confirmed that a difference of 188.489 s appeared between the cases where software was used among the provisioning resources and those where it was not. In the case of OS, there are various supports in terms of platforms and hypervisors, except for special cases (e.g., Baremetal system); however, in the case of software that operates internally, a large difference appears as the time required to download and set up the software, and this is reflected in the user-environment-initialization stage.

The total execution time of Test 2 was 309.2 s for the method using IaC and 102.6 s for the method using BVMO.

Table 10 lists the experimental results for increasing the number of software packages when creating a VM following Test 1 and Test 2.

**Table 10.** Test 3—VM with Number of Two Software.

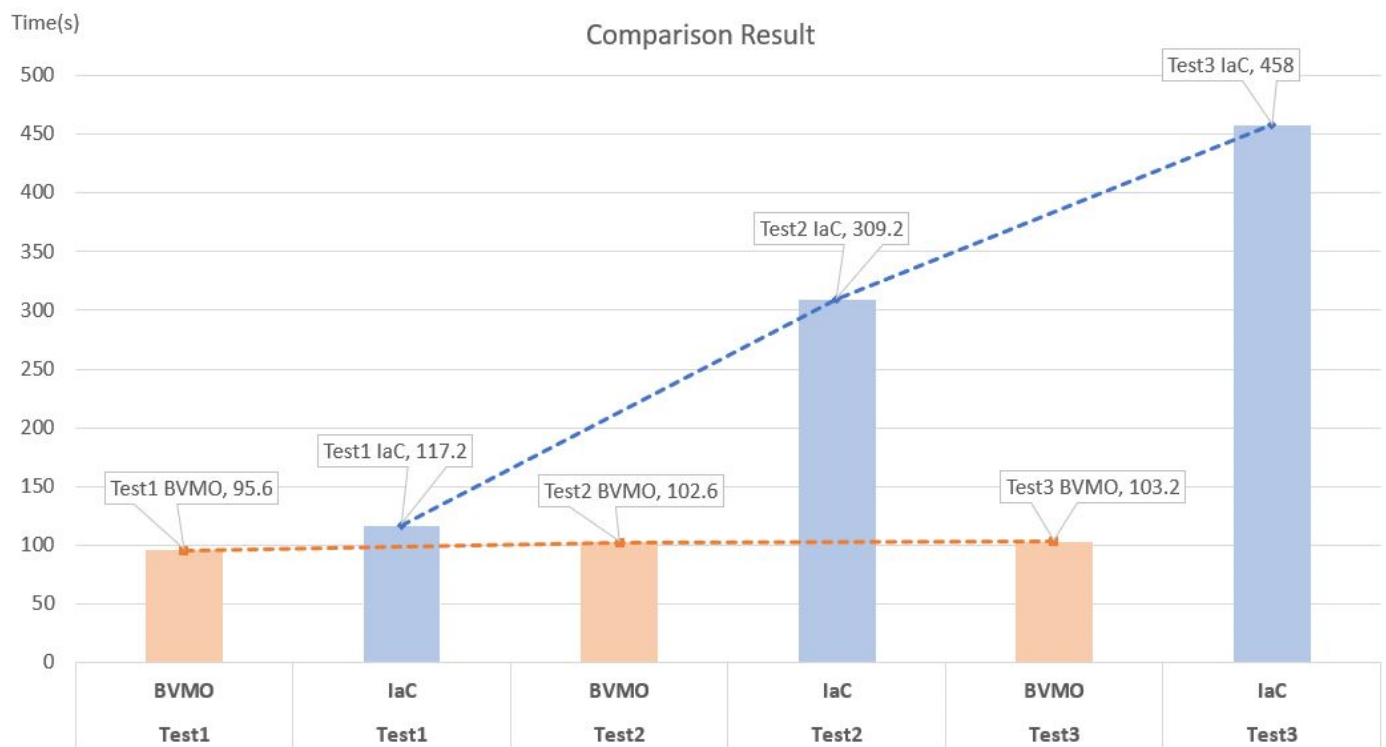
Method	Iteration (Count)	VM Creation Phase			Total (Seconds)
		Instance Spawn (Seconds)	Instance Build (Seconds)	Initialization (Seconds)	
IaC	1st	2.64	2.89	461.26	469
	2nd	2.7	3	440.83	449
	3rd	2.55	3.16	455.66	451
	4th	2.1	2.96	460.43	455
	5th	2.62	3.2	458.74	466
BVMO	1st	3.13	5.39	93.95	105
	2nd	3.09	5.31	89.91	101
	3rd	3.17	5.42	93.24	105
	4th	3.06	5.57	92.83	104
	5th	3.16	5.28	91.34	101

As listed in Table 10, in the case of the method using IaC, the average spawn time was 2.522 s, and the average build time was 3.042 s. When BVMO was used, the average spawn time was 3.122 s, and the average build time was 5.394 s. In the previous IaC method, the spawn time was approximately 1.9 s, and the build time was approximately 2.1 s; however, the average values differed by about 0.6 s and 0.9 s, respectively. This appears to occur because the IaC specification analysis process is added according to the software addition. However, because the BVMO method performs only volume connections, it is confirmed that there is no significant difference from the previous results.

In addition, the initialization time of the method using IaC was 455.384 s on average, and that of the method using BVMO was 92.584 s. This was confirmed by an increase in the load on the system as the software was added. However, it was confirmed through several experiments that BVMO, which only performs volume connections, produces consistent results.

As for the total execution time according to the increase in software, it was confirmed that IaC had an average of 458 s and BVMO had an average of 103.2 s.

Figure 17 shows the results of each experiment performed.



**Figure 17.** All Test Comparison Results.

As a result, it was confirmed that BVMO consistently has a time-reduction effect compared with the method using IaC. In contrast to the IaC method, which differs in time depending on the software provision, BVMO showed a consistent required time in each experiment. This is assumed to occur because BVMO utilizes provisioned resources, does not require additional resource-preparation time and operates consistently to create virtual machines (volume mapping after instance preparation).

## 5. Conclusions

We propose BVMO, a method that can increase reusability and flexibly create user-customized virtual machines. In contrast to existing virtual-machine-creation methods, a process and architecture for constructing a new customized virtual machine by configuring and combining a resource volume pool were established. The proposed method was specified by applying OpenStack, an actual open-source platform, and it was confirmed that this approach could reduce time and cost compared with existing methods. Therefore, the proposed BVMO can be applied to create a VM that satisfies user requirements and promotes the introduction of private clouds by organizations or companies by solving the technical difficulties in building a virtual machine. Currently, research is in progress on a mechanism that introduces artificial intelligence (AI) technology and recommends various provisioning resources in the resource pool of the proposed BVMO according to user needs. Additionally, research on a management structure that collects and manages historical data related to the resource pool, such as the frequency of use of the provisioned resources of the resource pool by users when creating VMs, requirements and mapping information for VM resources selected by requirements, is underway. In the future, we plan to expand the proposed BVMO to a virtual-machine-mediation framework that classifies and recommends VMs and research-related technologies.

**Author Contributions:** Conceptualization, J.P.; methodology, J.P.; software, S.J.; validation, S.J.; project administration, K.Y.; funding acquisition, K.Y. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. NRF-2021R1A2C1006177).

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The data used to support the findings of this study are available from the authors upon reasonable request.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Singh, R.P.; Haleem, A.; Javaid, M.; Kataria, R.; Singhal, S. Cloud Computing in Solving Problems of COVID-19 Pandemic. *J. Ind. Integr. Manag.* **2021**, *6*, 209–219. [CrossRef]
2. Parast, F.K.; Sindhav, C.; Nikam, S.; Yekta, H.I.; Kent, K.B.; Hakak, S. Cloud computing security: A survey of service-based models. *Comput. Secur.* **2022**, *114*, 102580. [CrossRef]
3. Jamsa, K. *Cloud Computing*, 2nd ed.; Jones & Bartlett Learning: Boston, MA, USA, 2022; pp. 1–23. ISBN 978-1-284-23397-1.
4. Lehweiss, M. *Future Networks, Services and Management*; Toy, M., Ed.; Springer: Cham, Switzerland, 2021; pp. 213–243. ISBN 978-3-030-81960-6.
5. Amazon Elastic Compute Cloud Documentation. Available online: [https://docs.aws.amazon.com/ec2/index.html?nc2=h\\_ql\\_doc\\_ec2](https://docs.aws.amazon.com/ec2/index.html?nc2=h_ql_doc_ec2) (accessed on 24 May 2023).
6. Open Cloud | Google Cloud. Available online: <https://cloud.google.com/open-cloud?hl=en> (accessed on 24 May 2023).
7. Open Source Cloud Computing Platform Software—OpenStack. Available online: <https://www.openstack.org/software/> (accessed on 24 May 2023).
8. Apache CloudStack: Open Source Cloud Computing. Available online: <https://cloudstack.apache.org/about.html> (accessed on 24 May 2023).
9. Kai, Z.; Youyu, L.; Qi, L.; Hao, S.C.; Liping, Z. Building a private cloud platform based on open source software OpenStack. In Proceedings of the 2020 International Conference on Big Data and Social Sciences (ICBDSS), Xi'an, China, 14–16 August 2020; pp. 84–87.
10. Barhate, S.M.; Dhore, M.P. Hybrid Cloud: A Cost Optimised Solution to Cloud Interoperability. In Proceedings of the 2020 International Conference on Innovative Trends in Information Technology (ICITIIT), Kottayam, India, 13–14 February 2020; pp. 1–5.
11. Alonso, J.; Echevarria, L.O.; Huarte, M. CloudOps: Towards the Operationalization of the Cloud Continuum: Concepts, Challenges and a Reference Framework. *Appl. Sci.* **2022**, *12*, 4347. [CrossRef]
12. Gupta, S.; Iyer, S.; Agarwal, G.; Manoharan, P.; Algarni, A.D.; Aldehim, G.; Raahemifar, K. Efficient Prioritization and Processor Selection Schemes for HEFT Algorithm: A Makespan Optimizer for Task Scheduling in Cloud Environment. *Electronics* **2022**, *11*, 2557. [CrossRef]
13. Belgacem, A. Dynamic resource allocation in cloud computing: Analysis and taxonomies. *Computing* **2022**, *104*, 681–710. [CrossRef]
14. Verma, A.; Bhattacharya, P.; Bodkhe, U.; Saraswat, D.; Tanwar, S.; Dev, K. FedRec: Trusted rank-based recommender scheme for service provisioning in federated cloud environment. *Digit. Commun. Netw.* **2023**, *9*, 33–46. [CrossRef]
15. Funika, W.; Koperek, P.; Kitowski, J. Automated cloud resources provisioning with the use of the proximal policy optimization. *J. Supercomput.* **2022**, *79*, 6674–6704. [CrossRef]
16. Rong, C.; Geng, J.; Hacker, T.J.; Bryhni, H.; Jaatun, M.G. OpenIaC: Open infrastructure as code—The network is my computer. *J. Cloud Comput.* **2022**, *11*, 12. [CrossRef]
17. Ansible Use Case. Available online: <https://www.ansible.com/use-cases> (accessed on 24 May 2023).
18. Boot Stages—Cloud-Init 23.2 Document. Available online: <https://cloudinit.readthedocs.io/en/latest/topics/boot.html> (accessed on 24 May 2023).
19. Chiari, M.; Pascalis, M.D.; Pradella, M. Static Analysis of Infrastructure as Code: A Survey. In Proceedings of the 2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C), Honolulu, HI, USA, 12–15 March 2022; pp. 218–225.
20. Awasthi, A.; Gupta, R. Multiple hypervisor based Open Stack cloud and VM migration. In Proceedings of the 2016 6th International Conference—Cloud System and Big Data Engineering (Confluence), Noida, India, 14–15 January 2016; pp. 130–134.
21. Martinez, J.; Ziadi, T.; Bissyandé, T.F.; Klein, J.; Traon, Y.L. Bottom-Up Technologies for Reuse: Automated Extractive Adoption of Software Product Lines. In Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), Buenos Aires, Argentina, 20–28 May 2017; pp. 67–70.
22. Shen, H.; Chen, L. A Resource-Efficient Predictive Resource Provisioning System in Cloud Systems. *IEEE Trans. Parallel Distrib. Syst.* **2022**, *33*, 3886–3900. [CrossRef]



23. Rajan, D.; Thain, D. Designing Self-Tuning Split-Map-Merge Applications for High Cost-Efficiency in the Cloud. *IEEE Trans. Cloud Comput.* **2017**, *5*, 303–316. [CrossRef]
24. Wei, Y.; Blake, M.B. Proactive virtualized resource management for service workflows in the cloud. *Computing* **2016**, *98*, 523–538. [CrossRef]
25. Nadeem, S.; Amin, N.; Zaman, S.K.; Khan, M.A.; Ahmad, Z.; Iqbal, J.; Khan, A.; Algarni, A.D.; Elmannai, H. Runtime Management of Service Level Agreements through Proactive Resource Provisioning for a Cloud Environment. *Electronics* **2023**, *12*, 296. [CrossRef]
26. Alwada'n, T.; Al-Tamimi, A.K.; Mohammad, A.H.; Salem, M.; Muhammad, Y. Dynamic congestion management system for cloud service broker. *Int. J. Electr. Comput. Eng.* **2022**, *13*, 872–883. [CrossRef]
27. Li, X.; Pan, L.; Liu, S. An online service provisioning strategy for container-based cloud brokers. *J. Netw. Comput. Appl.* **2023**, *214*, 103618. [CrossRef]
28. What Is Terraform | Terraform | HashiCorp Developer. Available online: <https://developer.hashicorp.com/terraform/intro> (accessed on 26 July 2023).
29. HashiStack. Available online: <https://hashistack.readthedocs.io/en/latest/> (accessed on 26 July 2023).
30. Welcome to the Heat Documentation!—Openstack-heat 20.1.0.dev63 Documentation. Available online: <https://docs.openstack.org/heat/latest/> (accessed on 26 July 2023).
31. Documentation | Vault | HashiCorp Developer. Available online: [https://developer.hashicorp.com/vault/docs?product\\_intent=vault](https://developer.hashicorp.com/vault/docs?product_intent=vault) (accessed on 7 August 2023).
32. Consul Documentation | Consul | HashiCorp Developer. Available online: [https://developer.hashicorp.com/consul/docs?product\\_intent=consul](https://developer.hashicorp.com/consul/docs?product_intent=consul) (accessed on 7 August 2023).
33. Documentation | Nomad | HashiCorp Developer. Available online: [https://developer.hashicorp.com/nomad/docs?product\\_intent=nomad](https://developer.hashicorp.com/nomad/docs?product_intent=nomad) (accessed on 7 August 2023).
34. Keystone, the OpenStack Identity Service—Keystone 23.1.0.dev57 Documentation. Available online: <https://docs.openstack.org/keystone/latest/> (accessed on 7 August 2023).
35. Welcome to Neutron's Documentation!—Neutron 23.0.0.0b3.dev257 Documentation. Available online: <https://docs.openstack.org/neutron/latest/> (accessed on 7 August 2023).
36. OpenStack Compute (Nova)—Nova 27.1.0.dev103 Documentation. Available online: <https://docs.openstack.org/nova/latest/> (accessed on 7 August 2023).
37. GCC, the GNU Compiler Collection—GNI Project. Available online: <https://gcc.gnu.org/> (accessed on 7 August 2023).
38. What Is Java and Why Do I Need it? Available online: [https://www.java.com/en/download/help/whatis\\_java.html](https://www.java.com/en/download/help/whatis_java.html) (accessed on 7 August 2023).
39. Our Documentation | Python.org. Available online: <https://www.python.org/doc/> (accessed on 7 August 2023).
40. Virtual Machine Instances | Compute Engine Documentations. Available online: <https://cloud.google.com/compute/docs/instances?hl=en> (accessed on 3 June 2023).
41. Amazon EC2 Instance Type—Amazon Web Service. Available online: <https://aws.amazon.com/en/ec2/instance-types/en> (accessed on 3 June 2023).
42. VM Sizes—Azure Virtual Machines | Microsoft Learn. Available online: <https://learn.microsoft.com/en-us/azure/virtual-machines/sizes/en> (accessed on 3 June 2023).
43. Launch and Manage Instances—Horizon 23.2.0.dev55 Documentation. Available online: <https://docs.openstack.org/horizon/latest/user/launch-instances.html> (accessed on 3 June 2023).
44. About Working with Virtual Machines—Apache CloudStack 4.18.0.0 Documentation. Available online: [https://docs.cloudstack.apache.org/en/latest/adminguide/virtual\\_machines.html#managing-virtual-machines](https://docs.cloudstack.apache.org/en/latest/adminguide/virtual_machines.html#managing-virtual-machines) (accessed on 3 June 2023).

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.