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

Joonseok Park  
Pusan National University  

Sumin Jeong  
Pusan National University  

Keunhyuk Yeom (✉ yeom@pusan.ac.kr)  
Pusan National University

Research Article

Keywords: cloud computing, private cloud, virtual machine, cloud platform

Posted Date: March 3rd, 2023

DOI: https://doi.org/10.21203/rs.3.rs-2632912/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License. Read Full License
Abstract

A cloud-computing company or user needs to 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 companies and users. In this paper, we propose a Bespoke Virtual Machine Orchestrator (BVMO) as a method of 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.

1. Introduction

Cloud computing [1] provides a service environment in which IT resources can be virtualized and used. This computing environment is evolving with the advent of public cloud services such as AWS and Google [2] and the emergence of open-source platforms that provide private cloud service construction, such as OpenStack and CloudStack. In the case of a public cloud, a user pays a certain amount and receives a prebuilt cloud service. A private cloud [3] is utilized by a company or a user to directly build and use a cloud service provision environment. “Hybrid cloud,” which refers to the use of a public cloud and private cloud together [4], 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 the service usage. To reduce the cost of cloud computing and build a cloud service environment in the form desired by companies or users, a private cloud environment must be established.

In the case of private clouds, dependence on cloud vendors can be reduced, but there is a technical difficulty: it is difficult to build the desired cloud service delivery environment without expert knowledge about the cloud. The most basic technical application for building and receiving cloud services is to build 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, a technology is needed to provide a customized virtual machine environment suitable for various companies or users who wish to operate a private cloud.

Recently, infrastructure as code (IaC) has emerged as a concept of automating and providing virtual machine provision and cloud infrastructure construction. Tools such as Ansible [5] and Cloud-Init [6], which are used for automating virtual machine construction as support tools reflecting the concept of IaC, are being researched and developed. In this paper, 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 apply the concept of software product lines
to build resource volumes used for virtual machines as core assets and present an approach to utilize them. We propose a Bespoke Virtual Machine Orchestrator (BVMO) for building a user-customized virtual machine by applying these concepts.

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 the BVMO. Section 4 presents a case study and evaluation of the process of creating a custom virtual machine using BVMO, and Section 5 concludes the paper.

2. Related Work

2.1 Private Cloud Platform

OpenStack [7] and CloudStack [8] are representative open-source platforms that support the construction of 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 is provided to support virtual machine creation and cloud environment construction using a Hot Orchestration Template. Similar to OpenStack, CloudStack is an open-source cloud platform that supports deployment and management of cloud service resources. It supports various hypervisors [9], such as Hyper-V, XenServer, and vSphere. While OpenStack can build a cloud by combining multiple components, most of the modules in CloudStack are bundled into one binary.

2.2 Software Product Line Engineering

Among the software development methodologies, the product line engineering [10] method 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) are identified in advance and reused to flexibly develop new software products.

Even in a cloud environment, there is a concept of resource provisioning [11], in which various cloud resources existing in a cloud platform are built, prepared, and utilized in advance before being combined. This can be linked to the approach of product line engineering. Therefore, in this paper, we propose a method of combining the concept of 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, i.e., resource provisioning, and a new virtual machine is built by binding these assets.

2.3 Provisioning Cloud Resources

Rajan et al. [12] proposed a method for distributing cloud resources by applying the Map-Reduce-Merge method based on cloud workloads. After cloud resources were classified into task units and the divided units were identified as one workload, the Map-Reduce-Merge technique was applied in the step of allocating the workload to a work queue.
In contrast, in the proposed approach, apart from the method of separating cloud resources into workload units and then dividing and conquering them, common and reusable cloud resources are identified in advance and built. The difference is that a virtual machine is created by combining prebuilt cloud resources according to requirements, in a manner similar to building with Lego blocks.

In the study of Wei et al. [13], the decision system was formalized in the configuration of the management system of the cloud platform 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. It involves merging instances by analyzing the quality of service (QoS). Therefore, this paper presents a case where QoS is considered in the process of combining. The proposed method differs from the method considering simple QoS in the part where combining is performed according to requirements and merging is performed by reusing pre-established resources.

3. Bespoke Virtual Machine Orchestrator

3.1 Conceptual Process for Bespoke Virtual Machine Orchestrator

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

First, in the Resource Pool Construction step, provisioning resources are divided into operating system, software, and data resource items. Operating system resources refer to CentOS, Ubuntu, etc. that operate virtual machines. Software resources refer to various software programs, such as GCC, Java, and Python, that users need in a virtual environment. Data resources represent various structured and unstructured data, such as text, documents, images, and videos.

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

Third, in the Resource Configuration step, setting and combination of selected resource items are performed.

Fourth, in the Customized VM Creation step, a user-customized virtual machine reflecting settings and combinations is created. The proposed method creates various provisioning resource (data, software, etc.) pools and provides flexibility in building a virtual environment by combining resource pools that meet the needs of each user.

3.2 Architecture of Bespoke Virtual Machine Orchestrator
To realize the construction process presented in Section 3.1, a virtual environment construction architecture using provisioning resources is employed, as shown in Fig. 2.

Table 1 describes the main components of the architecture presented in Fig. 2.

<table>
<thead>
<tr>
<th>Element</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Provisioning Resource Handler</td>
<td>A component that manages resources scattered in the cloud environment to create a virtual environment</td>
</tr>
<tr>
<td></td>
<td>Virtual Environment Handler</td>
<td>A component that builds, reconfigures, and manages virtual environments</td>
</tr>
<tr>
<td></td>
<td>Access Handler</td>
<td>A component that handles resource-related requests for user authentication, resource access permission setting, and requirements</td>
</tr>
</tbody>
</table>

The main modules of each component perform the following roles.

- **Resource Template Management**: Manages templates that specify provisioning resource binding information that reflects user requirements
- **Pool Management**: Supports Create, Read, Update, Delete (CRUD) of provisioned resources and allocation of provisioned resources according to interpretation of user requirements
- **Virtual Machine Creation**: Interprets template information and creates a virtual environment in which provisioning resources are bound
- **Virtual Machine Configuration**: Supports the creation, reconfiguration, and assembly of orchestration templates for combining provisioned resources
- **Authority Management**: Authenticates users and provides access to provisioning resources
- **Request Management**: Invokes the application programming interface (API) of the cloud platform that performs provisioning resource integration, requests resource combination, etc.

Table 2 describes the resources presented in the architecture.
As presented in the architecture, the Provisioning Resource Handler component manages provisioning resources and resource templates. Delivery of provisioned resources and API calls are performed through the Access Handler component. The Virtual Environment Handler component combines resource items and creates templates based on them. Additionally, it creates, separates, and combines provisioned resources using templates. Through this, a virtual environment reflecting the resources is built. A corresponding application scenario is described in Section 3.3.

### 3.3 Application Scenario for BVMO

BVMO allows the creation and management of user-customized virtual machines in a private cloud environment. Figure 3 shows an application scenario involving a virtual machine system based on the architecture presented in Section 3.2, in which a virtual environment is built by applying the provisioning resource pool proposed in this paper. When requesting creation of a virtual environment, user requirements are collected and analyzed. Then, provisioning resources suitable for the requirements are delivered. The virtual environment creation management system combines resource items by reflecting requirements (selected resource information). It also creates a template describing resource combination information that can be interpreted and executed on the cloud platform. The created template is used to combine necessary resources, and a user-customized virtual environment is created by applying additional information such as network settings required when creating a virtual environment.

### 4. Case Study And Evaluation

#### 4.1 Case Study

We present a case study in which the proposed virtual machine construction process is applied to an actual OpenStack private cloud. The environment is based on JavaScript and the framework that drives it.
The user requirements are presented in Table 3.

<table>
<thead>
<tr>
<th>Label</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Ubuntu 18.04 LTS</td>
</tr>
<tr>
<td>Software</td>
<td>Node.JS, npm</td>
</tr>
<tr>
<td>Hardware</td>
<td>CPU: 1 core</td>
</tr>
<tr>
<td></td>
<td>RAM: 1024 MB</td>
</tr>
<tr>
<td></td>
<td>Disk: 30 GB</td>
</tr>
<tr>
<td>Additional Software</td>
<td>Basic source code</td>
</tr>
</tbody>
</table>

Figure 4 shows the results of searching the resource volume pool and binding the OS and software according to the user requirements presented in Table 3.

In the search, the resources are classified by the name of the volume.

Hardware resources are loaded and reflected as variables in the template. Figure 5 shows an OpenStack Flavor template used to reflect hardware resources to VMs, and Fig. 6 shows a template that combines VMs and provisioning resources.

After the individual virtual machine resources in Figs. 5 and 6 are created, each resource is assembled according to the virtual machine template. Figure 7 shows a combination virtual machine template in which a Flavor template tailored to provisioning resources and two software provisioning resources are added to an instance template.

A virtual machine template reflecting the requirements, as shown in Fig. 7, is created and passed to the orchestrator. The delivered template undergoes the steps of OpenStack's orchestrator operation process (template verification, resource creation, resource setting, and completion) for the creation of a combined virtual machine in OpenStack, as shown in Fig. 8.

Figure 9 shows the operation of the built virtual environment and the combination of provisioned resources.

4.2 Evaluation

We compared the time required for each step between the existing virtual machine creation method, which performs internal control of cloud resources through Cloud-Init, and the proposed method, which performs resource connection required for a virtual machine based on provisioned resources. The virtual machine creation phase consists of the steps as shown in Fig. 10.
As shown in the figure, the virtual machine creation phase begins when a request is sent to the orchestrator (Request Orchestration). At the Resource Binding Time, the cloud platform performs the operation shown in Fig. 11.

After receiving the orchestration request, the orchestrator verifies the template and then initializes and creates cloud resources (network interfaces, volumes, etc.) described in the template. Then, the details of the resources are set according to the contents of the template.

The Virtual Machine Booting Time operation is shown in Fig. 12.

After the resource creation and configuration process, an empty virtual machine is created according to the configured resources. Depending on the settings of the operating system mounted on the virtual machine, Virtual Machine Booting Time is performed variably.

Figures 13 and 14 present graphs showing the central processing unit (CPU) and memory usage, respectively, for each stage in which orchestration requests were performed.

The steps corresponding to the numbers on the x-axes in Figs. 13 and 14 are presented in Table 4.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Template Verification and Analysis</td>
</tr>
<tr>
<td>2</td>
<td>Resource Initialization and Configuration</td>
</tr>
<tr>
<td>3</td>
<td>Empty Virtual Machine Creation</td>
</tr>
<tr>
<td>4</td>
<td>Operating System Initialization</td>
</tr>
<tr>
<td>5</td>
<td>Operating System Configuration</td>
</tr>
</tbody>
</table>

As shown in the figure, steps 1–3 used little memory. In particular, in step 2, it was confirmed that the CPU was not used effectively, because the CPU usage was low. In the existing method, other components (volume processing component, network device management component, etc.) used during the virtual resource preparation step were combined and operated; thus, there are problems with dependencies between them, differences in their completion times, etc., resulting in a bottleneck. In contrast, because the proposed method has a structure in which virtual resource preparation is already completed, it is not affected by the time required for connection and processing with other components; thus, the Resource Binding Time was improved, as shown in Fig. 15.

The evaluation experiment was conducted by implementing the method using Cloud-Init—a virtual machine creation method that reflects existing user data—and the proposed virtual machine creation
method ten times each. Figures 15 and 16 show the results obtained when the size of the provisioned resources (resource volume) was increased in increments of 5 GB.

As shown in Fig. 15, the size of the provisioned resources did not significantly affect the difference in the Resource Binding Time between the existing method and the proposed method. However, comparing the proposed method with the existing method revealed an average difference of approximately 8.1 s. It is judged that this occurs in the case of utilizing established provisioning resources and in the process of preparing uninitialized resources.

As shown in Fig. 16, the provisioning resource size did not significantly affect the Virtual Machine Booting Time between the existing method and the proposed method. However, the proposed method reduced the required time by 5.1 s on average.

This is because in contrast to the existing method in which a virtual machine must be configured from an image, the proposed method performs a reboot in a state where the OS is prepared in advance.

Figures 17 and 18 show the usage of computing resources during the Resource Binding Time. The average CPU usage was 31.336% and 24.65% for the proposed and the existing method, respectively. The average memory usage was measured to be 1.698% for the proposed method and 0.142% for the existing method. In the part of loading and processing pre-existing resources on the memory, it is judged that the proposed method had slightly higher CPU and memory usage than the existing method. However, compared with the existing method, the proposed method exhibited stable CPU and memory usage.

Figures 19 and 20 show the usage of computing resources at the Virtual Machine Booting Time. The average CPU usage was 19.242% for the proposed method and 20.288% for the existing method. The average memory usage was 7.52% for the proposed method and 7.146% for the existing method. During the Virtual Machine Booting Time, there were almost no differences in computing-resource usage between the existing method and the proposed method. This is because the hypervisor (KVM, QEMU, etc.) is because the proposed method does not affect the virtual machine configuration stage by performing a series of processes.

5. Conclusion

We propose BVMO—a method that can increase reusability and flexibly create a user-customized virtual machine. In contrast to the existing virtual machine creation method, a process and architecture for constructing a new customized virtual machine by configuring a resource volume pool and combining the volume pool were established. The proposed method was specified by applying OpenStack—an actual open-source platform—and it was confirmed that this approach can reduce the time cost compared with the existing method. In the future, we plan to study techniques for building and managing virtual machines in multi-private environments rather than single private cloud environments.

Abbreviations
BVMO: Bespoke Virtual Machine Orchestrator
CPU: Central processing unit
CRUD: Create, Read, Update, Delete
IaC: Infrastructure as code
QoS: Quality of service

**Declarations**

Acknowledgements

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

Authors’ contributions

Joonseok Park is mainly responsible for the idea of this manuscript and the design of the system and method. Sumin Jeong designed the experimental set-up and use cases, and obtained data. Keunhyuk Yeom provided the necessary environment and equipment for the experiment, as well as academic guidance for this article.

Availability of data and materials

The data used to support the findings of this study are available from the authors upon reasonable request.

Ethics approval and consent to participate

Not applicable.

Competing interests

The authors declare that they have no competing interests.

**References**


5. Ansible, https://www.ansible.com/use-cases

**Figures**

**Figure 1**

Conceptual Process of BVMO
Figure 2

Architecture
Figure 3

Application Scenario
Figure 4

Resource Query and Binding

```
1
2  
3  
4  
5  
6  
7  
8
9  

{  
  "type": "OS::Nova::Flavor",  
  "properties": {  
    "name": "test_flavor",  
    "ram": 1024,  
    "vcpus": 1,  
    "disk": 30  
  }  
}
```

Requirement Hardware – Flavor Binding

Figure 5
OpenStack Flavor Template

```json
{
  "type": "OS::Cinder::VolumeAttachment",
  "properties": {
    "instance_uuid": {
      "get_resource": "node-ubuntu"
    },
    "volume_id": {
      "get_resource": "test-instance"
    },
    "mountpoint": "/dev/vda"
  }
}
```

**Requirement Software – Volume Binding (Template level)**

**Figure 6**

OpenStack Volume-VM Integration Template

```json
{
  "test": {
    "type": "OS::Nova::Server",
    "properties": {
      "flavor": {
        "get_resource": "created_flavor"
      },
      "block_device_mapping_v2": [
        {
          "uuid": {
            "get_resource": "created_volume1"
          },
          "delete_on_termination": "False"
        },
        {
          "uuid": {
            "get_resource": "created_volume2"
          },
          "delete_on_termination": "False"
        }
      ]
    }
  }
}
```

**Figure 7**

Virtual machine templates that reflect requirements
Figure 8
Orchestration Result

```
test@test-test-instance-tcnzvnnxxuy2z:~/node_pack$ ls -al /dev | grep vd
brw-rw---- 1 root  disk  252,  0 Oct 27 16:01 vda
brw-rw---- 1 root  disk  252,  1 Oct 27 16:01 vda1
brw-rw---- 1 root  disk  252, 14 Oct 27 16:01 vda14
brw-rw---- 1 root  disk  252, 15 Oct 27 16:01 vda15
brw-rw---- 1 root  disk  252, 16 Oct 27 16:01 vdb
brw-rw---- 1 root  disk  252, 17 Oct 27 16:01 vdb1
brw-rw---- 1 root  disk  252, 30 Oct 27 16:01 vdb14
brw-rw---- 1 root  disk  252, 31 Oct 27 16:01 vdb15
brw-rw---- 1 root  disk  252, 32 Oct 27 16:01 vdc
brw-rw---- 1 root  disk  252, 33 Oct 27 16:01 vdc1
brw-rw---- 1 root  disk  252, 46 Oct 27 16:01 vdc14
brw-rw---- 1 root  disk  252, 47 Oct 27 16:01 vdc15
```

Figure 9
Integrating Resources and Checking Actions

```
test@test-test-instance-tcnzvnnxxuy2z:~/node_pack$ npm start
> node_pack@1.0.0 start /home/test/node_pack
> node index.js

server running at port 8080
```
Figure 10

Virtual Machine Creation Steps

```
Template Verification → Resource Initialization → Resource Configuration
```

Resource Binding Time

Figure 11

Behavior during Resource Binding Time

```
Empty Virtual Machine Creation → Operating System Initialization → Operating System Configuration
```

Virtual Machine Booting Time

Figure 12
Behavior at Virtual Machine Booting Time

Figure 13

Step-by-Step Comparison of CPU Usage

Figure 13

Step-by-Step Comparison of Memory Usage
Figure 14
Step-by-Step Comparison of Memory Usage

Figure 15
Resource Binding Time Comparison
Figure 16

Virtual Machine Booting Time Comparison
Figure 17

Resource Binding Time Average CPU Usage
Figure 18

Resource Binding Time Average Memory Usage
Figure 19

Virtual Machine Booting Time Average CPU Usage
Figure 20

Virtual Machine Booting Time Average Memory Usage