# Research on Application Migration Models Between Heterogeneous Container Clouds (Post-print)

**Authors:** Yang Kaiqi, Zhao Yulong, Chen Lin

**Date:** 2019-01-28T00:00:00+00:00

## Abstract

To address the challenge of migrating Docker container-based applications across heterogeneous container clouds, this paper investigates the orchestration principles of mainstream heterogeneous container orchestration engines and examines the heterogeneity among leading container service providers. Based on these investigations, a three-layer model for cross-heterogeneous-container-cloud application migration is proposed. To enhance migration efficiency, an application migration technique based on image pre-synchronization is further introduced. Experimental results demonstrate that the three-layer model successfully enables application migration across heterogeneous container clouds, and that with the introduction of the image pre-synchronization technique, the average application migration time between homogeneous-cloud heterogeneous container orchestration engines is reduced by 60.33%, while the average migration time between heterogeneous-cloud Kubernetes clusters is reduced by 43.67%.

## Full Text

## Preamble

**Vol. 37 No. 4**
*Application Research of Computers*

**Study on Migration of Applications Between Heterogeneous Container Clouds**

*Yang Kaiqi, Zhao Yulong, Chen Lin*
(Institute of Jiangnan Computing Technology, Wuxi, Jiangsu 214000, China)

**Abstract:** To address the challenge of migrating Docker container-based applications across heterogeneous container clouds, this paper investigates the orchestration principles of mainstream heterogeneous container orchestration en-

gines and examines the heterogeneity among leading container service providers. Building upon this foundation, we propose a three-layer model for migrating Docker container-based applications across heterogeneous container clouds. To improve migration efficiency, we introduce an image pre-synchronization technique for application migration. Experimental results demonstrate that the three-layer model successfully enables cross-cloud migration of applications, with the image pre-synchronization technology reducing migration time by an average of 60.33% between heterogeneous container orchestration engines in homogeneous clouds, and by 43.67% between Kubernetes clusters in heterogeneous clouds.

**Keywords:** Docker containers; application migration; three-layered model; heterogeneous container clouds

## 0 Introduction

Enterprises inevitably operate diverse heterogeneous technologies. The current cloud computing market exhibits vibrant competition, with international providers such as Amazon, Microsoft, and Google, and domestic providers including Alibaba, Huawei, and Meituan. To leverage proximity computing advantages, enhance global service capabilities, and avoid vendor lock-in, enterprises must deploy their applications across multiple geographic regions and data centers from different cloud providers. However, cross-domain data transmission incurs significant costs: it consumes substantial time and precious network bandwidth resources while offering low cost-effectiveness, and certain security policies prohibit cross-domain data transfer altogether. Consequently, both academia and industry advocate for application migration between heterogeneous container clouds.

Nevertheless, migrating applications across heterogeneous container clouds presents numerous challenges: (a) for complex applications, maintaining interaction with orchestration engines during migration to preserve health check mechanisms, node affinity, and other orchestration rules; (b) ensuring that Docker container-based applications can operate independently of storage and network environments before and after migration; (c) minimizing the total migration time for Docker container-based applications; and (d) preventing frequent migration operations from rapidly increasing data center energy consumption and network bandwidth usage, which could impact daily operations.

Current multi-data center container management primarily relies on cloud federation models, such as Kubernetes cluster federation, HashiCorp Nomad, and Mesos cluster federation. While these approaches manage containers across multiple data centers, they are limited to homogeneous container orchestration engines and cannot schedule resources between heterogeneous orchestrators, thus

preventing the migration of Docker container-based applications between Kubernetes clusters and Docker Swarm clusters.

Existing solutions exhibit significant limitations. The log-replay-based container migration method proposed in [4] suffers from increasing migration time proportional to container runtime, making it impractical for long-running enterprise applications, and it only works in homogeneous container cloud environments. The Checkpoint-Restart technique employed in [5,6] requires identical file systems and network configurations between source and destination, rendering it unsuitable for cross-provider, cross-region multi-data center migrations. The CRIU team, a leading research group in Docker container migration technology, adopts a similar approach to CR technology [8]. The three-layer application migration model in [7] divides the migration process into Base Layer, Application Layer, and Instance Layer. Although this model ensures file system consistency at the Base Layer, it fails to address heterogeneity between container clouds, making post-migration application takeover impossible. Consequently, the methods proposed in [4-7] cannot solve the problem of migrating Docker container-based applications between heterogeneous container clouds.

---

## 1 Three-Layer Model TMHCC for Migrating Docker Container-Based Applications Across Heterogeneous Container Clouds

### 1.1 Definition of Heterogeneous Container Clouds

Heterogeneous container clouds encompass two scenarios: homogeneous clouds with heterogeneous container orchestration engines, and heterogeneous clouds with homogeneous container orchestration engines. The former refers to cloud platforms with identical infrastructure but different container orchestration engines, while the latter describes cloud platforms with different infrastructures but the same orchestration engine. For instance, if two data centers both use Alibaba Cloud' s container service with identical storage and network types, but one employs Kubernetes while the other uses Docker Swarm, this constitutes a homogeneous cloud with heterogeneous orchestration engines. Conversely, if one data center uses Alibaba Cloud' s container service and another uses Huawei Cloud' s service—exhibiting heterogeneity in networking and storage—but both utilize Kubernetes, this represents a heterogeneous cloud with homogeneous orchestration engines.

### 1.2 Three-Layer Model

The three-layer model for migrating Docker container-based applications across heterogeneous container clouds is illustrated in [Figure 1: see original paper]. This model decomposes cross-cloud migration into three distinct layers: the image layer, orchestration layer, and application layer.

[Figure 1: see original paper]

**1.2.1 TMHCC Image Layer** In Docker, container images serve as the foundation for containers [9]. Therefore, ensuring that both source and destination data centers possess the required image files is critical when migrating Docker container-based applications. The three-layer model synchronizes image repositories between source and destination data centers at the image layer. Docker, Kubernetes, and major cloud providers such as Huawei and Alibaba all adhere to the OCI specification, ensuring identical container image formats that require no conversion during synchronization.

**1.2.2 TMHCC Orchestration Conversion Technology** The orchestration layer synchronizes orchestration templates based on the characteristics of source and destination data centers. Application orchestration templates comprise definitions of container services and their interdependencies, enabling deployment and management of multi-container applications. Docker Swarm supports Docker Compose files for application orchestration and deployment, while Kubernetes clusters support YAML or JSON formats for defining various resource objects, as well as Helm for simplified application deployment. For migrations between heterogeneous orchestration engines in homogeneous clouds, the orchestration layer synchronizes templates via SwarmNetes; for migrations between Kubernetes clusters in heterogeneous clouds, it synchronizes orchestration information through HelmConvert.

For application migration between Docker Swarm and Kubernetes clusters within homogeneous container clouds, we designed and developed SwarmNetes to enable orchestration information synchronization. The SwarmNetes architecture is shown in [Figure 2: see original paper]. SwarmNetes converts between Docker Compose files (v1, v2, and v3) and six Kubernetes resource objects: ConfigMap, Service, ReplicationController, DaemonSet, Deployment, Pod, and PersistentVolumeClaim.

We propose the TMHCC (Three-layer Model for Heterogeneous Container Clouds) for migrating Docker container-based applications across heterogeneous container clouds. This model applies to both homogeneous clouds with heterogeneous orchestration engines and heterogeneous clouds with Kubernetes clusters. Introducing image pre-synchronization technology effectively improves migration efficiency.

[Figure 2: see original paper]

The dashed line in [Figure 2: see original paper] illustrates the conversion process when migrating applications from Kubernetes to Docker Swarm clusters. SwarmNetes extracts application information from input Kubernetes orchestration files, stores it in the intermediate representation layer as SwarmNetes objects, converts these objects into Swarm objects recognizable by Docker Swarm clusters, and finally outputs Docker Swarm orchestration files. The solid line de-

picts the reverse migration process. The SwarmNetes orchestration conversion algorithm is presented in Algorithm 1.

**Algorithm 1 SwarmNetes Orchestration Information Conversion Algorithm**

1. SwarmNetesObject = LoadFile(InputFiles);
2. if Provider == Kubernetes then
3. KubernetesObject = SwarmNetes.ConvertToKubernetes(SwarmNetesObject);
4. SwarmFile = Kubernetes.Output(SwarmObject);
5. else
6. SwarmObject = SwarmNetes.ConvertToSwarm(SwarmNetesObject);
7. KubernetesFile = Swarm.Output(KubernetesObject);
8. end if

Heterogeneity in storage and networking between heterogeneous clouds creates invisible barriers for migrating Docker container-based applications between Kubernetes clusters. If an application uses provider-specific storage or networking, it may fail to start in the destination data center, which cannot recognize these resource types. For migrations between Kubernetes clusters in heterogeneous container clouds, we designed HelmConvert to transform source Chart templates into destination-compatible Chart templates.

The HelmConvert architecture is shown in [Figure 3: see original paper]. The input layer reads an application' s Chart template and processes its contents, such as rendering configurations from the values.yaml file into resource objects defined in the templates folder. The recognition layer identifies the results based on the cloud provider type (e.g., "Huawei" or "Ali" ) and their container service characteristics to determine the storage and network types used in the input Chart template. The conversion layer performs transformations using our heterogeneous storage and network conversion methods, producing Chart templates compatible with the destination provider. The output layer generates the converted Chart template.

[Figure 3: see original paper]

Huawei Cloud Container Service supports EVS (Elastic Volume Service), file storage, and OBS (Object Storage Service), while Alibaba Cloud Container Service supports cloud disks, NAS (Network Attached Storage), and OSS (Object Storage Service). Although implementation details vary, both providers support storage resources through PV (PersistentVolume) and PVC (PersistentVolumeClaim). Kubernetes provides these API resources to abstract storage details: cluster administrators focus on providing storage functionality via PVs, while users simply mount PVCs to containers without concerning themselves with technical implementations. This abstraction, analogous to the relationship between Pods and Nodes, enables transparent storage heterogeneity and facilitates conversion between different storage types. HelmConvert performs mutual conversions between Huawei Cloud EVS and Alibaba Cloud disks, Huawei file storage and Alibaba NAS, and Huawei OBS and Alibaba OSS.

Container services provide diverse networking access methods to satisfy complex inter-application communication scenarios, including intra-cluster access, VPC access, Layer-4 load balancing, and Layer-7 load balancing. The first two are private network accesses, while the latter two are public. Our research reveals that for intra-cluster access, both Huawei Cloud and Alibaba Cloud follow native Kubernetes standards, enabling seamless migration. For the other three access methods, HelmConvert identifies network types through characteristic attributes and performs appropriate conversions.

**1.2.3 TMHCC Application Layer** Docker container-based applications depend on image files and orchestration information files. After synchronizing images at the image layer and orchestration information at the orchestration layer, the destination data center possesses all essential application components. The application layer simply launches the application using these files. For homogeneous clouds with heterogeneous orchestration engines, applications are created in Kubernetes clusters via `kubectl create -f [file]` or in Docker Swarm clusters via `docker-compose up` or `docker stack deploy --compose-file=filename`. For heterogeneous cloud Kubernetes clusters, Helm simplifies application management through the `helm install` command.

### 1.3 Image Pre-Synchronization-Based Application Migration Technology

We refer to migration methods without image pre-synchronization as traditional methods. As shown in [Figure 4: see original paper], traditional methods involve five steps: (a) converting the source orchestration file into a destination-compatible format and synchronizing it; (b) attempting to pull dependent images from the destination repository during application creation; (c) if images are absent, synchronizing them from the source repository; (d) pulling images from the destination repository; and (e) creating and running the application.

The total migration time T for traditional methods is given by Equation (1):

$$T_1 = T_{\text{OrcSync}} + T_{\text{ImageSync}} + T_{\text{AppRun}}$$

where $T_{\text{OrcSync}} = T_{\text{OrcConvert}} + T_{\text{OrcRep}}$. $T_{\text{OrcConvert}}$ represents the time for SwarmNetes or HelmConvert to transform orchestration information, while $T_{\text{OrcRep}}$ is the synchronization time for the converted orchestration file. $T_{\text{ImageSync}}$ denotes image synchronization time between repositories, and $T_{\text{AppRun}}$ is the application creation time.

[Figure 4: see original paper]

Image pre-synchronization technology introduces an Image Pre-Synchronization (IPS) module in the source data center, as shown in [Figure 5: see original paper]. The IPS module continuously monitors for new image creation, immediately

synchronizing any newly built images to the destination repository if they are absent.

[Figure 5: see original paper]

The image pre-synchronization-based migration process involves: (a) real-time detection and synchronization of new images; (b) orchestration file conversion and synchronization; (c) image pulling during application creation; (d) fallback image synchronization if needed; (e) image pulling; and (f) application creation and execution.

The total migration time T is given by Equation (3):

$$T_2 = T_{\text{OrcSync}} + T_{\text{AppRun}}$$

Since image synchronization dominates migration time, the image pre-synchronization technology substantially reduces total migration time for cross-cloud Docker container-based applications.

---

## 2 Experiments and Analysis

The image pre-synchronization-based three-layer model extends the TMHCC model with image pre-synchronization technology. We conducted experiments using various Docker container-based applications and compared results before and after introducing this optimization.

### 2.1 Homogeneous Cloud with Heterogeneous Container Orchestration Engines

To validate feasibility and effectiveness between Docker Swarm and Kubernetes clusters (homogeneous cloud, heterogeneous orchestration), we selected four representative applications:

a) **Counter**: Tracks database access counts and displays them on a web page. Depends on `redis:3.0` (91.5MB) and `tuna/docker-counter23:latest` (696.0MB).

b) **Wordpress**: A personal blogging system using PHP and MySQL. Depends on `wordpress:4.8-apache` (408.0MB) and `mysql:5.6` (256.0MB).

c) **Gitlab**: A repository management system with Git-based code management and web services. Depends on `redis:latest` (133.0MB), `postgresql:9.5` (232.0MB), and `gitlab:8.13` (770MB).

d) **Vote**: A voting application for Dogs vs. Cats with web-based results. Depends on five images: `redis:alpha` (27.8MB), `postgres:9.4` (263.0MB), `docker/example-voting-app-worker` (574.0MB), `docker/example-voting-app-vote` (83.5MB), and `tmadams333/example-voting-app-result` (226.0MB).

**2.1.1 Experimental Environment** We utilized OpenStack Pike to establish both Kubernetes and Docker Swarm environments. As shown in [Figure 6: see original paper], the Kubernetes cluster comprised a Master node (10.33.122.77), Node 1 (10.33.122.86), and Node 2 (10.33.122.81), with configurations detailed in . The Docker Swarm cluster included a Manager node (10.33.35.35), Node 1 (10.33.35.34), and Node 2 (10.33.35.19), with configurations in . All nodes were connected via 1000M network bandwidth.

[Figure 6: see original paper]

We migrated the four applications from Kubernetes to Docker Swarm using both traditional and pre-synchronization methods. Traditional migration times were 121.71s, 117.46s, 193.25s, and 185.10s, respectively, while the pre-synchronization model required only 46.97s, 54.28s, 62.06s, and 82.08s.

Subsequent experiments migrating from Docker Swarm to Kubernetes yielded similar improvements: traditional methods took 128.34s, 120.25s, 184.02s, and 191.22s, whereas the pre-synchronization model completed in 45.72s, 53.32s, 62.73s, and 79.52s.

Results in [Figure 7: see original paper] and [Figure 8: see original paper] demonstrate that image synchronization dominates migration time. The image pre-synchronization model reduces migration time between Docker Swarm and Kubernetes clusters by an average of 60.33%.

[Figure 7: see original paper]
[Figure 8: see original paper]

### 2.2 Heterogeneous Cloud Kubernetes Clusters

To validate the model between Huawei Cloud and Alibaba Cloud Kubernetes clusters (heterogeneous cloud, homogeneous orchestration), we selected Nginx—a high-performance HTTP server, reverse proxy, and email proxy developed by Igor Sysoev.

We designed three Nginx variants to cover all storage and network heterogeneity scenarios: Nginx1 uses Huawei EVS storage and VPC access (Huawei) versus Alibaba Cloud disk and VPC access (Alibaba); Nginx2 uses Huawei file storage with Layer-4 load balancing versus Alibaba NAS with Layer-4 load balancing; Nginx3 uses Huawei OBS with Layer-7 load balancing versus Alibaba OSS with Layer-7 load balancing. All variants depend on the `nginx:latest` image (109MB).

**2.2.1 Experimental Environment** We established Kubernetes clusters on both Huawei and Alibaba Cloud container services, as shown in [Figure 9: see original paper]. Data Center 1 (Beijing) on Huawei Cloud included one Master (192.168.0.193) and two Workers (192.168.0.200, 192.168.0.202) with a private registry (192.168.0.204). Data Center 2 (Shenzhen) on Alibaba Cloud featured

three Masters (192.168.0.195-197) and three Workers (192.168.0.198-200) with a private registry (192.168.0.200). Node configurations are detailed in and .

[Figure 9: see original paper]

**2.2.2 Experimental Results** Migrating Nginx1, Nginx2, and Nginx3 from Huawei to Alibaba Cloud required 28.16s, 24.13s, and 25.96s using traditional methods, but only 13.72s, 12.76s, and 12.95s with the pre-synchronization model [Figure 10: see original paper].

Reverse migrations from Alibaba to Huawei Cloud took 41.33s, 33.81s, and 37.91s traditionally, versus 25.68s, 22.71s, and 21.68s with pre-synchronization [Figure 11: see original paper].

Results demonstrate that image synchronization constitutes the primary migration bottleneck. The pre-synchronization model reduces migration time between Huawei and Alibaba Kubernetes clusters by an average of 43.67%.

[Figure 10: see original paper]
[Figure 11: see original paper]

---

## 3 Conclusion

To address the need for migrating Docker container-based applications across heterogeneous container clouds, we proposed the TMHCC three-layer model. We developed SwarmNetes at the orchestration layer to shield heterogeneity between orchestration engines, enabling migration between Docker Swarm and Kubernetes in homogeneous clouds. We also created HelmConvert to mask provider-specific heterogeneity, facilitating migration between Kubernetes clusters in heterogeneous clouds. Additionally, we introduced image pre-synchronization technology to optimize migration efficiency. Future work should focus on reducing network bandwidth consumption from the three-layer model to prevent operational disruptions during frequent inter-data center transfers. As Docker container technology continues evolving, with ongoing improvements to orchestration engines and cloud provider services, we must continuously monitor these developments and adapt our model to accommodate emerging features.

## References

[1] AbdelBaky M, Diaz-Montes J, Parashar M, et al. Docker containers across multiple clouds and data centers [C]//Proc of the 8th IEEE/ACM International Conference on Utility and Cloud Computing. New York: ACM Press, 2015: 368-371.

[2] Kumar K M, Kumar D S, Murudi V. Multi Data Center Cloud Cluster Federation-Major Challenges & Emerging Solutions [C]//Proc of IEEE Interna-

tional Conference on Cloud Computing in Emerging Markets. Piscataway, NJ: IEEE Press, 2016: 107-122.

[3] Gupta V, Chaunhan S, Sharma D. Platform virtualization: understanding virtual machines, LXC, Docker, Kubernetes and Ubernetes [J]. International Journal of Innovations in Engineering and Technology. 2016, 7(6): 442-447.

[4] Yu C Y, Huan F. Live migration of docker containers through logging replay [C]//Proc of the 3rd International Conference on Mechatronics and Industrial Informatics. Paris, France: Atlantis Press, 2015: 623-626.

[5] Mirkin A, Kuznetsov A, Kolyshkin K. Containers checkpointing and live migration [C]//Proc of Linux Symposium. 2008: 85-90.

[6] Laadan O, Hallyn S E. Linux-CR transparent application checkpoint-restart in Linux [C]//Proc of the Linux Symposium. 2010.

[7] Machen A, Wang S Q, Leung K K, et al. Live service migration in mobile edge clouds [J]. IEEE Wireless Communications. 2018, 25(2).

[8] CRIU [EB/OL]. [2018-08-28]. https://criu.org/Usage_scenarios#Container_live_migration.

[9] 浙江大学 SEL 实验室. Docker 容器与容器云 [M]. 2 版. 北京: 人民邮电出版社, 2016.

[10] Kubernetes [EB/OL]. [2018-09-21]. https://kubernetes.io/.

[11] Docker Swarm [EB/OL]. [2018-09-21]. https://docs.docker.com/.

[12] 毛祺, 卢胜林. 基于 Docker Swarm 集群的容器迁移策略的实现 [J]. 信息技术, 2016(9): 156-160.

[13] 卢胜林, 倪明, 张翰博. 基于 Docker Swarm 集群的调度策略优化 [J]. 信息技术, 2016(7): 147-151.

*Note: Figure translations are in progress. See original paper for figures.*