Ready to Use Virtual Machine Pool Cache using Warm Cache

Sudeep Kumar, Deepak Kumar Vasthimal, and Musen Wen

eBay Inc., 2025 Hamilton Ave, San Jose, CA 95125, USA
{sudekumar, dvasthimal, mwen}@ebay.com

Abstract. Today, a plethora of distributed applications are managed on internally hosted cloud platforms. Such managed platforms are often multi tenant by nature and not specifically tied to a single use-case. Smaller footprint of infrastructure on a managed cloud platform has its own set of challenges especially when applications are required to be infrastructure aware for quicker deployments and response times. There are often times and challenges to quickly spawn ready to use instances or hosts on such infrastructure. As part of this paper we outline mechanisms to quickly spawn ready to use instances for application while also being infrastructure aware. In addition, paper proposes architecture that provides high availability to deployed distributed applications.

Keywords: cloud computing, virtual machine, elastic, elastic search, consul, cache, java, kibana, mongoDB, high performance computing, architecture.

1 Introduction

Conventional on-demand Virtual Machine (from now referred as VM [7] or VMs) provisioning methods on a cloud [12] platform can be time-consuming and error-prone, especially when there is need to provision [25] VMs in large numbers swiftly.

The following list captures different issues that are often encountered while trying to provision a new VM [7] instance on the fly.

- Insufficient availability of compute resources due to capacity constraints.
- Desire to place VMs on different fault domains to avoid concentration of VM [7] instances in the same rack [11] and that eventually leads to non-availability of deployed applications over them.
- Transient failures or delays in the service provider platform result in failure or an increase in time to provision a VM [7] instance.

The proposed Elasticsearch-as-a-service platform for VM [7] provisioning is a cloud-based platform that provides distributed, easy to scale, and fully managed on demand Elasticsearch [1] clusters. This platform uses the OpenStack [23] based Nova module to get different compute resources (VMs). Nova is designed to power massively scalable, on-demand, self-service access to compute resources. The SAAS

David C. Wyld et al. (Eds): CCSEA, BIoT, DKMP, CLOUD, NLCAI, SIPRO - 2020
pp. 85-99, 2020. CS & IT - CSCP 2020 DOI: 10.5121/csit.2020.101008

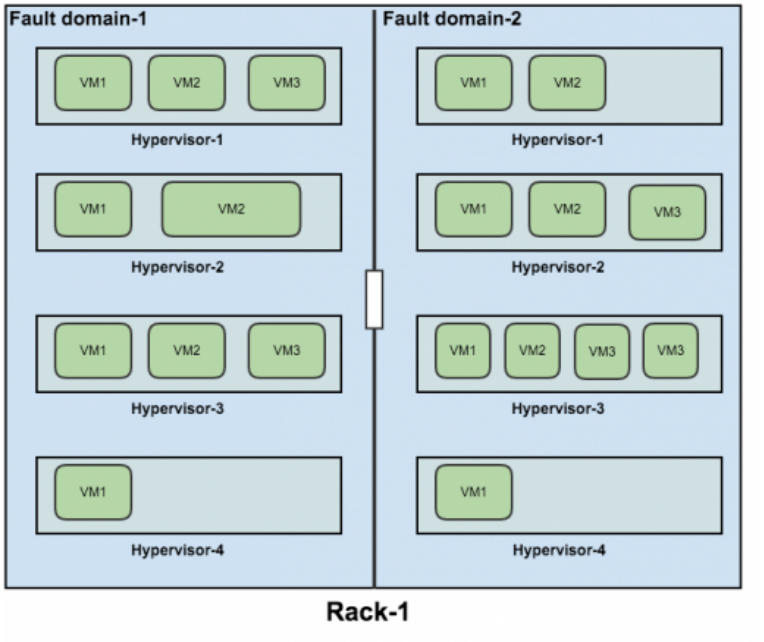

Fig. 1. Rack-1

(Software as a service) [18] platform is available across multiple data centers with ability to manage massively large number of managed VMs.

Typically, the time taken for provisioning a complete Elasticsearch [1] cluster via Nova APIs is directly proportional to the largest time taken by the member node to be in a ready to use state (also known as active state). Typically, provisioning a single node could take up to three minutes (95th Percentile) but can be up to 15 minutes in worst case scenario. Therefore, in a fairly large size cluster, proposed platform would take a long time for complete provisioning of an entire VM [7] farm. This greatly impacts the turnaround time to remediate production issues. In addition to provisioning time, it is time-consuming to validate newly created VMs.

There are many critical applications within eBay that leverage proposed platform for their search use cases. Therefore, as a platform provider, high availability and resiliency [15] to failures are of utmost importance to ensure that in a case of catastrophic cluster event (such as a node or an infrastructure failure), the system can quickly flex up provisioned clusters in seconds. Node failures are also quite common in a cloud-centric world [17], and applications need to ensure that there is sufficient resiliency built in. To avoid over-provisioning nodes, remediation actions such as flex-up (adding a new node) should ideally be done in seconds to ensure high availability. Flex-up also ensures that distributed applications are resilient [15] to node failures. Warm cache is targeted to solve issues mentioned above by

creating a VM instances cache from which VM will used for cluster provisioning. The cache will be created by leveraging various internal systems/services such as NOVA, CONSUL,CMS.

New hardware capacity is acquired as racks from external vendors are procured and provisioned. Each rack [11] typically has two independent fault domains with minimal resource overlap (For example, different networks), and sometimes they dont share a common power source. Each fault domain hosts many hypervisors [6], which are virtual machine managers. Standalone VMs are provisioned on such hypervisors [6]. VMs can be of different sizes (tiny, medium, large, and so on). VMs on the same hypervisor [6] can compete for disk and network I/O resources, and therefore can lead to noisy neighbor issues.

Nova provides ways to be fault-aware and hypervisor-aware. However, it is still difficult to successfully achieve guaranteed rack [11] isolation during run-time provisioning of VM [7] instances. For example, once we start provisioning VMs, there is no guarantee that we will successfully create VM [7] instances on different racks. This depends entirely on the underlying available hardware at that point in time. Rack [11] isolation is important to ensure high availability of Elasticsearch [1] master nodes (cluster brain). Every master node in an Elasticsearch [1] cluster must reside on a different rack [11] for fault tolerance. If a rack fails, at least some other master node in another rack [11] can take up active master role. Additionally, all data nodes of a given cluster must reside on different hypervisors [6] for logical isolation. Proposed platform's VM [7] provisioning APIs must fail immediately when we cannot get VMs on different racks or hypervisors [6]. A subsequent retry will not necessarily solve this problem.

2 Solution

The warm-cache module intends to solve these issues by creating a cache pool [21] of VM [7] instances well ahead of actual provisioning needs. Many pre-baked VMs are created and loaded in a cache pool [21]. These ready-to-use VMs cater to the cluster provisioning needs of the Software As A Service (SAAS) platform. The cache is continuously built, and it can be continuously monitored via alerts and user-interface (UI) dashboards. Nodes are periodically polled for health status, and unhealthy nodes are auto-purged from the active cache. At any point, interfaces on warm-cache can help tune or influence future VM [7] instance preparation.

The image in 2 shows a sample Consuls [2] web UI.

The warm-cache module leverages open source technologies like Consul [2], Elasticsearch [1], Kibana [4], Nova [5], and MongoDB for realizing its functionality.

Consul [2] is an open-source distributed service discovery tool and key value store. Consul [2] is completely distributed, highly available, and scalable to thousands of nodes and services across multiple data centers. Consul [2] also provides distributed locking mechanisms with support for TTL (Time-to-live).

Fig. 2. Consul Web Interface

Figure 3 shows a representative warm-cache KV store in Consul [2].
We use Consul [2] as key-value (KV) store for these functions:

- Configuring VM build rules: VM build rules uses different instance templates. These instance templates are well-known or predefined. For Example: 'g16highmem' instance would imply 16 VCPUs, 100GB RAM and 1 TB solid state storage. The number of instances is configured as rules against every instance template.
- Storing VM flavor configuration metadata [20]: To ensure availability, multiple instances are started. These instances are responsible for creation of warm caches. To avoid, multiple instances working on the same set of rules at any given time and ensure mutual exclusion of work, leader election via distributed locks is used that is provided by consul.
- Leader election [13] (via distributed locks)
- Persisting VM-provisioned information: Once these instances are created, metadata information relating to provisioned instances must be persisted as this information is looked up during during user-initiated provisioning requests.

3 Architecture

Elasticsearch [1] is a highly scalable open-source full-text search and analytics engine. It allows you to store, search, and analyze big volumes of data quickly and in near real time. It is generally used as the underlying engine/technology that powers applications that have complex search features and requirements. Apart from provisioning and managing Elasticsearch [1] clusters for our customers, we ourselves use Elasticsearch [1] clusters for our platform monitoring [3] needs. This is a good

```
{
    "instance": "                                                    ",
    "flavour": "g2-highmem-16-     ",
    "hypervisor_id": "805c4721cc9a0284c825b13faec4d6d27723a23926310286c4a58644",
    "rack_id": "CHD02-02-500-0204-1",
    "server_id": "eac9b928-9f41-40e4-bfd9-d04694669a18",
    "group_name": "group-29",
    "tenant_name": "b943c795dfd047478c2a8af823ab380a",
    "compute_url": "                                                    ",
    "   ": "      ",
    "created_time": 1468568710881,
    "image_id": "ca343bc8-cf74-4072-b383-61537c9e4bae"
}
```


Fig. 3. Warm Cache KV Store in Consul

way to validate our own platform offering. Elasticsearch [1] backend is used for warm-cache module monitoring [3].

Kibana [4] is built on the power of Elasticsearch [1] analytics capabilities to analyze your data intelligently, perform mathematical transformations, and slice and dice your data as you see fit. We use Kibana [4] to depict the entire warm-cache build history stored in Elasticsearch [1]. This build history is rendered on Kibana [4] dashboard with various views. The build history contains information such as how many instances were created and when were they created, how many errors had occurred, how much time was taken for provisioning, how many different Racks are available, and VM [7] instance density on racks/hypervisors. Warm-cache module can additionally send email notifications whenever the cache is built, updated, or affected by an error.

We use the Kibana [4] dashboard to monitor active and ready-to-use VM instances of different flavors in a particular datacenter, as shown in 4.

MongoDB is an open-source, document database designed for ease of development and scaling. warm-cache uses this technology to store information about flavor details. Flavor corresponds to the actual VM-underlying hardware used. (They can be tiny, large, xlarge, etc.). Flavor details consist of sensitive information such as image-id, flavour-id, which are required for actual Nova [5] compute calls. warm-cache uses a Mongo [9] service abstraction layer (MongoSvc) to interact with the backend MongoDB [9] in a secure and protected manner. The exposed APIs on MongoSvc are authenticated and authorized via Keystone integration.

CMS (Configuration Management System) is a high-performance, metadata-driven persistence and query service for configuration data with support for RESTful [16] API and client libraries (Java and Python). This system is internal to eBay,

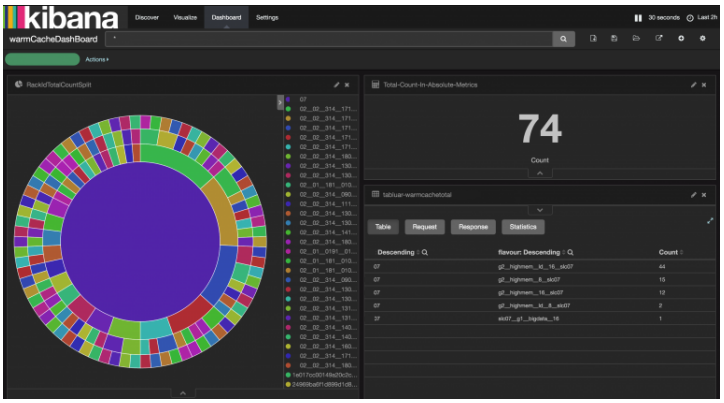

Fig. 4. Kibana Dashboard

and it is used by warm-cache to get hardware information of various compute nodes (including rack [11] and hypervisor [6] info).

4 System Design

The warm-cache module is built as a plug-gable library that can be integrated or bundled into any long running service or daemon process. On successful library initialization, a warm-cache instance handle is created. Optionally, a warm-cache instance can enroll for leader election [13] participation. Leader instances are responsible for preparation of VM cache pools for different flavors. warm-cache will consist of all VM pools for every flavor across the different available data centers. The figure 5 the system design of warm-cache.

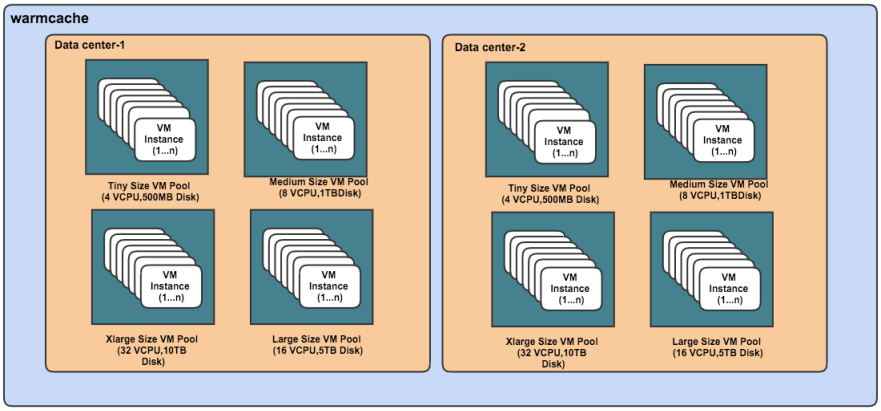

Fig. 5. System Design

The figure 6 depicts the system dependencies of warm-cache.

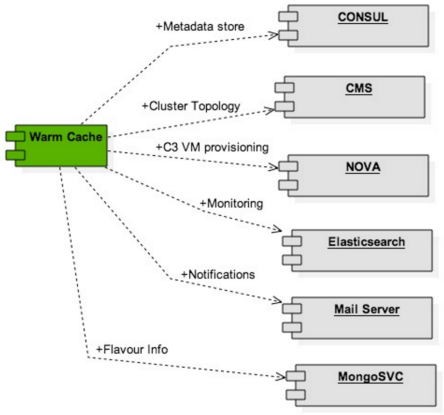

Fig. 6. System Dependency

The warm-cache module is expected to bring down VM instance preparation time to few seconds. It should also remedy a lot of exceptions and errors that occur while VM instances get ready to a usable state, because these errors are handled well in advance of actual provisioning [25] needs. Typical errors that are encountered today are nodes not available in Foreman due to sync issues and waiting for VM instances to get to the active state.

The figure 7 below depicts the internal state diagram of the warm-cache service. This state flow is triggered on every warm-cache service deployed. Leader election [13] is triggered at every 15-minute boundary interval (which is configurable).

This leader election is done via Consul [2] which locks with an associated TTL (Time-to-live). After a leader instance is elected, that particular instance holds the leader lock and reads metadata from Consul [2] for each Availability Zone (AZ, equivalent to a data center). These details include information such as how many minimum instances of each flavor are to be maintained by warm-cache. Leader instance spawns parallel tasks for each availability zone (AZ) and starts preparing the warm cache based on predefined rules. Preparation of a VM instance is marked as complete when the VM instance moves to an active state (for example, as directed by an open-stack Nova [5] API response). All successfully created VM instances are persisted on an updated warm-cache list maintained on Consul [2]. The leader instance releases the leader lock on the complete execution of its VMs build rules and waits for next leader election [13] cycle. The configuration of each specific

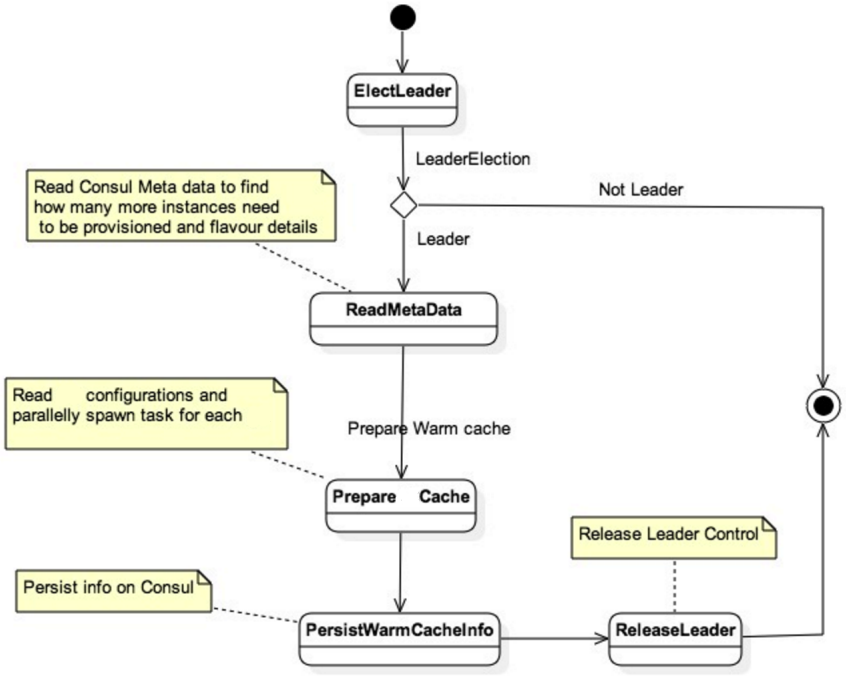

Fig. 7. System-State Diagram

flavor (for example, g2-highmem-16) is persisted in Consul [2] as build rules for that particular flavor. The figure 8 shows an example.


```
{
    "max_instances_per_cycle": 5,
    "min_fault_domain": "2",
    "reserve_cap": 10,
    "group_hint": null,
    "compute_url": "                                                    ",
    "user_data": "IyEvYmluL2Jhc2gNCnB1cHBldCBhZ2VudCAtLXRlc3Q                    ",
    "total_instance_count": 12
}
```


Fig. 8. Sample Rule

In above sample rule, the max_instance_per_cycle attribute indicates how many instances are to be created for this flavor in one leadership cycle. min_fault_domain is used for the Nova [5] API to ensure that at least two nodes in a leader cycle go to different fault domains. reserve_cap specifies the number of instances that will be blocked and unavailable via warm-cache. user_data is the base64-encoded Bash

script that a VM instance executes on first start-up. total_instances keeps track on total number of instances that need to be created for a particular flavor. An optional group_hint can be provided that ensures that no two instances with the same group-id are configured on the same hypervisor [6].

For every VM instance added to warm-cache, following metadata is persisted to Consul [2]:

- Instance Name
- Hypervisor ID
- Rack ID
- Server ID
- Group name (OS scheduler hint used)
- Created time

Since there are multiple instances of the warm-cache service deployed, only of them is elected leader to prepare the warm-cache during a time interval. This is necessary to avoid any conflicts among multiple warm-cache instances. Consul [2] is again used for leader election [13]. Each warm-cache service instance registers itself as a warm-cache service on Consul [2]. This information is used to track available warm cache instances. The registration has a TTL (Time-To-Live) value (one hour) associated with it. Any deployed warm cache service is expected to re-register itself with the warm-cache service within the configured TTL value (one hour). Each of the registered warm-cache services on Consul [2] attempts to become to elect itself as a leader by making an attempt to acquire the leader lock on Consul [2]. Once a warm-cache service acquires a lock, it acts as a leader for VM cache pool [21] preparation. All other warm-cache service instances move to a stand-by mode during this time. There is a TTL associated with each leader lock to handle leader failures and to enable leader reelection.

In the figure 9, leader is a Consul [2] key that is managed by a distributed lock [10] for the leadership role. The last leader node name and leader start timestamp are captured on this key. When a warm-cache service completes it functions in the leader role, this key is released for other prospective warm-cache service instances to become the new leader.

The leadership time-series graph 10 depicts which node assumed the leadership role. The number 1 in the graph below indicates a leadership cycle.

When a leader has to provision a VM instance for a particular flavor, it first looks up for meta information for the flavor on MongoDB [9] (via MongoSvc). This lookup provides details such as image-Id and flavor-Id. This information is used when creating the actual VM instance via Nova [5] APIs. Once a VM is created, its rack-id information is available via CMS. This information is stored in Consul [2] associated with a Consul [2] key $AZ/$INSTANCE, where $AZ is the Availability Zone and$INSTANCE is the actual instance name. This information is also then persisted on Elasticsearch [1] for monitoring [3] purpose.

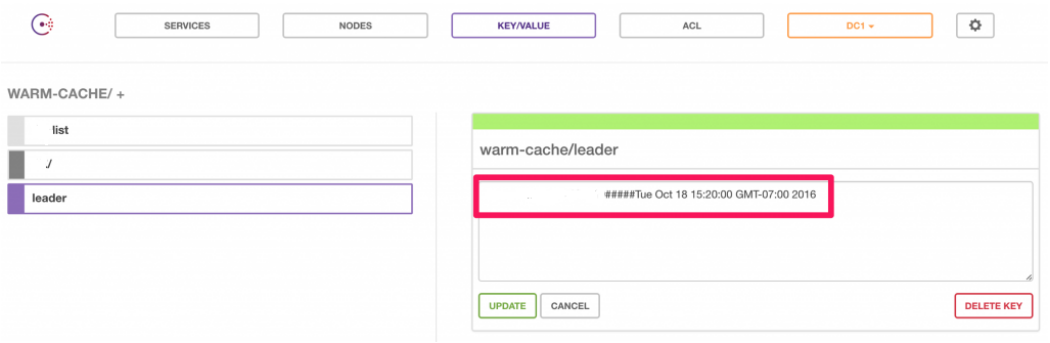

Fig. 9. Leader

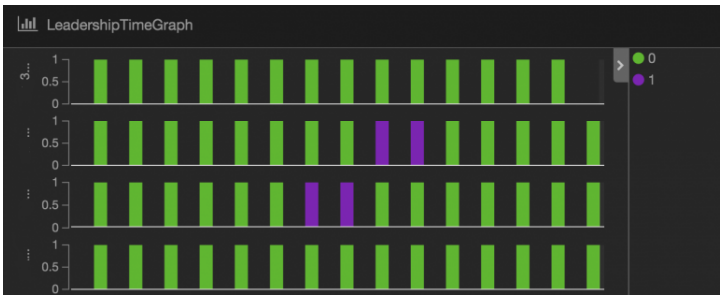

Fig. 10. Leadership time-series graph

The figure 11 shows a high-level system sequence diagram [22] (System Sequence Diagram) of a leader role instance:

A Kibana [4] dashboard as shown in figure 12 can be used to check how VM [7] instances in the cache pool are distributed across available racks. The following figure shows how many VM [7] instances are provisioned on each rack. Using this information, Dev-ops can change the warm-cache build attributes to influence how the cache should be built in future.

The following options are available for acquiring VM instances from the warm-cache pool:

- The Rack-aware mode option ensures that all nodes provided by warm-cache reside on different racks.
- The hypervisor-aware mode option returns nodes that reside on different hypervisors [6] with no two nodes sharing a common hypervisor [6].
- The Best-effort mode option tries to get nodes from mutually-exclusive hypervisors [6] but does not guarantee it.

The figure 13 illustrates the sequence diagram [22] for acquiring a VM.

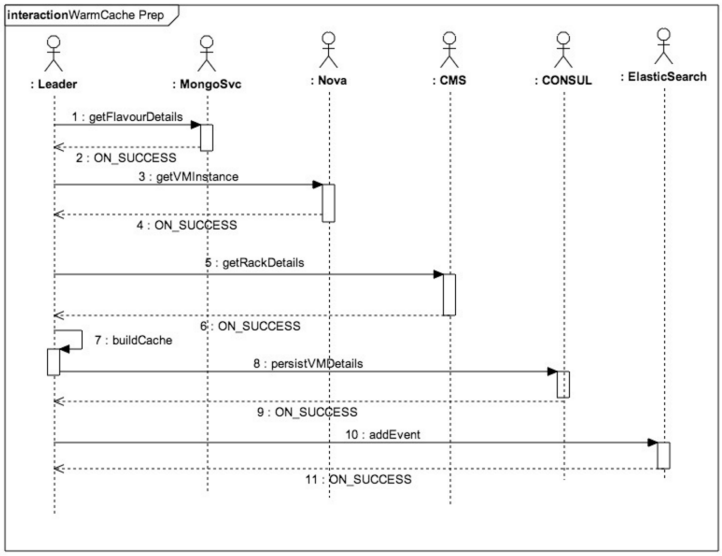

Fig. 11. System Sequence Diagram

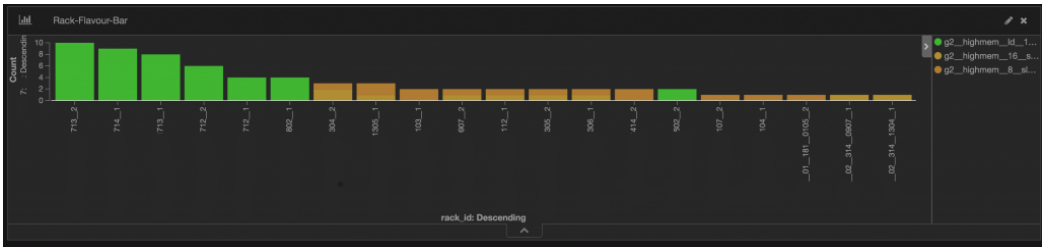

Fig. 12. VM Instances distribution

The corresponding metadata [20] information on Consul [2] for acquired VM instances is updated and removed from the active warm-cache list.

Apart from our ability to quickly flex up, another huge advantage of the warm-cache technique compared to conventional run-time VM creation methods is that before an Elasticsearch [1] cluster is provisioned, we know exactly if we have all the required non-error-prone VM nodes to satisfy to our capacity needs. There are many generic applications hosted [24] on a cloud [12] environment that require the ability to quickly flex up or to guarantee non-error-prone capacity for their application deployment needs.These distributed applications generate logs [8] on individual VMs or Nodes that needs to be collected and made available for application developers for debugging. They can take a cue from the warm-cache approach for solving similar problems.

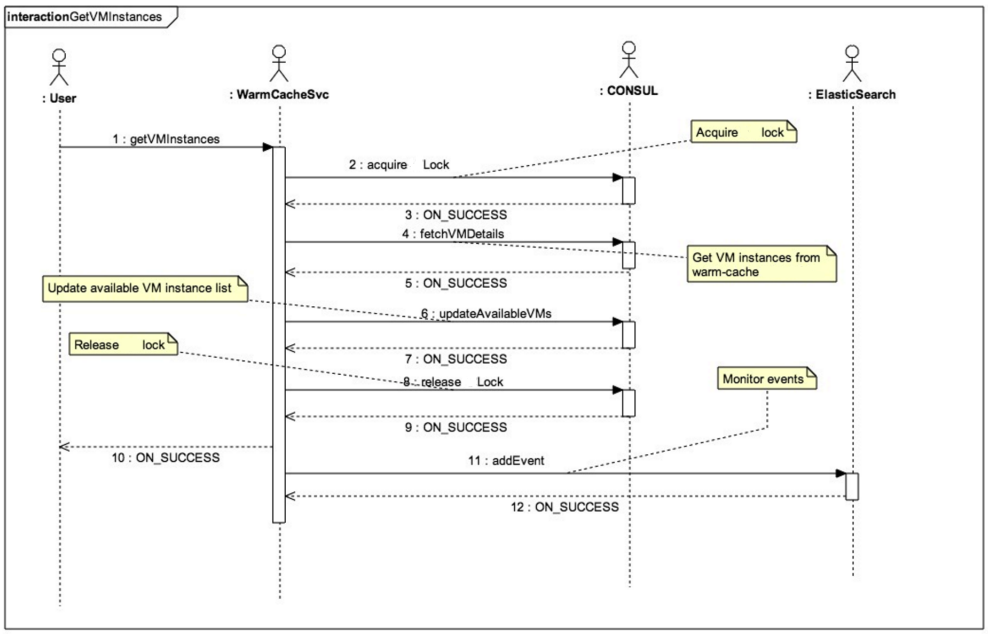

Fig. 13. Sequence Diagram of Acquiring a VM

5 Related Work

There are other published methods around instance caching. In order to speed up virtual server provisioning, there have been approaches to expedite the transfer of host specific metadata files using different transfer techniques. To reduce boot time, these approaches instantiate a VM, and store it in cache on standby mode. This saves time to create an instance from a template and boot VMs. Our paper focuses on providing infrastructure-aware custom caching mechanism with instance build rules for distributed applications, such as Elasticsearch. Also, our proposed method provides instance provisioning at faster pace. We have been able to get to cater to user initiated instance requests within few seconds.

6 Future Work

Currently our implementation outlines on a mechanism of creating cached instance on OpenStack [23] for a distributed application [14] like Elasticsearch [1]. We intend to extend this to other managed platforms such as Kubernetes [19] and Apache Mesos [26] . Platform will be made more generic to easily extend to support other applications on potentially different managed platform offerings. Currently, rules managed requires user prompt and we intend to automate instance creation rules based on historic usage.

7 Conclusion

Managed platforms such as Kubernetes [19], Mesos [26] etc, provide ability to spawn on demand instances. These instances would take up to few minutes, to be completely ready which might not be acceptable for critical hosted applications. Instance provisioning [25] mechanisms like the one outlined in the paper can alleviate such issues. Also a ready to use instance cache guarantees the availability of nodes in case of capacity flex up. There are additional infrastructure rules that are application specific such as being rack [11] aware. The outlined mechanism in the paper allows creating similar abstractions on top of managed cloud [12] offerings out there.

References

1. Elastic Search - https://www.elastic.co/
2. Consul - https://www.consul.io/
3. D. K. Vasthimal, S. Kumar and M. Somani, "Near Real-Time Tracking at Scale," 2017 IEEE 7th International Symposium on Cloud and Service Computing (SC2), Kanazawa, 2017, pp. 241-244.
4. Kibana - https://www.elastic.co/products/kibana
5. Nova - https://docs.openstack.org/nova/latest/
6. Gerald J. Popek and Robert P. Goldberg. 1974. Formal requirements for virtualizable third generation architectures. Commun. ACM 17, 7 (July 1974), 412-421.
7. J. E. Smith and Ravi Nair, "The architecture of virtual machines," in Computer, vol. 38, no. 5, pp. 32-38, May 2005.
8. D. K. V, R. R. Shah and A. Philip, "Centralized log management for pepper," 2011 IEEE Third International Conference on Cloud Computing Technology and Science, Athens, 2011, pp. 1-3.
9. Kristina Chodorow, (2010) MongoDB: The Definitive Guide. "O'Reilly Media, Inc."https://www.overleaf.com/project/5cdb3db3fc43da79efecd23d
10. Y. Breitbart, D. Georgakopoulos, M. Rusinkiewicz and A. Silberschatz, "On rigorous transaction scheduling," in IEEE Transactions on Software Engineering, vol. 17, no. 9, pp. 954-960, Sept. 1991.
11. The Computer Rack section of The University of Iowa's DEC PDP-8, documents a relay rack made in 1965; Nov. 2011.
12. B. Rochwerger et al., "The Reservoir model and architecture for open federated cloud computing," in IBM Journal of Research and Development, vol. 53, no. 4, pp. 4:1-4:11, July 2009.
13. Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. 2010. ZooKeeper: wait-free coordination for internet-scale systems. In Proceedings of the 2010 USENIX conference on USENIX annual technical conference (USENIXATC'10). USENIX Association, Berkeley, CA, USA,
14. Arora, Sanjeev; Barak, Boaz (2009), Computational Complexity A Modern Approach, Cambridge, ISBN 978-0-521-42426-4
15. D. Vasthimal, "Robust and Resilient Migration of Data Processing Systems to Public Hadoop Grid," 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), Zurich, 2018, pp. 21-23.
16. "Web Services Architecture". World Wide Web Consortium. 11 February 2004. 3.1.3 Relationship to the World Wide Web and REST Architectures.

17. D. K. Vasthimal, P. K. Srirama and A. K. Akkinapalli, "Scalable Data Reporting Platform for A/B Tests," 2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS), Washington, DC, USA, 2019, pp. 230-238.
18. N. Gold, A. Mohan, C. Knight and M. Munro, "Understanding service-oriented software," in IEEE Software, vol. 21, no. 2, pp. 71-77, March-April 2004.
19. Kelsey Hightower, Brendan Burns, Joe Beda, (2017) Kubernetes: Up and Running: Dive Into the Future of Infrastructure. "O'Reilly Media, Inc."
20. Hui Han, C. L. Giles, E. Manavoglu, Hongyuan Zha, Zhenyue Zhang and E. A. Fox, "Automatic document metadata extraction using support vector machines," 2003 Joint Conference on Digital Libraries, 2003. Proceedings., Houston, TX, USA, 2003, pp. 37-48.
21. Wolfe A, "Software-Based Cache Partitioning for Real-time Applications", Third International Workshop on Responsive Computer Systems, 1993 Sep, OCLC: 39246136
22. Xiaoshan Li, Zhiming Liu and H. Jifeng, "A formal semantics of UML sequence diagram," 2004 Australian Software Engineering Conference. Proceedings., Melbourne, Victoria, Australia, 2004, pp. 168-177.
23. Omar SEFRAOUI, Mohammed AISSAOUI, Mohsine ELEULDJ, "OpenStack: Toward an Open-Source Solution for Cloud Computing" 2012 International Journal of Computer Applications (0975 - 8887), Volume 55 - No. 03, October 2012.
24. S. Kumar and D. K. Vasthimal, "Raw Cardinality Information Discovery for Big Datasets," 2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS), Washington, DC, USA, 2019, pp. 200-205.
25. B. Guenter, N. Jain and C. Williams, "Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning," 2011 Proceedings IEEE INFOCOM, Shanghai, 2011, pp. 1332-1340.
26. Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: a platform for fine-grained resource sharing in the data center. In Proceedings of the 8th USENIX conference on Networked systems design and implementation (NSDI'11). USENIX Association, Berkeley, CA, USA, 295-308.

Authors

Sudeep Kumar has deep expertise in building scalable and resilient platforms capable of handling petabytes of unstructured data every day. His industry experience spans across different domains like E-commerce, Embedded systems and Telecom. Specialized technical skills include solving Big data problems, Building platforms and frameworks, Client Side programming and scalable Server side backends.

Deepak Kumar Vasthimal received Bachelors in Computer Science and Engineering from Vijayanagar Engineering College, Bellary India and Masters of Science in Software Systems from BITS Pilani, India. He is currently Senior MTS, Software Engineer at eBay Inc. San Jose, California USA. He has filed several patents, research publications in the fields of recommendation algorithms, adaptive data platforms, search algorithms, digital advertising platforms, temporal social networks, graphical user interface, payments, marketplaces, wearable hardware and system infrastructure. His current research and engineering focus is building scalable, multi-tenant, distributed machine learning platform at eBay.

Musen Wen is an applied researcher in Search Science and Technology at eBay Inc. His research research includes statistical machine learning, deep learning, reinforcement learning and large-scale practical problems, for example, e-Commerce search, recommendation, and online advertising. Musen holds a PhD in statistical machine learning from the University of California.