

Tech Report - Feasible Solution Design for Vertico

Abstract—In Volunteer Computing (VC), Volunteers donate their idle computers to many scientific projects like Folding@home for understanding protein folding and how it relates to disease. VC applies to the particular set of problems. Typically VC projects work by breaking the problems into chunks called work units, which are sent to computers around the world to be analyzed.

We are aiming to extend the idea of VC to different kinds of applications which live in the cloud, so that we minimize the operating cost. VC for Cloud is challenging problem till today. Addressing those issues and providing possible solution design is our goal. The end product will have many benefits, for example, an organization can minimize the Power bills and donate monetary credits to Volunteers. It is like a win-win situation for volunteers and organization. So we introduce VERTICO, stands for Volunteer Computing Research Cloud Framework. In this paper, we are giving possible solution design for VERTICO and its evaluations.

I. PROBLEM STATEMENT

Cloud computing works by applying many technologies but limited to data centers. Volunteers are ready to donate their Computer Machines and giving good results. Now, we have to figure out how to run Cloud on Volunteer Machines. Bringing cloud computing to volunteer machine requires to work towards to modern User Experience, Network and security. Adding security and network components to existing cloud technology could easily bring cloud computing into volunteer machine. So in this paper we are taking that first step to in cover obstacles with a proof of concept.

II. BACKGROUND INFORMATION

BOINC[3] is running VC for many Scientific Projects over ten years. On Jun 6, 2017, NSF funds new model for BOINC for three years, Advanced Computing Center to develop a new framework for BOINC-based volunteer computing, in which volunteers sign up for science goals rather than projects. Studying BOINC will be useful in refining our designs. Also, the outcome of this paper might help in refining BOINC new model.

III. RELATED WORK

Amazon or Google or other cloud platforms using mainly the Virtual Machines (VM). They manage all its machines by a scheduler. Many other critical components make the whole Cloud. The core component is the VM. In that there are two types of Hypervisors - the first one runs on the bare metal machine, the second one runs on the installed operating system. Both types used in Cloud. OpenStack, Dockers and open source Virtual machines technologies such as qemu, VirtualBox, etc. are widely successful in creating a cloud

on data centers. There are many question which are still not addressed efficiently, for example - How to expose volunteer machine network to the world safely? and How secure we can achieve by bringing cloud into volunteer machine? How easy to hack Virtual Machine?

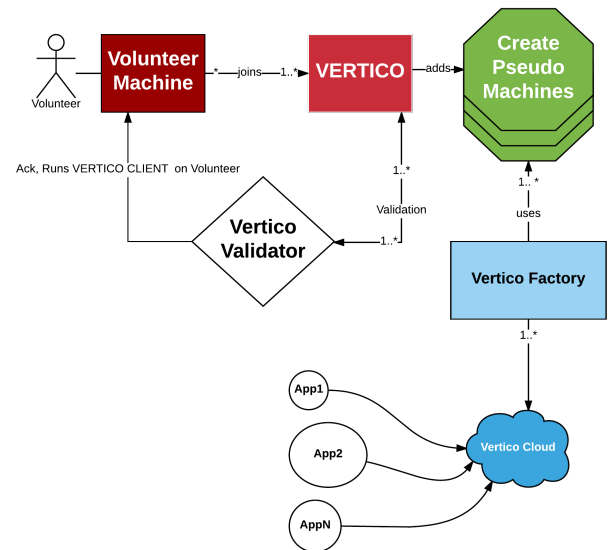
IV. FEASIBLE SOLUTION DESIGN

In this section, we are going to explain each modules in detail and various design strategies.

A. Initial Design goals:

Initially we are trying to mimic Boinc in Vertico, because its very much important to have task and work based architecture. So we tried to achieve the Boinc capability in Volunteer computing. Later, we extended the framework on running virtual machine and containers inside volunteer machine. So in conclusion, the end-product will have all features of cloud computing. Here, volunteer registers to Vertico WebApp, then Vertico factory will create/update the volunteer machine capability. We called it as pseudo virtual machine. Any cloud application will have an interface to Vertico factory, so that they can deploy the cloud computing application into volunteer machine. Fig 1 gives a quick overview.

Fig. 1. Initial High level overview Diagram



B. Keep it simple:


We are currently allowing only 5 actions(refer fig 2) Volunteer can perform. In Boinc, Volunteer can choose

- 1) User Experience in terms of easy to use.
- 2) Prone to attack is less, since volunteer doesn't know what project running makes difficult to hack. Say for example Volunteer is running MapReduce Job for Facebook, then volunteer can try to guess what's going with computer, thus its vulnerable. However it doesn't applies for Boinc Projects.

```
graph LR; KILLED((KILLED)) -- "Volunteer :: register" --> ACTIVE[ACTIVE]; ACTIVE -- "Volunteer :: start" --> SERVING[SERVING]; SERVING -- "Volunteer :: stop" --> PAUSE[PAUSE]; PAUSE -- "Volunteer :: start" --> SERVING; SERVING -- "Volunteer :: unregister" --> INACTIVE[INACTIVE]; INACTIVE -- "Volunteer :: cleanup" --> KILLED;
```

We used Auth0 for identity management using Various Trusted agents(like Google, Facebook, or Github etc.) The main motivation behind this present in this paper on SCC[8], we can have initial confidence level of running a cloud App on particular Instance. Say I am Software Engineer - I will understand Vertico Better than a User who is Doctor. So confidence level would be high for my system. We are always thinking the ways how we can integrate SCC with Vertico. So we kept the WebApp as separate module using Auth0(at present only email/google/facebook registration allowed) platform for Identity Management.


- 1) OpenStack- We can't use directly OpenStack, instead we need to modify the openstack. We choose not to go ahead with OpenStack because of complexity in changing existing code in very short period of time(2 months). *Open stack possibility is quite complex to explore, but in future we can use OpenStack Codebase as a reference to final product. So do not add any functionality into openstack, instead extend as a new project in open stack. For example, Open stack Glance project provides image service - we can use that in Vertico project, but extend it as new project, also rewrite the entire code to adapt Vertico Goals*
- 2) Docker - If docker doesn't exist then Vertico would be inventing it. We rely mainly on dockers to make this possible.




Vertico Cloud

Log In

Sign Up





Sign Up with Facebook



Sign Up with Google

or





Sign Up >

The diagram illustrates the architecture of the **vertigo-webapp**. It features a central **factory** component (yellow shield) that interacts with several other parts:

- webapp** (white box) containing **Angular** (yellow shield) is connected to the **factory**.
- cli** (white box) containing **aug17.NodeJS** and **future: GO** (yellow shield) is also connected to the **factory**.
- The **factory** is connected to **rdbs** (white cylinder) and **postgres** (white box) containing **postgres** (yellow shield).
- rdbs** is connected to **rdbs-bkup** (white box) via a dashed line.
- The **factory** is connected to **dev-cli** (white box) containing **future: GO** (yellow shield).

Legend:

- Future Development**: Represented by a dashed box.
- Scale up based on load**: Represented by a white box.

Additional text at the bottom:

- cli** - vertico command line interface to use volunteer system
- dev-cli** - vertico command line interface to deploy cloud apps/instances
- rdbs** - Tested with Postgres
- vertico-webapp** - For registration, unregistration and redeem credits etc.

- 3) VirtualBox - Oracle Virtualbox very mature software in terms of compatibility with Windows, Darwin, and Linux OS. So we choose Virtualbox and its programming interfaces.
- 4) Angular 4.0 and above - The latest version quite fast and highly productive. Also building and deploying is very easy and small in size. The entire webapp is just 300kb in compressed mode. *In future, the end-product will have many dynamic features, so we choose Angular thinking in future we will have high dynamic features like Amazon Shopping site*
- 5) NodeJS and NPM - We choose this just because highly productive and fast language compared to Python.
- 6) Golang - In future we are choosing this because its quite fast compared to Java, and also building go app is quite easy.
- 7) Cuba-platform - We choose a platform which is highly productive and enterprise level of software. It's quite easy to extend the cuba-platform using Java.
- 8) RDBMS - Without doubt, we choose relational

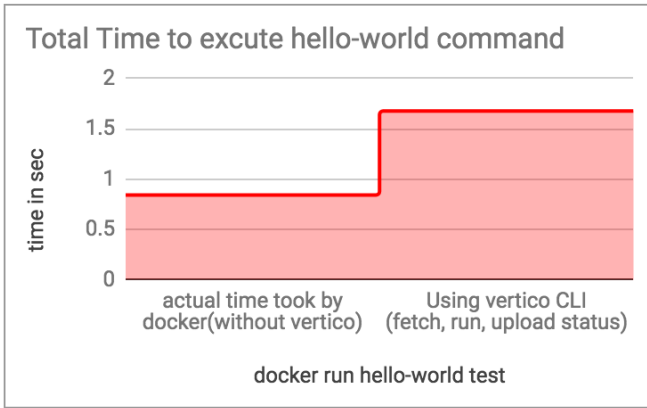
database, because its still top in performance. We recommended by Cuba-platform developers to go ahead with Postgres instead mysql. We can switch to any RDBMS.

E. Evaluation

Strategy: We simulated the environment for 1 million users, and 0.5 million instances. Every below experiment is running on top of it.

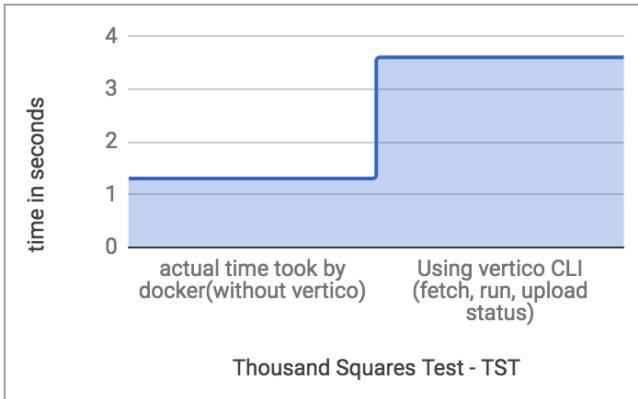
- 1) Simple hello-world test - in which we used docker simple hello-world image to execute this test. The fig3, shows its analysis. We see we have some network overhead, other than that it looks pretty decent in performance. However, for production we can gain much more performance using *golang* for the command line wrapper for docker apis.

Fig. 5. Docker Hello world latency



- 2) TST - Thousand Square Test, in this we are computing the 1000 squares in *Python - Julia docker container*.

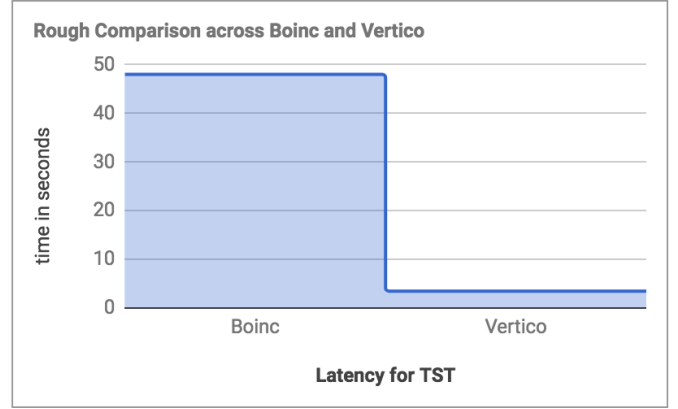
Fig. 6. TST - Thousand Square Test latency



- 3) Rough Comparison with Boinc: We are using same TST, here we compared it with Boinc, the performance is below. **Here, we have not developed all the features of how Boinc Works, for example we are not calculating the CPU credits during in**

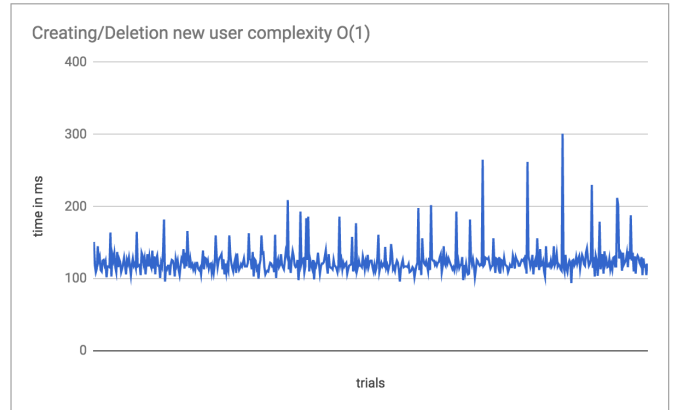
this evaluation. For this evaluation, we are running docker version of Boinc server, and single volunteer is connected to it. The below figure shows a rough estimate on latency. We used Thousand Square Test in Boinc as well as Vertico. *Boinc creates two task for every single work. In Vertico, we have not taken that into consideration for now, but in future. Time took for single task in Boinc is approximately 50seconds. In Vertico its 3.5 seconds.*

Fig. 7. Boinc vs Vertico



- 4) Vertico Factory Performance: In this, we are using RDBMS(postgres). We simulated an environment for 1 millions users, and 0.5 million instances. The database size is around 0.5 GB. **Time complexity for Create, Read, Update, Delete is same as any SQL query time complexity.** Also, we can apply optimization, like indexing can be applied to improve the performance. Just to give an understanding, we took time to insert users in this simulated environment. We observer the insertion is $O(1)$. The, simulate dataset is available here.[1]

Fig. 8. Vertico Factory Performance



V. FUTURE WORK

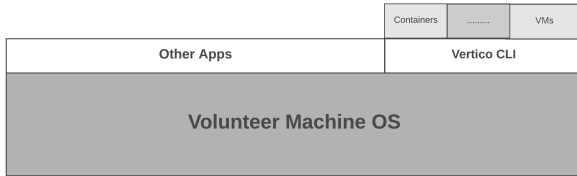
User Experience, Network and Security are the most important pillars in Vertico.

A. User Experience

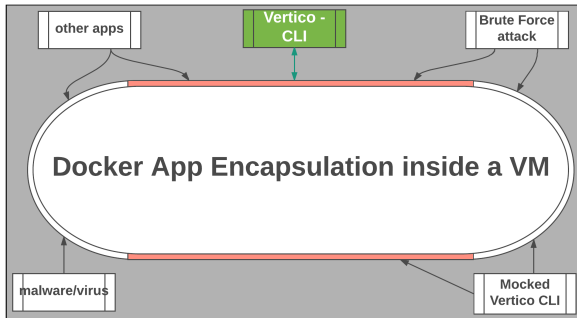
User/Volunteer must have high level of Trust Factor to Vertico-CLI, because we are asking for super privileges. Once Uncle Ben said, "Great power comes with great responsibility", so we must adapt to that concept while building Vertico-CLI. One of the ways we can make this possible is using Virtual Machine. At present we are not following encapsulation, but in future we must encapsulate all the cloud app running on Volunteer Machine.

Fig. 9. App Encapsulation

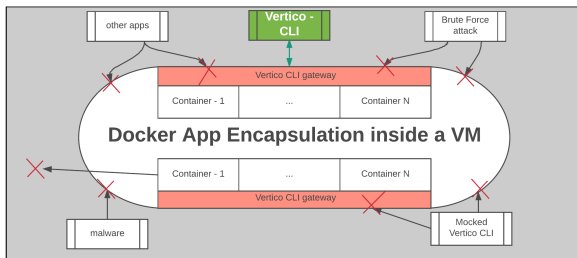
Current App Encapsulation



Proposed App Encapsulation using VM



Proposed App Encapsulation using VM in detail



Volunteer OS(darwin, linux, or Windows)

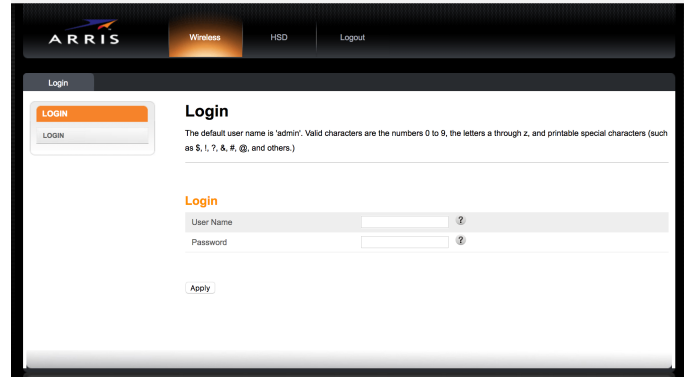
B. Security

- We must secure the virtual machine. Encrypting the virtual machine would be safer, also in future we should provide security scale. Security scale indicate how secure it is compared to other cloud computing platforms - **this is the main reason we are not even roughly comparing the current development of Vertico with AWS, Google Cloud.**

C. Network

- Almost every volunteer will be behind NAT, there are many ways we can circumvent legally using TCP/UDP punch hole technique. But, we need to explore on "Is it worth to do so?" - simple answer is NO[12]" In my point of view, this is a business problem, many Internet vendors are exposing web-application as gateway to modify settings such as port-forwarding/password etc. For example, I am using RCN Internet, I use RCN-Admin-setting(fig 10) exposed by RCN to change settings. **My Only question, if they can expose web application as Admin controller, then why they are not exposing APIs as gateway?** If they do then we can achieve Vertico very easily and Securely. So that, vertico cli will use port-forwarding to solve the volunteers who are behind the NAT.

Fig. 10. RCN modem setting page exposed to Volunteer



VI. CONCLUSION

Application architecture are evolving, now designing a new app goes like AI first, Cloud first, Mobile first precedence. I think few years down the line Solution Architects might think the Vertico way also. So I believe today no one curious about Vertico as much as we do because Cloud is way affordable, what if that is not the case in near future - because we are in the big-data era. We just need to connect some more dots, like UX, Security and Network with Vertico to make more reasonable choice for cloud computing.

ACKNOWLEDGMENT

My sincere thanks to Project Mentors(Prof. Ioan Raicu and Prof Kyle Chard) , and Alex(helped me greatly in getting up to the speed with many advanced tech concepts), for reviewing our design and supporting us throughout the duration of this project. We also would like to thank our department, chameleon cloud and the university for providing the infrastructure for the study of this project. Gratitude to all researches whose papers we have studied (complete list mentioned in the references section) during the course of this project.

REFERENCES

- [1] Source code - <https://github.com/iitc>
- [2] BOINC, <https://boinc.berkeley.edu/>

- [3] Open-stack, <https://www.openstack.org/assets/pdf-downloads/virtualization-Integration-whitepaper-2015.pdf>
- [4] Bayanihan: building and studying web-based volunteer computing systems using Java <https://pdfs.semanticscholar.org/38e9/cb4dc3692d54fa0f1ef834aabbad136a>
- [5] High-performance task distribution for volunteer computing — <http://ieeexplore.ieee.org/abstract/document/1572226/>
- [6] Virtualization Essentials book by Matthew Portnoy
- [7] White paper on modern app architecture by Docker, <https://goto.docker.com/modern-app-architecture.html>
- [8] A Social Compute Cloud: Allocating and Sharing Infrastructure Resources via Social Networks <http://ieeexplore.ieee.org/document/6727497/>
- [9] Auth0 - <https://auth0.com/>
- [10] part-1 - <https://youtu.be/RznhlignV9k>
- [11] other videos - <https://iitc.github.io>
- [12] <https://stackoverflow.com/questions/45093203/udp-punch-hole-for-a-web-server>

VII. VERTICO - VERSION V.1 SCREEN SHOTS

The easy way to validate a framework is by giving the examples, so the proof of concept for the final production app for vertico is in github[1]. All application are dockerized and source code is available at [1]. Demo vidoes are available at [10][11].

A. Vertico CLI

Fig. 11. Vertico Command line app running on Linux

```
> vertico --help
Vertico
Beta-0.0.1

Usage: vertico [options]

Options:
  -v, --version            output the version number
  -r, --register <args>   Register this machine to my account using email and secret.
                          Examples: "vertico <username> <secretkey>"
  -s, --start              start the vertico Service
  -f, --fetch              fetch the vertico Service (available only for developers)
  -p, --pause              stop the vertico Service
  -u, --unregister <args> Unregister this machine to my account using email and secret.
                          Examples: "vertico <username> <secretkey>"
  -h, --help              output usage information
```

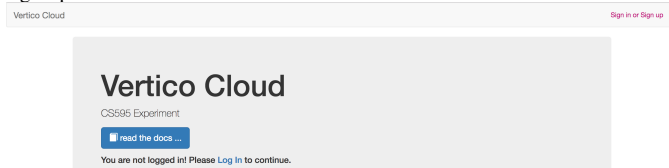
Fig. 12. Vertico Command line - welcome screen

```
Last login: Sat Aug 12 22:15:51 on tty001
~$ vertico
Vertico
Beta-0.0.1

[2017-08-13 23:17:46.669] [INFO] VirtualBox - Listing VMs
[2017-08-13 23:17:46.690] [INFO] VirtualBox - VirtualBox version detected as 5
Run Aug 13 2017 23:17:46 GMT-0500 (CDT) ~ info
COMMAND ~ info
ID ~ info
NAME ~ info
STATUS ~ info
HOSTS ~ info
PASSWORD ~ info
EXAMPLE ~ info
aajay14@hello-world ~ info
Run Aug 13 2017 23:17:46 GMT-0500 (CDT) ~ info {}
Run Aug 13 2017 23:17:46 GMT-0500 (CDT) ~ info Name ~ info
Run Aug 13 2017 23:17:46 GMT-0500 (CDT) ~ info Unique Machine ID ~ info
Run Aug 13 2017 23:17:46 GMT-0500 (CDT) ~ info OS ~ info
Run Aug 13 2017 23:17:46 GMT-0500 (CDT) ~ info CPU ~ info
Run Aug 13 2017 23:17:46 GMT-0500 (CDT) ~ info Memory ~ info
Run Aug 13 2017 23:17:46 GMT-0500 (CDT) ~ info DiskSpace Available ~ info
Run Aug 13 2017 23:17:46 GMT-0500 (CDT) ~ info PublicIP ~ info
Run Aug 13 2017 23:17:46 GMT-0500 (CDT) ~ info PrivateIP ~ info
Run Aug 13 2017 23:17:46 GMT-0500 (CDT) ~ info Registered ~ info
Run Aug 13 2017 23:17:46 GMT-0500 (CDT) ~ info For more info, try ~ info
Run Aug 13 2017 23:17:46 GMT-0500 (CDT) ~ info "vertico --help"
```

B. Vertico Webapp

Fig. 13. Vertico Web UI integrated with Auth0[9] for google/facebook sign up.



C. Vertico Factory

Fig. 14. Volunteer profile details

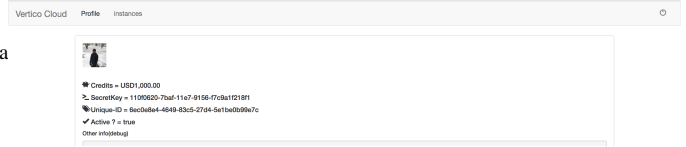


Fig. 15. Volunteer donated details

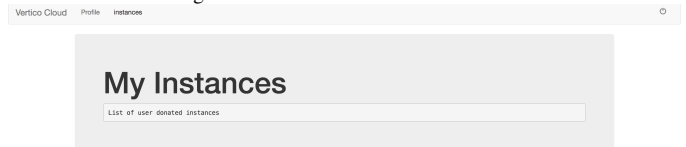


Fig. 16. List of donors

Name	Donors email	Total Credits	Active	Secret key
Ajay Ramesh	ajay14@gmail.com	1,000	<input checked="" type="checkbox"/>	1e28c5d0-7b78-11e7-beef-99415c448bd6
Ajay Ramesh	ajayamesh@gmail.com	1,000	<input checked="" type="checkbox"/>	41db9110-7b78-11e7-a2b5-3b75aba9b728
Ajay Ramesh	aramesh6@hawk.it.edu	1,000	<input checked="" type="checkbox"/>	2c955d0-7b78-11e7-a7e6-0ddfd40ec0c8
Admin User	user.instance@gmail.com	1,000	<input checked="" type="checkbox"/>	6c8e1240-7b76-11e7-87ad-f3411f05577


Fig. 17. Donor view in detail

Name	Operating System	Public IP	Private IP	Ram	Disk space	Cpu	Status	Credits	Info
Banyan-Tree-403-Asia-TE455T	Mac OS X	104.194.105.32	104.194.105.32	8 GB	26.03835678100586 GB	Intel® Core™ i5-5257U	Serving	100	("system", ("
Baobab-Tree-499-Asia-TE461ST	Mac OS X	208.59.156.176	192.168.0.6	8 GB	14.17108154296875 GB	Intel® Core™ i5-5257U	Active	100	
Baobab-Tree-499-Asia-TE461ST	Mac OS X	208.59.156.176	192.168.0.6	8 GB	14.17108154296875 GB	Intel® Core™ i5-5257U	Active	100	
Baobab-Tree-937-Australia-TE893ST	Mac OS X	208.59.156.176	192.168.0.6	8 GB	5.0174560546875 GB	Intel® Core™ i5-5257U	Active	100	("system", ("
Coconut-Tree-937-Australia-TE893ST	Mac OS X	208.59.156.176	192.168.0.6	8 GB	5.0174560546875 GB	Intel® Core™ i5-5257U	Serving	100	("system", ("
Linden-Tree-547-NorthAmerica-TE378ST	Mac OS X	208.59.156.176	192.168.0.6	8 GB	19.019317628953125 GB	Intel® Core™ i5-5257U	Active	100	("system", ("
Linden-Tree-618-Australia-TE193ST	Ubuntu	129.114.108.1	10.40.0.138	125.68359375 GB	212.0528564453125 GB	Intel® Xeon® E5-2670 v3	Serving	100	("system", ("
Sequoia-Tree-849-SouthAmerica-TE138ST	Ubuntu	129.114.108.103	10.40.0.2	125.68352508544922 GB	223.86115264892578 GB	Intel® Xeon® E5-2670 v3	Serving	100	("system", ("

Fig. 18. List of Apps and its meta-data

Name	Operating System	Public IP	Private IP	Ram	Disk space	Cpu	Status
Linden-Tree-794-Antarctica-TE944ST	Ubuntu	129.114.108.103	10.40.0.2	125.68352508544922 GB	223.86087036132812 GB	Intel® Xeon® E5-2670 v3	Serving
Baobab-Tree-499-Asia-TE461ST	Mac OS X	208.59.156.176	192.168.0.6	8 GB	14.17108154296875 GB	Intel® Core™ i5-5257U	Active
Baobab-Tree-499-Asia-TE461ST	Mac OS X	208.59.156.176	192.168.0.6	8 GB	14.17108154296875 GB	Intel® Core™ i5-5257U	Active
Baobab-Tree-499-Asia-TE461ST	Mac OS X	208.59.156.176	192.168.0.6	8 GB	19.019317628953125 GB	Intel® Core™ i5-5257U	Serving
Coconut-Tree-937-Australia-TE893ST	Mac OS X	208.59.156.176	192.168.0.6	8 GB	5.0174560546875 GB	Intel® Core™ i5-5257U	Active
Baobab-Tree-499-Asia-TE461ST	Mac OS X	208.59.156.176	192.168.0.6	8 GB	14.17108154296875 GB	Intel® Core™ i5-5257U	Active
Sequoia-Tree-849-SouthAmerica-TE138ST	Ubuntu	129.114.108.103	10.40.0.2	125.68352508544922 GB	223.86115264892578 GB	Intel® Xeon® E5-2670 v3	Serving
Linden-Tree-618-Australia-TE193ST	Ubuntu	129.114.108.1	10.40.0.138	125.68359375 GB	212.0528564453125 GB	Intel® Xeon® E5-2670 v3	Serving
Banyan-Tree-403-Asia-TE455T	Mac OS X	104.194.105.32	104.194.105.32	8 GB	26.03835678100586 GB	Intel® Core™ i5-5257U	Serving

Fig. 19. Creation of Docker Cloud Application


Vertico Cloud Services

Donors editor ×
Instance browser ×
Applications editor ×

[Applications browser](#) > Applications editor

App name

Deploy instance
▼
...
Q

Image location

Status
▼

App type
▼

Init script


docker run hello-world

Stop script

Applog

Hello from Docker!
This message shows that your
installation appears to be working
correctly.

Fig. 20. Creation of Virtual Machine Cloud Application


Vertico Cloud Services

Donors editor ×
Instance browser ×
Applications editor ×

[Applications browser](#) > Applications editor

App name

Deploy instance
▼
...
Q

Image location

Status
▼

App type
▼

Init script

docker run hello-world

Stop script

Applog

Hello from Docker!
This message shows that your
installation appears to be working
correctly.