My Private Cloud

How to build a low-cost private cloud platform

Mark K. Reha

Abstract—Can you build an expandable fully functioning private cloud platform with Raspberry Pi's that is powered by Platform as a Service cloud computing services? Can you build a private cloud for about $500 in an afternoon that would easily fit on a shelf or in your closet? This paper outlines an approach that includes a list of parts and prescriptive guidance for how to build your own fully functioning private cloud platform with Raspberry Pi's for about $500 and in just a few hours.

Index Terms— Cloud Computing, Computer Uses in Education, Distributed Architectures, Information Systems Education, Information Technology and Systems Applications, Software Engineering for Internet Projects, Solution Reference Architectures, Special-Purpose and Application-Based Systems

1 Introduction

The author is a full-time professor at a major university and primarily teaches software design, software development, and software engineering courses. These courses teach students programming languages, web development frameworks, databases, cloud computing, and embedded systems development. The author also worked in the industry for 35 years and held many roles as a software architect and in various leadership positions.[3] One of the many amazing changes the author has seen in technology over the recent years is the proliferation of the use of public cloud platforms and the use of the Raspberry Pi in embedded systems. The author has deployed several applications to all the major public cloud platforms, which even for simple applications can be quite costly per month. The author has also developed several applications that run on the Raspberry Pi. The idea for this project came from the previously mentioned technologies. Could someone build a fully functioning cloud platform to host applications and databases for their own personal use, used in academia, or even for use in industrial applications? Could someone build a fully functioning cloud platform that was affordable, easy to use, and was easy to manage? This paper will not only answer these questions but also provide the reader with all the build instructions and application code via the implementation of *"My Private Cloud"* reference design [1], which anybody can use to build their very own low-cost private cloud platform.

How does this paper differ from other articles and resources that can be found on the Internet when it comes to using Raspberry Pi's to create a cloud platform? The author really could not find any resources that provided a complete solution for building a cloud platform beyond installing Docker and Kubernetes on a cluster of Raspberry Pi's. Most resources simply outlined the steps to install Kubernetes along with Docker and then join Kubernetes worker nodes to the Kubernetes master node. Demonstrations in these articles were not any more complex than how to manually run a few simple Docker images on the cluster. The reference design outlined in this paper is a complete and fully functioning cloud platform allowing a developer to provision a Platform as a Service (PaaS) container, deploy their application code to the container, and allow end users to run actual web and database driven applications. This cloud platform reference design also includes a strategy that could be used to manage and monitor the cloud platform.

This paper is divided into several sections each focusing on various aspects of the private cloud platform reference design. This includes the following focus areas:

- **Information Technology**: This section is focused on how to build an affordable cloud platform from a hardware, networking, storage, enclosure, and power perspective. These will essentially provide all the typical Infrastructure as a Service (IaaS) cloud computing services.
- **Cloud Engineering**: This section is focused on all the software and configuration that needs to be completed on the hardware to make a scalable, extendable, and affordable cloud platform.
- **Platform as a Service**: This section is focused on how the cloud platform supports Platform as a Service (PaaS) cloud computing services and how runtimes can be easily added to the cloud platform to extend its functionality.
- **Portal Application:** This section is focused on how various runtimes can be provisioned by a software developer with the ability to deploy application code and run web-based applications that could be backed by a database. The Portal application provides developers with access to fully functioning PaaS cloud computing services and allows end users to run the deployed applications.

• *The author Mark K. Reha is an Assistant Professor and Program Chair at Grand Canyon University, Phoenix, AZ 85017 USA (e-mail: mark.reha@gcu.edu).*

- **Cloud Monitoring and Administration:** This section is focused on how the cloud platform can be monitored and managed by cloud administrators.

To close out this introduction and before the details of how to build the cloud platform are presented, the overall physical and logical architecture of the cloud platform will be discussed.

The physical architecture, as shown in the figure below, consists of a 4-node cluster of servers that support most of the hardware that make up the cloud platform. There is a node for the Kubernetes master node and three nodes for the Kubernetes worker nodes. If desired, the architecture could be easily extended to support a redundant master node and additional worker nodes. The master node is the single access entry point for the cloud platform and runs the Kubernetes control plane. The worker nodes provide compute resources that provide the CPU, memory, and storage necessary to run applications on the cloud platform. The master node and all worker nodes are implemented using a Raspberry Pi 4 Model B with 4GB of memory. The master node and worker nodes are networked together to form a private cloud computing network using a low-cost ethernet switch. The master node and all worker nodes are powered with an AC adapter that is approved for the Raspberry Pi 4, powered thru a single 15A power strip, and cooled with heat sinks and fans that are approved for the Raspberry Pi 4. The physical architecture supports all the typical Infrastructure as a Service (IaaS) cloud computing services.

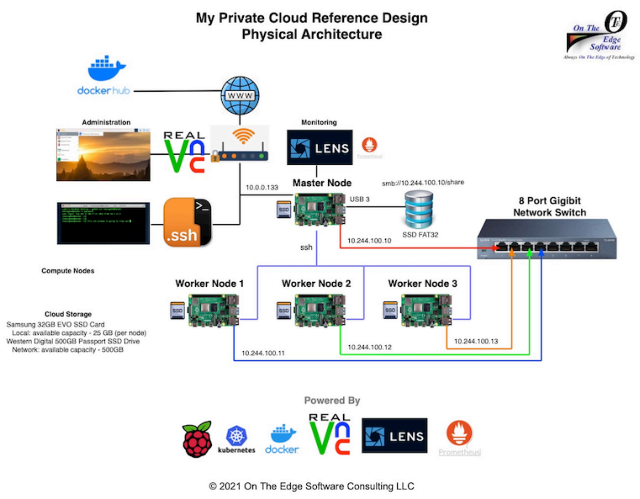

Fig. 1. Physical Cloud Architecture

The following table summarizes the total compute resources provided by the 3-node worker node cluster. It should be noted that additional worker nodes can be easily added to the cloud platform to increase the total available compute resources.

TABLE 1
Cloud Platform Compute Resources

CPU	RAM	Local Storage
3x3 for 9 Cores [1]	3x3 for 9 GB [2]	3x25 for 75GB [3]

The logical architecture, as shown in the figure below, includes applications not only for the end users, developers, but also for the administrators of the cloud platform. The application used by the end user and developer, referred to as the Portal application and backed by a relational database, was implemented as a web-based application that included in its design an open-source Java library that was used to communicate to the Kubernetes control plane. For administration and monitoring of the cloud platform a desktop application was incorporated into the design along with the ability to also connect to the Kubernetes master node directly via VNC and SSH. The logical architecture supports all the typical Platform as a Service (PaaS) cloud computing services.

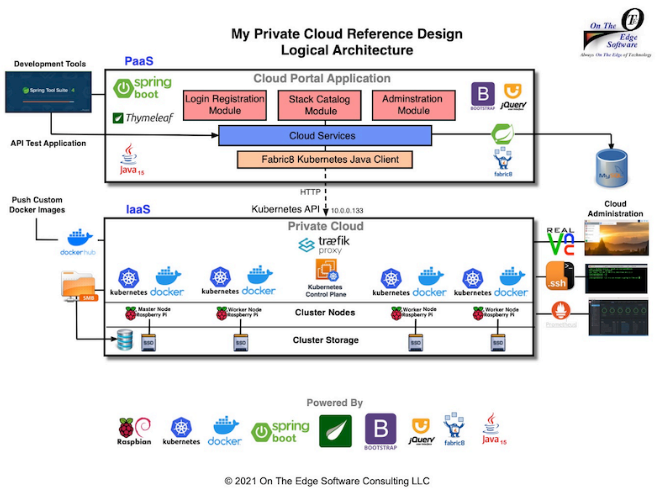

Fig. 2. Logical Cloud Architecture

2 Information Technology

2.1 Introduction

This section outlines the Information Technology related tasks required to build the physical aspects of the cloud platform. These tasks include building the actual cluster of Raspberry Pi's, mounting the Raspberry Pi's in an enclosure, networking the Raspberry Pi's together, setting up the cloud storage that will be available, and installing the operating system boot image on each of the Raspberry Pi's. Detailed step-by-step instructions are not provided in this paper but are available as noted in the references [1] in this paper. It will take about 1 to 2 hours to complete the building of the cluster. The total cost of the cluster will be about $500 and consist of the following parts that can be purchased from many online retailers, including Amazon:[1]

[1] 1Core on each Pi allocated to operating system, Kubernetes, and Docker
[2] 1GB on each Pi allocated to operating system, Kubernetes, and Docker
[3] 7GB on each Pi allocated to base install and stored Docker images

TABLE 2
Cloud Platform Hardware

Part	Vendor	Quantity	Cost
Raspberry Pi 4	Raspberry Pi 4 Model B with 4GB	4	$260
32GB SSD Card	SAMSUNG 32GB 95MB/s microSDHC EVO Select Memory Card	4	$30
Heat Sinks & Fans	Enokay Raspberry Pi 4 Model B Cooling Fan RAM Heatsink Set	4	$40
Power Adapters	CanaKit 3.5A Raspberry Pi 4 Power Supply	4	$40
Network Cables	Short Cat6 Ethernet Cables	1	$10
Ethernet Switch	D-Link Ethernet Switch 8 Port Gigabit (DGS-108)	1	$35
500GB USB-3 Drive (optional)	WD 500GB My Passport Go SSD USB-3 Drive	1	$65
Enclosure	GeauxRobot Raspberry Pi 4 7-Layer Dog Bone Stack Enclosure	1	$25
Power Strip	DERSECO 1875W Power Strip	1	$25

To expand the cloud platform additional worker nodes can be easily added to the cloud platform. The enclosure and power strip will accommodate 3 additional Raspberry Pi's. The cost of adding an additional Raspberry Pi 4 worker node will be less than $100 and will take about 1 hour to assemble and configure.

2.2 Raspberry Pi Preparation

Each Raspberry Pi will require a boot image to be copied to its SD Card. The boot image contains the actual operating system, which is referred to as the Raspbian OS. The standard Raspberry Pi Imager desktop application available from the Raspberry Pi web site is used to copy the boot image to the SD Card. The standard Raspbian OS will be used in the design. It should be noted that at the time of completing this cloud reference design that only a 32-bit version of the Raspbian OS was available. The following are the high-level tasks that need to be completed to create the boot image on the Raspberry Pi SD Card. Refer to the detailed step-by-step instructions available as noted in the references [1].

1. Download the Raspberry Pi Imager application.
2. Insert the SD card into an SD card adaptor and available SD card slot on a PC.
3. Run the Raspberry Pi Imager application and follow its onscreen instructions to copy the Raspbian OS to the SD card.
4. Configure the boot image to enable SSH.
5. Configure the boot image to enable VNC.
6. Configure the boot image to enable WiFi.
7. Configure the boot image to enable the Desktop.
8. Insert the SD card into the Raspberry Pi card slot.
9. Power up the Raspberry Pi and verify that it booted without any problems, and it is accessible via VNC and SSH over a WiFi network. For VNC access, use the VNC Viewer desktop application, which is available on the RealVNC home page.
10. Power off the Raspberry Pi.

2.3 Assembling the Cluster

The following are the high-level tasks that need to be completed to assemble the actual physical cluster. Refer to the detailed step-by-step instructions available as noted in the references [1].

1. Attach the Raspberry Pi heat sinks per the vendor instructions.
2. Attach the Raspberry Pi fan to the GPIO connector per the vendor instructions.
3. Label each of Raspberry Pi's: M1 for Master Node, N1 for Worker Node 1, N2 for Worker Node 2, and N3 for Worker Node 3.
4. Mount the Master Node to the 1st Layer of the Dog Bone Enclosure per the vendor instructions.
5. Mount the Worker Node 1 to the 2nd Layer of the Dog Bone enclosure per the vendor instructions. Mount the Raspberry Pi fan for the Master Node to the side of the enclosure.
6. Mount the Worker Node 2 to the 3rd Layer of the Dog Bone enclosure per the vendor instructions. Mount the Raspberry Pi fan for the Worker Node 1 to the side of the enclosure.
7. Mount the Worker Node 3 to the 4th Layer of the Dog Bone enclosure per the vendor instructions. Mount the Raspberry Pi fan for the Worker Node 2 and Worker Node 3 to the side of the enclosure.
8. Plug the power strip into the wall and make sure the power is turned off.
9. Plug each of the Raspberry Pi's power adapter into the power strip and then plug in each connector of the power adaptor into the Raspberry Pi.
10. Plug an Ethernet Cable into each of the Raspberry Pi's ethernet port and then to an open port on the Ethernet Network Switch. Power on the Ethernet Network Switch.

You should now have all the hardware in place to continue configuring the cluster. The following picture illustrates what a completely assembled cluster should look like.

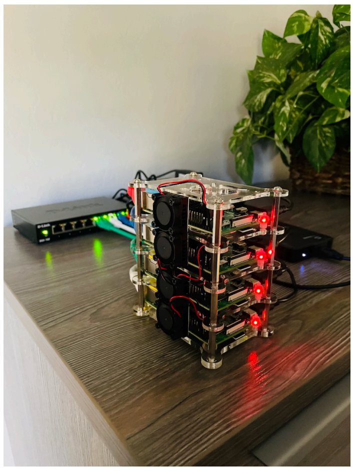
Fig. 3. Raspberry Pi Cluster

2.4 Networking

A private network will be configured to be used for communication internally between each of the Raspberry Pi's and by the Kubernetes control plane. The following are the high-level tasks that need to be completed to setup the private network. Refer to the detailed step-by-step instructions available as noted in the references [1].

1. Update the *hosts* file on each Raspberry Pi. A 10.244.100.0/24 network and a hostname of master-node, worker-node1, worker-node2, worker-node3 should be setup.
2. Update the *dhcpcd.conf* file on each Raspberry Pi to setup each Raspberry Pi's primary ethernet adapter with a static IP address of 10.244.100.10 for the master node and 10.244.100.11 thru 10.244.100.13 for each of the worker nodes.

2.5 Storage

The cloud platform will utilize available storage on each of the worker nodes SD card as well as a shared USB-3 SSD network drive. With a 32GB SD card each worker node will be able to leverage about 25GB of storage on the SD card for end user applications. The USB-3 SSD network drive should be plugged into the USB-3 port on the master node. This drive will be configured in the next section. It should be noted that, if desired, the storage on the reference design could be easily expanded by using a 64GB SD card on the Raspberry Pi. If should also be noted that if you want to reduce cost and some of the complexity of the cloud platform that the USB-3 SSD network drive could be eliminated from the solution; however, having centralized storage does have one major advantage. When using local storage, if one of the Pods fail, you will have to redeploy your application because Kubernetes will bring your Pod up another Node or Pod and therefore your application code will be lost.

3 Cloud Engineering

With all the IaaS cloud services and IT related tasks now complete the focus will shift to all the software that is required to be installed on each of the nodes to create an actual working cloud platform. It will take about 1 to 2 hours to complete installing all the software and configuration on the cluster. The total cost of the software will be $0. The following is a list of all the software installations that will be performed on the nodes in the cluster:

TABLE 3
Cloud Platform Software

Software	Vendor	Cost
Kubernetes K8s v1.21	kubernetes.io	$0
Docker CE v18.09	docker.com	$0
Samba	raspberrypi.org	$0
Proxy	traefik.io	$0

The following are the high-level tasks that need to be completed to setup the master node. Refer to the detailed step-by-step instructions available as noted in the references [1].

1. Install Docker.
2. Install Kubernetes.
3. Setup the Kubernetes Flannel Network.
4. Install Samba.
5. Prepare the network storage.
6. Setup the Samba network share.
7. Install the Kubernetes CIFS Driver.
8. Setup the Kubernetes Persistent Volumes.
9. Setup the Kubernetes Persistent Claims.
10. Setup the Traefik Proxy.
11. Setup the scheduled Cron Jobs.

The following are the high-level tasks that need to be completed to setup each of the worker nodes. Refer to the detailed step-by-step instructions available as noted in the references [1].

1. Install Docker.
2. Install Kubernetes.
3. Install Samba.
4. Join the Kubernetes cluster with a Join Token.
5. Prepare the local storage.
6. Setup the scheduled Cron Jobs.

Within just 3 to 4 hours, you now have a functional cloud platform. To validate that the Kubernetes control plane and all worker nodes are connected to the cluster the following commands can be run from an SSH session connected to the master node:

- kubectl get pods --output=wide
- kubectl get all --all-namespaces

4 Platform as a Service

We now continue with the completion of the cloud platform and where this reference design differentiates itself from other articles written on how to use Raspberry Pi's to build a cloud platform. We will now add PaaS computing services to the cloud platform. The PaaS cloud computing services will provide a complete development and deployment environment in the cloud platform that enable developers to deliver cloud-based web and database applications. After all, what good is a cloud platform if a developer cannot easily access the cloud to deploy applications or that does not allow an end user to easily run applications from the cloud?

To enable PaaS computing services in the cloud platform requires that the following capabilities are available for developers and end users:

1. An easy-to-use end user Portal application that at a minimum provides the following features:
   a. Ability to select available application runtimes.
   b. Ability to select available database runtimes.
   c. Ability to configure the application and database runtimes with desired resource constraints and minimal configuration parameters.
   d. Ability to deploy application code.
   e. Ability to delete an application and database.
   f. Ability to easily access and run an application.
2. Ability for a Portal application to communicate with the Kubernetes control plane to perform various tasks, such as cloud service provisioning and service de-provisioning, that would require no user intervention.
3. Ability for the cloud platform to be easily extended by allowing cloud administrators to easily add new application and database runtimes.
4. Ability for the cloud resources to be easily monitored.

The following application and database runtimes were configured and tested on the cloud platform reference design:

- Web Server runtimes:
  - o Nginx and Apache
- Web Application Server runtimes:
  - o Java Tomcat
  - o ASP.NET Core
  - o Apache PHP
  - o Apache PHP with Laravel
- Database runtimes:
  - o MySQL
  - o PostgreSQL
  - o MongoDB
- Tools:
  - Visual Studio Code
  - NodeJS Development Server

Additional runtimes can be easily added to extend the PaaS cloud service offerings by finding available Docker images from the public Docker Hub repository that are compliant with the Raspberry Pi 4 and the 32-bit Raspbian OS (Arm32v6 or Arm32v7), testing those Docker images using the Java console application provided in the reference design, and then simply inserting a row into a table in the Portal application database. It should be noted that you can also build your own Docker images, push the Docker image to your public Docker Hub repository, and reference the Docker image from within the Portal application database.

5 Portal Application

The Portal application is a secure web-based application that developers can access to provision application runtimes and deploy their application code. End users can also access the Portal application to easily run applications but for simplicity these users could simply be given the URL of the application deployed on the cloud platform. Refer to the detailed step-by-step instructions available as noted in the references [1] for how to download, configure, build, and run the Portal application. The Portal application was designed using current technologies that include Java 15, Spring Boot, Thymeleaf templates, and Bootstrap. The Portal application is backed by a MySQL 8.0 database. Because the Portal application was built with Spring Boot, which allows web applications to be run in a standalone environment, the Portal application can be easily hosted on a shared server located in your own private network or on the cloud administrators, developers, or end users PC.

The Portal application supports the following features:

- Registration of new users.
- Log in of users.
- Access to the PaaS cloud services catalog.
- Ability to select and provision a desired runtime from the PaaS cloud services catalog.
- Ability to select from three different runtime container sizes (i.e., CPU and memory provisioned).
- Ability to deploy application code to a runtime.
- Ability to delete an application and its container.
- Ability to manage users.
- Secure web application design.
- Responsive web application design.

The Portal application was designed using a custom cloud services Java class. This class acted as a façade to hide the internal implementation that leveraged the complex Fabric8 Kubernetes API library. This class and library provided the ability for the Portal application to programmatically communicate directly with the Kubernetes control plane to perform all the necessary Kubernetes deployment, service, and pod management on the cloud platform.

The following test applications, configured with various CPU and memory configurations, were deployed on the cloud platform reference design to validate its functionality. All test applications performed with very reasonable and acceptable response times.

- Web Site on Nginx web server.
- Web Site on Apache web server.
- MySQL 5 relational database.
- Spring 4 on Java 8 and MySQL web application.
- Spring Boot 2 on Java 15 and MySQL web application.
- PHP 7.1 web application.
- PHP 7.4 and MySQL web application.
- PHP Laravel web application.
- Angular 7.2 web application.
- React 16.13 web application.
- ASP.NET Core 5.0 web application.

The following is a screenshot of the Portal application that is provided in the reference design.

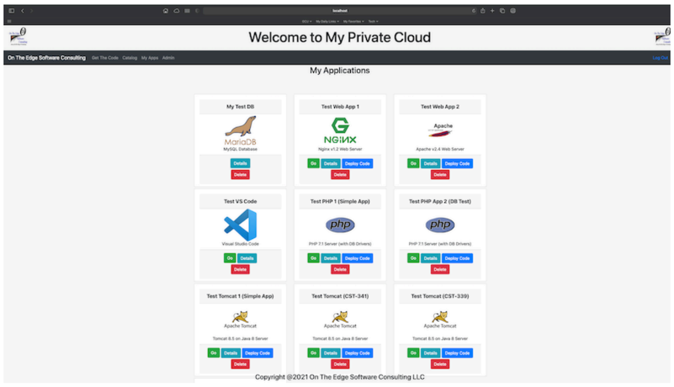

Fig. 4. Portal Application

6 Cloud Monitoring

Cloud administrators can monitor and troubleshoot issues on the cloud platform by using the free Lens desktop application. Refer to the detailed step-by-step instructions available as noted in the references [1] for how to download, configure, and run the Lens application. The Lens application is a powerful application that provides the following features:

- Ability to easily install Prometheus, which is used for monitoring and gather metrics across the entire cluster.
- Ability to monitor resources being consumed on the master node and all worker nodes.
- Ability to monitor the health of the master node and all worker nodes.
- Ability to directly access any of the Kubernetes pods to troubleshoot issues to:
  - View pod logs and application logs.
  - Open a shell into a pod.
- Ability to restart Kubernetes deployments.
- Ability to setup cloud storage.

The following is a screenshot of the Lens application that is supported by the reference design.

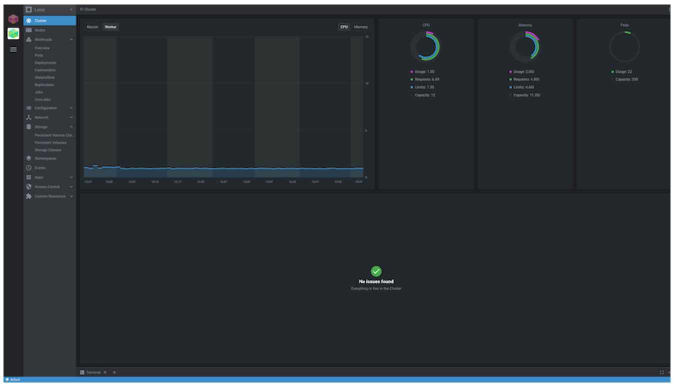

Fig. 5. Lens Application

7 Future Enhancements

There are several refinements to the initial release of the cloud platform reference design that could be made to improve the design. A few of these improvements and enhancements include the following:

- Improve and tighten up the network and security.
- Configure and enable DNS.
- Disable VNC and Desktop on all worker nodes.
- Add support for horizontal scaling from the Portal.
- Add ability to add new runtimes from the Portal.
- Add ability to edit existing runtimes from the Portal.
- Upgrade to the 64-bit Raspbian OS.

8 Conclusions

As mentioned throughout this paper all of the source code and detailed step-by-step instructions for assembling and building this reference design are available on the authors public GitHub repository [1]. You can also see a video demonstration of this fully functioning cloud platform based on this reference design on the authors YouTube channel [2].

For about $500 and in an afternoon, you can build your own fully functioning extendable private cloud platform with PaaS cloud computing services that can be used in a variety of scenarios. A few scenarios could include but not limited to personal use, teaching cloud computing concepts and design in academia, used to host applications for small businesses, and for use in industrial applications.

References

[1] M. Reha, (2021) "My Private Cloud" [GitHub repository]. https://github.com/markreha/myprivatecloud.

[2] M. Reha, (2021) "My Private Cloud" [YouTube video]. https://youtu.be/b0HLzCCKaRU.

[3] M. Reha, (2021) "My Home Page" [Personal website]. https://www.ontheedgesc.com/personal/.

Mark K. Reha My professional background after graduating with a Bachelor of Science in Electrical Engineering started in 1982 where I spent almost 35 years in Embedded Systems development, Windows development, Enterprise development, Enterprise architecture, and Technical Management. Shortly after obtaining my Master's in Education/Adult Education and Training in 2011 I joined Grand Canyon University (GCU) where I am part of the full-time faculty and an Assistant Professor and Program Chair for the Bachelor of Science in Software Development and the Master of Science in Software Development programs. I am the author of a book and have published several peer reviewed journal articles in the software design and development domain. I am currently not a member of the IEEE and the IEEE Computer Society.