RaaSI: Users Guide

Michael Coffey

July 5, 2022

Contents

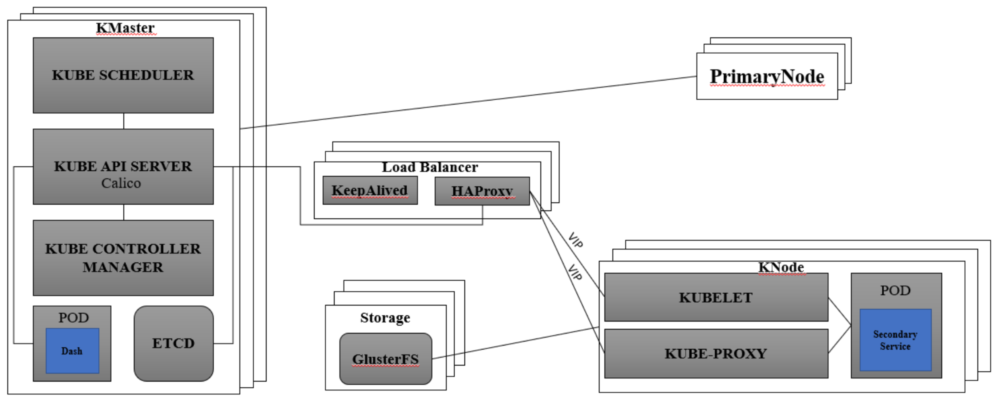
1 Introduction

The purpose of this document is to show readers the use cases of RaaSI and how to execute each use case. However, before the use cases of RaaSI are discussed, the reader must have an understanding of the architecture and modes of RaaSI, along with the requirements needed for machines to belong to a RaaSI cluster. After this is discussed, the reader will see how to install RaaSI in section 2. In section 3 managing secondary services is discussed and in section 4 managing a primary service is shown. In order to download RaaSI please visit [2].

1.1 Architecture

The architecture of RaaSI is split into four distinct parts; the RaaSI Master Nodes (RMN), RaaSI Worker Nodes (RWN), RaaSI Load Balancer (RLB), and RaaSI Storage Nodes (RSN). The culmination of these parts make up the RaaSI cluster. This cluster enables RaaSI to monitor primary services, deploy and execute secondary services, maintain the state of secondary services, and provide high-availability throughout the RaaSI cluster. The interconnection of the RaaSI cluster is shown in 1.

Figure 1: RaaSI Architecture Diagram



1.1.1 RaaSI Master Nodes (RMN)

Kubernetes (the underlying architecture of RaaSI) is designed as a Master Slave Architecture [6], where the Master Nodes monitor and manage the Worker Nodes in the cluster. In the RaaSI cluster, the Master Nodes are labeled as RMN and the Worker Nodes are labeled as RWN. Due to the responsibility of the RMN, services that are essential to the monitoring and management of the RWN are set up on these nodes. These services are the Kube API Server, Kube Controller Manager, Kube Scheduler, and ETCD. In order to successfully manage the primary services, a primary service monitoring API was developed to check the health of primary services and is stored on the RMN. Finally, the a CLI to manage RaaSI is also stored on the RMN. This CLI allows users to step through RaaSI with ease.

1.1.2 RaaSI Load Balancer (RLB)

In order to provide high-availability (HA) to the RMN the RLB is created and used to manage the multiple controller nodes on the RMN. However, if nothing else is done, this does not resolve the HA problem, it merely transfers the problem from the RMN to the RLB. To fix this, HA must be kept on the RLB as well. For HA to be maintained on the RMN and the RLB keepalived and haproxy [4] are run on the RLB. Three instances of Load Balancers are created to provide HA on the RLB.

1.1.3 RaaSI Storage Nodes (RSN)

When running secondary services on RaaSI, statefulness of these secondary services is crucial. If the state of theses services is not preserved, the information that they gathered while they were running is useless. To ensure this statefulness, RaaSI connects RaaSI Worker Nodes (RWN) directly to a distributed file system stored on the RSN. By default the distributed file system used by RaaSI is GlusterFS. However, users can use whatever distributed file system they want if they download it directly to the nodes that they intend on running it.

1.1.4 RaaSI Worker Nodes (RWN)

The RWN are nodes that existed on the infrastructure prior to RaaSI that RaaSI is using to run secondary services. These nodes serve some other purpose for the original infrastructure, the computational ability of these nodes that is not being used for the nodes primary purpose is what the kubernetes worker node and the secondary service is being run on. These nodes are added to the RaaSI cluster through the CLI run on the RMN. There are three essential services related to the kubernetes worker node that are being run on each of the RWN; Kubelet, Kube-Proxy, and the secondary service pod.

1.2 Modes

In order to provide users with the freedom to configure their RaaSI cluster in ways that best fits their environment, we have provided users with two modes to run their RaaSI cluster in, standard mode and high availability mode. It is recommended that users run their RaaSI cluster in high availability mode in order to provide the maximize the resiliency of RaaSI; however, if resources are low, users can also run RaaSI in its standard mode.

1.2.1 Standard

The standard mode of RaaSI eliminates the use of the RLB and multiple RMN. In this mode, a single RMN connects directly to the RWN to manage the RaaSI cluster. The problem with this mode is that only having one RMN provides a single point of failure and if this node fails, the RaaSI cluster will fail with it. Also, only having one RMN can act as a bottleneck for the data-flow in the RaaSI cluster, due to all of the RWN communicating with only one RMN instead of splitting the traffic between multiple.

1.2.2 High Availability

For the high availability mode to be set up, the RaaSI cluster must use the RLB and have a minimum of three RMN. The number of RMN can very, depending on how many RMN the user wants to add. When multiple RMN are used, a single RMN is no longer a single point of failure since all RMN would need to fail in order for the RaaSI cluster to fail. This mode is also more efficient in data communication between the RWN and the RMN due to the traffic being spread across all nodes in the RMN.

1.3 Requirements

In order to run RaaSI, certain system requirements must be met to ensure that the nodes in RaaSI have enough resources to successfully undergo the responsibilities required of them. As previously discussed, RaaSI is able to run in two modes, standard and high availability. The system requirements of these two modes due to the high availability mode needing more resources than the standard mode. Regardless of the mode, it is recommended to separate the management nodes (i.e., RMN and RLB) from the remainder of RaaSI. This separation provides decoupling between the management and execution layers of RaaSI. In this section, the requirements of the standard and high availability modes of RaaSI are discussed in the form of storage, CPU, characteristics, and connectivity.

1.3.1 Standard Mode

For the standard mode of RaaSI to be run, users need to have a single RMN, at least three RSN, and multiple RWN. Each RMN used in RaaSI should have the following available; 2GB or more of RAM, 1.5 or more cores, network connectivity to all other nodes in RaaSI, ssh connectivity (for installation and management), and a unique hostname, MAC address, and product uuid [5]. Each RSN used in RaaSI should have the following available; 1GB or more of RAM, 1 or more core, ssh connectivity (for installation and management), and network connectivity to the RWN [3]. Finally, each RWN used in RaaSI should have the following available; 1GB or more of RAM, 1 or more core, network connectivity to RMN and RSN, ssh connectivity (for installation and management), and a unique hostname, MAC address, and product uuid [5]. If a node is being used as a RSN and a RWN, the requirements for the RSN and RWN must both be fulfilled, that is; 2GB or more of RAM, 2 or more cores, network connectivity to RMN, RSN, and RWN, ssh connectivity (for installation and management), and a unique hostname, MAC address, and product uuid.

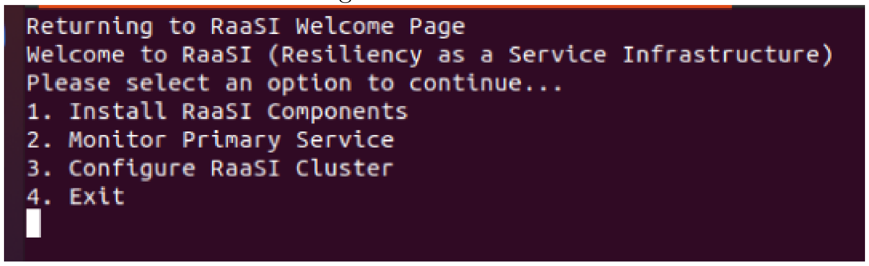
1.3.2 High Availability Mode

For RaaSI to be run in high availability mode, users must have at least three RMN, three RLB nodes, at least three RSN, and multiple RWN. Each RMN used in RaaSI should have the following available; 2GB or more of RAM, 1.5 or more cores, network connectivity to all other nodes in RaaSI, ssh connectivity (for installation and management), and a unique hostname, MAC address, and product uuid [5]. Each RLB used in RaaSI should have the following available; 1GB or more of RAM, 1 or more core, network connectivity to RMN and RWN, ssh connectivity (for installation and management), and a unique hostname, MAC address, and product uuid. Each RSN used in RaaSI should have the following available; 1GB or more of RAM, 1 or more core, ssh connectivity (for installation and management), and network connectivity to the RWN [3]. Finally, each RWN used in RaaSI should have the following available; 1GB or more of RAM, 1 or more core, network connectivity to RMN and RSN, ssh connectivity (for installation and management), and a unique hostname, MAC address, and product uuid [5]. If a node is being used as a RSN and a RWN, the requirements for the RSN and RWN must both be fulfilled, that is; 2GB or more of RAM, 2 or more cores, network connectivity to RMN, RSN, and RWN, ssh connectivity (for installation and management), and a unique hostname, MAC address, and product uuid.

2 Installation

After nodes matching the requirements for RaaSI have been acquired and RaaSI is downloaded from [2], users will need to install the components necessary for RaaSI to begin execution. To do so, users can navigate to the RaaSIController.sh script found in the RaaSIController directory. This script begins the RaaSI CLI as shown in 2. Once the user has started the RaaSI CLI, they can select option 1, "Install RaaSI Components", to begin installing the components of RaaSI. This option will direct the user to the RaaSI Component Installation CLI shown

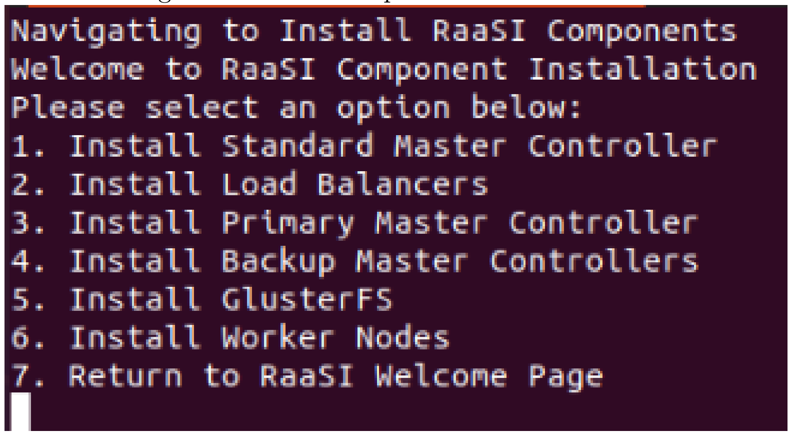
Figure 2: RaaSI CLI



```
Returning to RaaSI Welcome Page
Welcome to RaaSI (Resiliency as a Service Infrastructure)
Please select an option to continue...
1. Install RaaSI Components
2. Monitor Primary Service
3. Configure RaaSI Cluster
4. Exit
```

in 3.

Figure 3: RaaSI Component Installation CLI

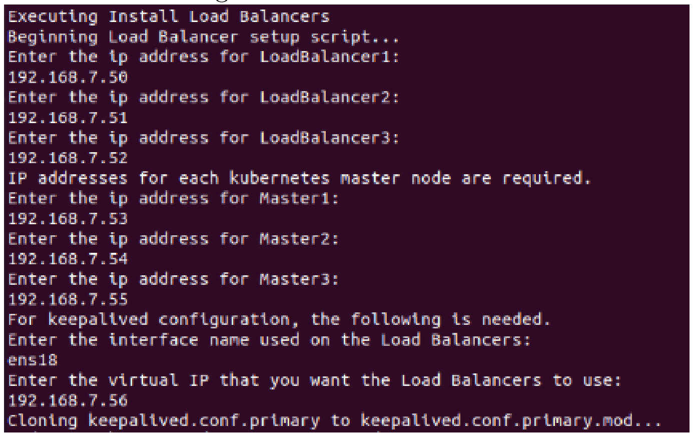


```
Navigating to Install RaaSI Components
Welcome to RaaSI Component Installation
Please select an option below:
1. Install Standard Master Controller
2. Install Load Balancers
3. Install Primary Master Controller
4. Install Backup Master Controllers
5. Install GlusterFS
6. Install Worker Nodes
7. Return to RaaSI Welcome Page
```

2.1 Load Balancers

To begin the installation of RLB nodes, select option 2 in the RaaSI Component Installation CLI labeled ”Install Load Balancers.” Once the RLB installation script begins, follow the prompts to properly set up the RLB. The prompts will ask for the IP address of the three nodes that the user wants to set up as the RLB nodes. It will then ask the user for the IP addresses of the RMN that the RLB will be connecting to. The RMN does not need to be installed at this point; however, the RLB needs the IP addresses that will be used for the RMN in order to balance the traffic between the nodes in the RMN. The RLB installation script will then ask for the virtual IP address (VIP) to be used as a proxy for the RMN and the interface to put this VIP on. Finally, the script will prompt the user to provide the username and password for a user that can ssh to each of the nodes in the RLB. Once all of this information has been collected by the script, the RLB will be automatically installed on the nodes provided to the script. This script takes about five minutes to complete execution. An example of the running RLB installation script is shown in 4.

Figure 4: RLB Installation



```
Executing Install Load Balancers
Beginning Load Balancer setup script...
Enter the ip address for LoadBalancer1:
192.168.7.50
Enter the ip address for LoadBalancer2:
192.168.7.51
Enter the ip address for LoadBalancer3:
192.168.7.52
IP addresses for each kubernetes master node are required.
Enter the ip address for Master1:
192.168.7.53
Enter the ip address for Master2:
192.168.7.54
Enter the ip address for Master3:
192.168.7.55
For keepalived configuration, the following is needed.
Enter the interface name used on the Load Balancers:
ens18
Enter the virtual IP that you want the Load Balancers to use:
192.168.7.56
Cloning keepalived.conf.primary to keepalived.conf.primary.mod...
```

2.2 Master Nodes

Master Nodes are devices that manage the state of RaaSI. These nodes are responsible for creating secondary service pods, distributing secondary service pods to worker nodes, observing worker nodes and their health metrics, running scripts to setup and configure RaaSI, hosting the Primary Service Monitoring API, starting secondary services on worker nodes when a primary service is deprecated, and stopping secondary services when primary services come back online. If RaaSI is being run in standard mode, users will only install one Master Node; however, if RaaSI is run in high availability mode, users can set up as many Master Nodes as they want. The benefit of setting up multiple Master Nodes is that the communication between Master Nodes and Worker Nodes is distributed between the accessible Master Nodes. This distribution of communication alleviates the bottleneck of one Master Node being responsible for all computation and communication to a multitude of Worker Nodes.

2.2.1 Standard Mode Master Node

In order to initiate the Standard Master Controller installation script, the user should select option 1, ”Install Standard Master Controller,” from the ”Install RaaSI Components” page in the RaaSI CLI. Note that this script will be run on the local device that it is initiated on, so users should copy the RaaSI CLI to the Master Node that they are wanting to setup. Once the script is initiated, the user will be guided through each phase of the installation.

The user will first be asked if they have completed the ”Initial System Configuration” step. This step updates the device and turns off swap from the system if the user has not already done so. After this step is completed, the user will be asked if they have completed the "Kubernetes Installation" step. If the user has not, the script will install the required Kubernetes components to the system; kubelet, kubectl, and kubeadm. Next the script will ask users if they have completed the "Kubernetes Configuration" step. This step will update some kubernetes configuration files with the appropriate settings. Finally, the script will run the commands needed to make the device a Master Node through the use of kubeadm, install an overlay network used for communication throughout the cluster, install a device manager that allows pods to interact with physical devices on Worker Nodes, and install a dashboard that users can view to see metrics on the RaaSI cluster. This script takes about seven minutes to complete execution.

2.2.2 Primary Master Nodes

If users are wanting to run RaaSI in high availability mode with multiple Master Nodes, they first need to install a Primary Master Node that will act as the link between Master Nodes to the Load Balancer. To install the Primary Master Node the user should select option 3, "Install Primary Master Controller," on the "Install RaaSI Components" in the RaaSI CLI. Note that this script will be run on the local device that it is initiated on, so users should copy the RaaSI CLI to the Master Node that they are wanting to setup. Once the scrip is initiated, the user will be guided through each phase of the installation. This installation process is similar to that of installing a standard mode Master Node with a few exceptions.

The user will first be asked if they have completed the "Initial System Configuration" step. This step updates the device and turns off swap from the system if the user has not already done so. After this step is completed, the user will be asked if they have completed the "Kubernetes Installation" step. If the user has not, the script will install the required Kubernetes components to the system; kubelet, kubectl, and kubeadm. Next the script will ask users if they have completed the "Kubernetes Configuration" step. This step will update some kubernetes configuration files with the appropriate settings. Once Kubernetes is configured properly, the script will ask the user for the IP address and port of the Load Balancer. For this, the user should input the IP address they chose for the Virtual IP (VIP) when setting up the Load Balancer. The port that the user should use is 6443, unless this port was manually changed. Finally, the script will run the commands needed to make the device a Master Node through the use of kubeadm while connecting it to the Load Balancer, install an overlay network used for communication throughout the cluster, install a device manager that allows pods to interact with physical devices on Worker Nodes, and install a dashboard that users can view to see metrics on the RaaSI cluster. This script takes about seven minutes to complete execution.

2.2.3 Backup Master Nodes

After the Primary Master Node has been setup, users that are running RaaSI in high availability mode should select option 4, "Install Backup Master Controllers," on the "Install RaaSI Components" page in the RaaSI CLI. This option will install another Master Node that will be used along side the Primary Master Node to provide high availability and limit bottle-necking. This script can be run on a remote system, but needs to be executed for each additional Master Node that is added to RaaSI. The steps of this installation script are very similar to those for the script "Install Standard Master Controller," the only difference in this script is that the kubeadm installation command attaches the Master Node to the Primary Master Node through an added argument. This script takes about seven minutes to complete execution.

2.3 Stateful Storage

Setting up stateful storage should be completed before Worker Nodes are created due to the Worker Nodes connecting to the stateful storage during their installation. To initiate the setup of stateful storage, the user should select option 5, "Install GlusterFS," on the "Install RaaSI Components" page in the RaaSI CLI. This script will ask the user for three IP addresses of devices that they would like to setup GlusterFS on. The script will also ask the user for the name of a user that can be connected to on there devices through ssh. Once the user has given the script this information, the script will execute on each desired device and install GlusterFS while setting up GlusterFS data encryption through TLS. This script takes about twelve minutes to complete execution. The stateful storage that RaaSI uses by default is GlusterFS; however, if the user wants to use a different method for stateful storage they can manually configure their own storage and connect it to the Worker Nodes.

2.4 Worker Nodes

The final component to be installed in the RaaSI cluster is the Worker Nodes. To install these components, the user should select option 6, "Install Worker Nodes," on the "Install RaaSI Components" page in the RaaSI CLI. This script can be run remotely to setup each Worker Node. It also needs to be run once for each Worker Node to be setup. Once the option is selected the script will begin execution and the user will be guided through the installation process.

The user will first be asked if they have completed the "Initial System Configuration" step. This step updates the device and turns off swap from the system if the user has not already done so. After this step is completed, the user will be asked if they have completed the "Kubernetes Installation" step. If the user has not, the script will install the required Kubernetes components to the system; kubelet, kubectl, and kubeadm. Next the script will ask users if they have completed the "Kubernetes Configuration" step. This step will update some kubernetes configuration files with the appropriate settings. Once

Kubernetes is configured properly, the script will ask the user to input the IP addresses that were used for GlusterFS installation along with users that can access these devices through ssh. The script will then connect the device that is being turned into a Worker Node to its GlusterFS cluster as a client, while encrypting its connection through TLS. After this, the script will run the commands needed to make the device a Worker Node through the use of the kubeadm join command generated by the Master Node. Once the device has connected to the cluster as a Worker Node, the script will ask the user if they want to search for connected devices on this Worker Node. These connected devices will be used to determine what secondary service should go on the node. If the user selects "Y," the script will prompt the user to enter the name of the device to search for. Once this information is given, the script will check if the Worker Node has this device connected through the lspci command. If the node is connected to the requested device, it will be labeled with that information to be used while setting up the secondary service. This script takes about seven minutes to complete execution.

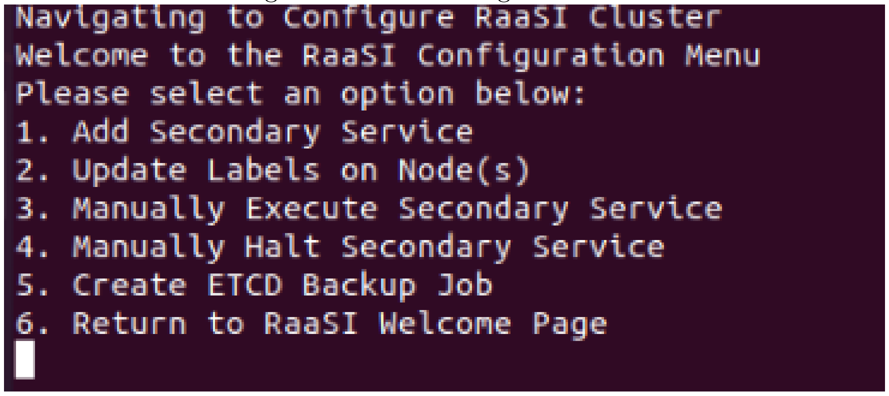
3 Configuring the RaaSI Cluster

Once RaaSI has been successfully installed, users can begin configuring their RaaSI cluster to best fit their environment. This configuration is done through the CLI on the "Configure RaaSI Cluster" page (option 3 on the welcome page). After the user selects this option on the welcome page of the RaaSI CLI, they will be presented with six configuration options shown in figure 5. The user can create a new secondary service by selection option 1 "Add Secondary Service," they can search through the Worker Nodes of RaaSI to find and label nodes with devices that are required for certain secondary services through option 2 "Update Labels on Node(s)," they can manually start or stop the execution of a specific secondary service on Worker Nodes by selection option 3 "Manually Execute Secondary Service" and option 4 "Manually Halt Secondary Service," and they can create a backup job for ETCD that will continuously run in the background through selecting option 5 "Create ETCD Backup Job."

3.1 Creating a Secondary Service

Secondary services are the services that RaaSI brings online over the Worker Nodes in order to provide graceful degradation for the primary service that they are created to support. These services are created by users of RaaSI through containerized images and are placed on Docker Hub. When creating a containerized image for your secondary service, make sure that your image is continuously executing some command. If the image does not continuously run, the deployment of the image will fail to deploy due to kubernetes constraints. Once the containerized image has been created and placed on Docker Hub, the user can execute option 1, "Add Secondary Service," on the "Configure RaaSI Cluster" page in the RaaSI CLI.

Figure 5: RaaSI Configuration CLI



```
Navigating to Configure RaaSI Cluster
Welcome to the RaaSI Configuration Menu
Please select an option below:
1. Add Secondary Service
2. Update Labels on Node(s)
3. Manually Execute Secondary Service
4. Manually Halt Secondary Service
5. Create ETCD Backup Job
6. Return to RaaSI Welcome Page
```

This script will walk the user through the creation of a deployment for their secondary service. The script will ask the user what the name of the secondary service should be, the name of the device that the secondary service uses to run (this is the device that was labeled on nodes when they were created), the Docker Hub image location, the number of the port that is required to be open for this service to run (if there is not any port required to be open, the user should type 80 here), the minimum and maximum amount of memory needed for this service to run (in GB), the minimum and maximum amount of CPU needed for this service to run, and the location of the stateful storage mounted drive that the service can access (by default this is /gv0). The script will then generate a yaml file that supports the deployment requirements that were previously specified and start the deployment with zero replicas. The deployment is started with zero replicas in order to prevent its instant execution across Worker Nodes. In order to deploy these secondary services across their assigned Worker Nodes, the user should select the option ”Manually Execute Secondary Service” or setup automatic deployment through the Primary Service Monitoring API. RaaSI uses Docker Hub to pull and execute secondary services by default; however, users can also create their own kubernetes deployments and add their own private images to the deployments if that is preferred.

3.2 Updating Node Devices

Worker Nodes are labeled with the devices that are required to run secondary services. This is one of the factors that RaaSI looks at when assigning secondary services to be run on nodes (this along with resources that the node has compared to the resources that the secondary service needs to run). These Worker Nodes can be scanned (with lspci) and labeled with the name of the device when they are first being installed (visit section 2.4 ”Worker Nodes” to read more about this); however, if a new device needs to be searched for and labeled on its appropriate Worker Nodes after the nodes have been installed, users can select option 2, "Update Labels on Node(s)," on the "Configure RaaSI Cluster" page in the RaaSI CLI.

This script will prompt the user for the device that they want to find on nodes and ask the user if they want to look at every Worker Node in the cluster or a specific node. If the user specifies that they want to look for the device on every Worker Node in the cluster, the script will prompt the user for the login credentials for each Worker Node, search that Worker Node for the device, and label the node with the name of the device if it is found. If the user wants to search for the device on a specific node, the script will ask them for the IP address along with the login credentials for the specific node in question, search the node for the device, and label it with the name of the device if it is found. The user can also choose to manually label nodes with the name of the device that the secondary service is looking with kubernetes commands. This can be done by adding the label "device=$< deviceName >$" to the node that they want.

3.3 Secondary Service Deployment

In order for secondary services to be effective in RaaSI, they need to be deployed to the Worker Nodes that are able to support them. This deployment can either be automatic or manual. If users setup the primary service monitoring API, they can specify which secondary services should be automatically triggered when a primary service is degraded. However, users can also choose to manually deploy and halt these secondary services.

3.3.1 Manually Deploy Secondary Service

Manually deploying a secondary service can help to speed up the performance of RaaSI. The first time that a secondary service is deployed it can take up to 30 minutes before it is ready to execute on the Worker Nodes that it was deployed on. This delay is because it takes a long time to pull these containers from Docker Hub (especially if they are large in size). However, after these containers are deployed for the first time, it becomes much quicker to deploy them for subsequent use. Due to this, it is recommended that users manually deploy secondary services when they are created to the Worker Nodes that will be housing them. This can be accomplished by selecting option 3, "Manually Execute Secondary Service," on the "Configure RaaSI Cluster" page in the RaaSI CLI. Once this script is initiated, it will ask the user for the name of the secondary service and the device that the secondary service should search for. The script then executed this deployment on all nodes that have this device along with the required resources that were specified in the setup of the secondary service.

3.3.2 Manually Halt Secondary Service

After secondary services are manually deployed to increase the performance of RaaSI, they should be manually halted. To manually halt a secondary service deployment, the users should select option 4, "Manually Halt Secondary Service," on the "Configure RaaSI Cluster" page in the RaaSI CLI. This script will ask the user for the name of the secondary service that they are attempting to halt. Once this information is given, the script will halt the deployment on all Worker Nodes that were running the secondary service.

3.4 Backing Up the ETCD

Kubernetes stores the state of the cluster and information regarding deployments in the ETCD. This stateful storage of kubernetes is placed in a pod and kept on one of the Master Nodes. However, if all Master Nodes fail, the information stored in the ETCD might be lost. This is why it is important to backup the ETCD periodically. RaaSI allows its users to do this by selecting option 5, "Create ETCD Backup Job," on the "Configure RaaSI Cluster" page in the RaaSI CLI. Once this script is run, it asks for the IP address and login credentials for the remote device to send the ETCD backup to. Once this information is given, ssh pubkey authentication is setup between the remote machine and the Master Node that is running the script. This pubkey authentication allows the script to periodically send the backup through scp to the remote machine. This backup is set to occur every 30 minutes; however, users can change this time by modifying the "doModify.sh" script. The backup is also stored in the "backup" directory in the home of the user that was provided when first executing the "Create ETCD Backup Job" script.

4 Managing Primary Services

Primary services are the essential services that RaaSI is providing graceful degradation towards in the form of secondary services. These primary services must be monitored in order to determine when they are degraded enough for the secondary services that are supporting them should begin execution. The ability to monitor these primary services and execute secondary services when their attached primary service is degraded is done through the primary service monitoring API.

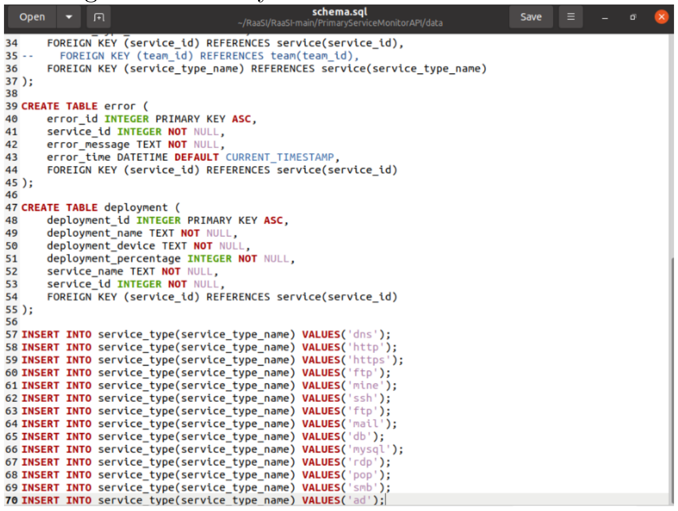
4.1 Primary Service Monitor API

The primary service monitoring API is a derivation from a judging software designed for cybersecurity competition to monitor the health of services [1]. The software for the primary service monitoring API is written in Python 3, bash, and yaml. To use the primary service monitoring API, users first must add primary services to monitor and how to monitor them the the API, along with the secondary service deployments that are associated with these primary services. After this, the user should initialize the primary service monitoring API through the RaaSI CLI. Finally, the user can update the services that are being monitored and deployed at any time by run the update script found in the primary service monitoring API.

4.1.1 Configure New Service

To monitor a new primary service with the primary service monitoring API, users must add the primary service to the database used by the API, add the method used to monitor the health of the service, and add the new service to "all-services.yaml" file. First, the user should check the "PrimaryServiceMonitoringAPI/data/schema.sql" file to see if their service name is already in the database. If it is not, the user should add the new service to the bottom of the file in the form of "INSERT INTO service_type(service_type_name) VALUES('$< NewService >$')" (replacing $< NewService >$ with the name of the service to add. This file is shown in figure 6.

Figure 6: Primary Service Monitor API Schema

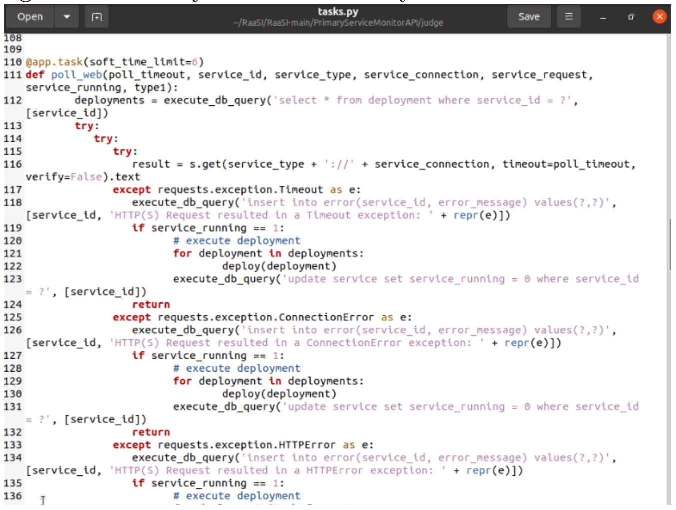


```
    FOREIGN KEY (service_id) REFERENCES service(service_id),
--    FOREIGN KEY (team_id) REFERENCES team(team_id),
    FOREIGN KEY (service_type_name) REFERENCES service(service_type_name)
);

CREATE TABLE error (
    error_id INTEGER PRIMARY KEY ASC,
    service_id INTEGER NOT NULL,
    error_message TEXT NOT NULL,
    error_time DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (service_id) REFERENCES service(service_id)
);

CREATE TABLE deployment (
    deployment_id INTEGER PRIMARY KEY ASC,
    deployment_name TEXT NOT NULL,
    deployment_device TEXT NOT NULL,
    deployment_percentage INTEGER NOT NULL,
    service_name TEXT NOT NULL,
    service_id INTEGER NOT NULL,
    FOREIGN KEY (service_id) REFERENCES service(service_id)
);

INSERT INTO service_type(service_type_name) VALUES('dns');
INSERT INTO service_type(service_type_name) VALUES('http');
INSERT INTO service_type(service_type_name) VALUES('https');
INSERT INTO service_type(service_type_name) VALUES('ftp');
INSERT INTO service_type(service_type_name) VALUES('mine');
INSERT INTO service_type(service_type_name) VALUES('ssh');
INSERT INTO service_type(service_type_name) VALUES('ftp');
INSERT INTO service_type(service_type_name) VALUES('mail');
INSERT INTO service_type(service_type_name) VALUES('db');
INSERT INTO service_type(service_type_name) VALUES('mysql');
INSERT INTO service_type(service_type_name) VALUES('rdp');
INSERT INTO service_type(service_type_name) VALUES('pop');
INSERT INTO service_type(service_type_name) VALUES('smb');
INSERT INTO service_type(service_type_name) VALUES('ad');
```

After the new service has been added to the database, the user should add a health monitoring method for the service. This can be done in the "PrimaryServiceMonitoringAPI/judge/tasks.py" file. In this file the user should create a new method that checks the health of the service that they are adding. Examples of these methods are shown throughout this file such as monitoring a web server shown in figure 7. Once this method has been created, the user should add the reference to their new service and the health monitoring method that is used for the service to the "poll()" method as shown in figure 8.

Finally, the user should navigate to the PrimaryServiceMonitoringAPI/all-services.yaml file and add their new primary service to the file under the "services" section of the file as shown in figure 9. In this file, the variable "ser-

Figure 7: Primary Service Monitory API Health Monitor 2



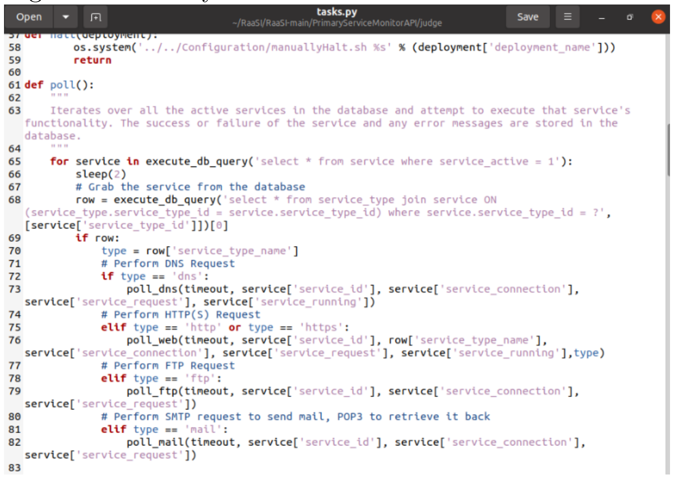
```
@app.task(soft_time_limit=6)
def poll_web(poll_timeout, service_id, service_type, service_connection, service_request, service_running, type1):
        deployments = execute_db_query('select * from deployment where service_id = ?', [service_id])
        try:
           try:
              try:
                 result = s.get(service_type + '://' + service_connection, timeout=poll_timeout, verify=False).text
              except requests.exception.Timeout as e:
                 execute_db_query('insert into error(service_id, error_message) values(?,?)', [service_id, 'HTTP(S) Request resulted in a Timeout exception: ' + repr(e)])
                 if service_running == 1:
                        # execute deployment
                        for deployment in deployments:
                               deploy(deployment)
                        execute_db_query('update service set service_running = 0 where service_id = ?', [service_id])
                 return
              except requests.exception.ConnectionError as e:
                 execute_db_query('insert into error(service_id, error_message) values(?,?)', [service_id, 'HTTP(S) Request resulted in a ConnectionError exception: ' + repr(e)])
                 if service_running == 1:
                        # execute deployment
                        for deployment in deployments:
                               deploy(deployment)
                        execute_db_query('update service set service_running = 0 where service_id = ?', [service_id])
                 return
              except requests.exception.HTTPError as e:
                 execute_db_query('insert into error(service_id, error_message) values(?,?)', [service_id, 'HTTP(S) Request resulted in a HTTPError exception: ' + repr(e)])
                 if service_running == 1:
                        # execute deployment
```

vice_type_name" is the name of the service type that you added to the database, the variable "service_name" is a reference name to the primary service (it's important to give this a unique name if there are multiple primary services of the same type), the variable "service_connection" is the location of this primary service (this can be an IP address or DNS hostname), and the variable "service_request" is used to add arguments to the health monitor check. The example shown curls google.com and checks if "home" is anywhere in the response.

4.1.2 Add Deployments

After primary services have been added to the primary service monitoring API, secondary service deployments should be connected to these primary services. This can be done by modifying the "PrimaryServiceMonitoringAPI/all-services.yaml" file. This file is shown if figure 9. Once in this file, users should add their new deployments under the "deployments" section. In this section the variable "deployment_name" refers to the name of the secondary service deployment that was setup by the user, the variable "deployment_device" refers to the device used to schedule these deployments on Worker Nodes, the variable "deployment_percentage" is the percent (from 0-100) of matching Worker Nodes that should receive these secondary services, and the variable "service_name" is used to map the deployment to the primary service by matching it to the services variable "service_name".

Figure 8: Primary Service Monitor API Heath Monitor 1



```python
def halt(deployment):
        os.system('../../Configuration/manuallyHalt.sh %s' % (deployment['deployment_name']))
        return

def poll():
    """
    Iterates over all the active services in the database and attempt to execute that service's
functionality. The success or failure of the service and any error messages are stored in the
database.
    """
    for service in execute_db_query('select * from service where service_active = 1'):
        sleep(2)
        # Grab the service from the database
        row = execute_db_query('select * from service_type join service ON
(service_type.service_type_id = service.service_type_id) where service.service_type_id = ?',
[service['service_type_id']])[0]
        if row:
            type = row['service_type_name']
            # Perform DNS Request
            if type == 'dns':
                poll_dns(timeout, service['service_id'], service['service_connection'],
service['service_request'], service['service_running'])
            # Perform HTTP(S) Request
            elif type == 'http' or type == 'https':
                poll_web(timeout, service['service_id'], row['service_type_name'],
service['service_connection'], service['service_request'], service['service_running'],type)
            # Perform FTP Request
            elif type == 'ftp':
                poll_ftp(timeout, service['service_id'], service['service_connection'],
service['service_request'])
            # Perform SMTP request to send mail, POP3 to retrieve it back
            elif type == 'mail':
                poll_mail(timeout, service['service_id'], service['service_connection'],
service['service_request'])

```

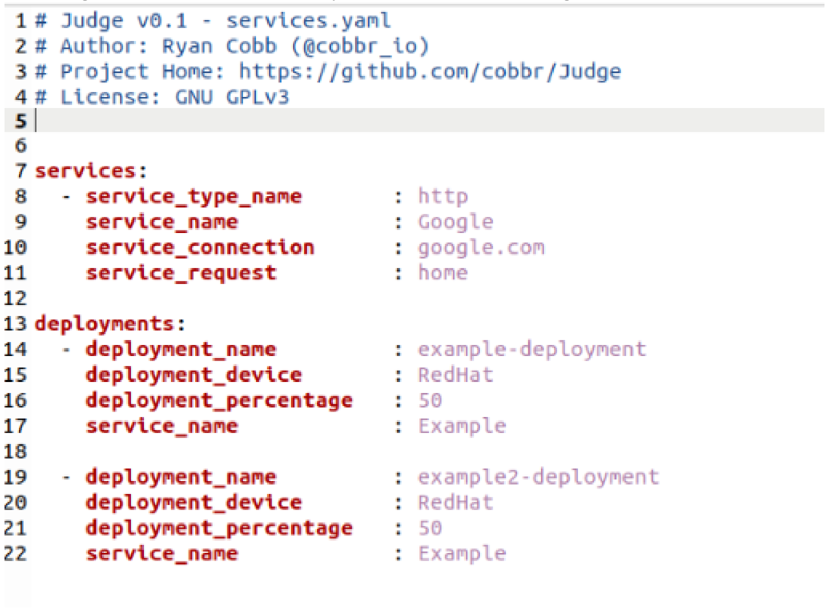
4.1.3 Initialization

Once the primary service monitoring API has been modified for the primary and secondary services used in RaaSI, the user should select option 2, "Monitor Primary Service," on the "RaaSI Welcome Page" of the RaaSI CLI. This will navigate the user to the "Monitor Primary Service" page shown in figure 10. On this page the user can select option 1, "Install the Primary Service Monitoring API" to initialize the primary service monitoring API. This script will distribute the primary service monitoring evenly among all Master Nodes in the RaaSI cluster. It will then install the required software needed to execute the API and start the API on each of the Master Nodes. This API will continuously monitor the health of primary services and when the service is degraded the API will deploy the secondary services associated to that primary service. Once the primary service becomes healthy again, the API will halt the secondary services that are associated. If users have updates to add to the primary service monitoring API, they can manually add them to the API then select option 2, "Update the Primary Service Monitoring API," to publish and execute the update across all Master Nodes.

5 Conclusion

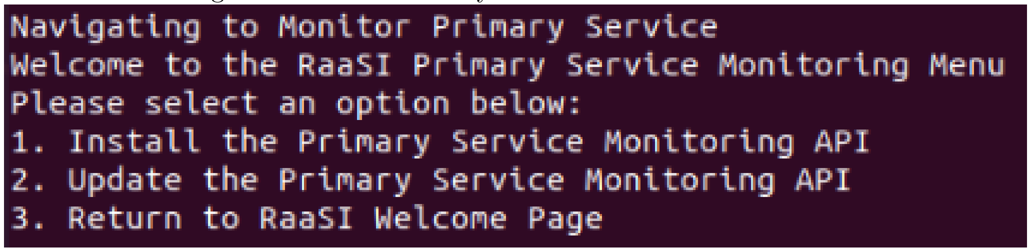
This concludes the RaaSI User Guide. For more detailed information on the aspects touched upon here, users can read the RaaSI Developers Guide. There is also a technical report that has been created demonstrating the need for RaaSI, works that are related to RaaSI, and discusses tests run on RaaSI. That report is titled RaaSI Technical Report.

Figure 9: RaaSI Primary Service Monitoring API All-Services



```
# Judge v0.1 - services.yaml
# Author: Ryan Cobb (@cobbr_io)
# Project Home: https://github.com/cobbr/Judge
# License: GNU GPLv3


services:
  - service_type_name        : http
    service_name             : Google
    service_connection       : google.com
    service_request          : home

deployments:
  - deployment_name          : example-deployment
    deployment_device        : RedHat
    deployment_percentage    : 50
    service_name             : Example

  - deployment_name          : example2-deployment
    deployment_device        : RedHat
    deployment_percentage    : 50
    service_name             : Example
```

Figure 10: RaaSI Primary Service Monitor CLI



```
Navigating to Monitor Primary Service
Welcome to the RaaSI Primary Service Monitoring Menu
Please select an option below:
1. Install the Primary Service Monitoring API
2. Update the Primary Service Monitoring API
3. Return to RaaSI Welcome Page
```

References

[1] Ryan Cobb. *Judge*. Version 0.1. Mar. 2022. URL: `https://github.com/cobbr/judge`.

[2] Michael Coffey. *Resiliency as a Service Infrastructure (RaaSI)*. Version 2.0. Mar. 2022. URL: `https://github.com/CoffeyBean60/RaaSI`.

[3] *Gluster Docs: Overview*. `https://docs.gluster.org/en/main/Install-Guide/Overview/`. Accessed: 2022-05-11.

[4] *High Availability Considerations*. `https://github.com/kubernetes/kubeadm/blob/main/docs/ha-considerations.md`. Accessed: 2022-03-29.

[5] *Kubernetes Cluster Hardware Recommendations*. `https://docs.kublr.com/installation/hardware-recommendation/`. Accessed: 2022-05-11.

[6] *Master/Slave*. `https://www.techopedia.com/definition/2235/masterslave`. Accessed: 2022-03-29.