RaaSI: Developer Guide

Michael Coffey

July 5, 2022

Contents

1 Introduction

The purpose of this document is to give developers who are using RaaSI a more technical look at how the system can be used and customized. For a more generic view of the use cases of RaaSI (including how to get RaaSI installed and configured), users can view the "RaaSI Users Guide." This developers guide will show users how to customize their installation of RaaSI through advanced manipulation, configure their RaaSI cluster to best match their needs, instruct the user on how to develop and manage secondary services used in RaaSI, and give an insight on how the source code of RaaSI is managed.

2 Installation

The basic installation of RaaSI can be completed through the RaaSI CLI and by following the steps detailed in the Users Guide. This installation process can be customized to more closely fit the needs of users by the modification of scripts found in the RaaSI CLI. These scripts are written in bash to allow users to customize them in any manor that best fits their specific situation. The scripts should be seen as a guide to RaaSI rather than RaaSI itself. Through script modification, users can easily create more load balancers and stateful storage nodes, as well as change the type of stateful storage that secondary services use.

2.1 RaaSI Setup

The RaaSI CLI guides users through the installation of all components used in RaaSI; however, it limits the users to have only three load balancers and three storage nodes. The load balancer and storage node count is set to three by default because three nodes are the minimum amount needed to provide accurate voting between the nodes to make communication decisions. This number can be increased if the user has available nodes that they would like to use for the purpose of load balancing or storing the stateful data of secondary services.

In order to change the number of load balancers used in RaaSI, developers should navigate to the "LoadBalancerSetup/LoadBalancerInstallation.sh" script shown in figure 1. Once in the file, the developer can add as many load balancers to the installation script as they desire. Due to all of the load balancers using a virtual IP (VIP) to communicate between Master and Worker Nodes, this is the only script that needs modification.

Figure 1: RaaSI Load Balancer Installation Script

```
#!/bin/bash

echo "Beginning Load Balancer setup script..."

echo "Enter the ip address for LoadBalancer1: "
read -r LB1_ip

# validation
val=$(../Validation/checkValidation.sh "$LB1_ip" 1)
while [ "passed" != "$val" ];
do
        echo "Unexpected Response: expected IP address"
        echo "Enter the ip address for LoadBalancer1: "
        read -r LB1_ip
        # validation
        val=$(../Validation/checkValidation.sh "$LB1_ip" 1)
done

echo "Enter the ip address for LoadBalancer2: "
read -r LB2_ip

# validation
val=$(../Validation/checkValidation.sh "$LB2_ip" 1)
while [ "passed" != "$val" ];
do
        echo "Unexpected Response: expected IP address"
        echo "Enter the ip address for LoadBalancer2: "
        read -r LB2_ip
        # validation
        val=$(../Validation/checkValidation.sh "$LB2_ip" 1)
done

echo "Enter the ip address for LoadBalancer3: "
read -r LB3_ip

# validation
val=$(../Validation/checkValidation.sh "$LB3_ip" 1)
while [ "passed" != "$val" ];
do
        echo "Unexpected Response: expected IP address"
        echo "Enter the ip address for LoadBalancer3: "
        read -r LB3_ip
        # validation
        val=$(../Validation/checkValidation.sh "$LB3_ip" 1)
done

```

It is a slightly more complex process if developers want to change to number of secondary service storage nodes due to the amount of scripts that need to be changed for this to work. The first script that developers will need to change to modify the count of storage nodes is "GlusterSetup/ GlusterInstallationMaster.sh" shown in figure 2. To add more storage nodes to this script, developers need to request more IP address and login credentials from the user of the script. Developers also must modify the Worker Node installation

Figure 2: RaaSI Storage Node Installation Script

```
#!/bin/bash

echo "Beginning GlusterFS setup script..."

echo "Enter the ip address for Storage1: "
read -r storage1_ip

# validation
val=$(../Validation/checkValidation.sh "$storage1_ip" 1)
while [ "passed" != "$val" ];
do
        echo "Unexpected Response: expected IP address"
        echo "Enter the ip address for Storage1: "
        read -r storage1_ip
        # validation
        val=$(../Validation/checkValidation.sh "$storage1_ip" 1)
done

echo "Enter the ip address for Storage2: "
read -r storage2_ip

# validation
val=$(../Validation/checkValidation.sh "$storage2_ip" 1)
while [ "passed" != "$val" ];
do
        echo "Unexpected Response: expected IP address"
        echo "Enter the ip address for Storage2: "
        read -r storage2_ip
        # validation
        val=$(../Validation/checkValidation.sh "$storage2_ip" 1)
done

echo "Enter the ip address for Storage3: "
read -r storage3_ip

# validation
val=$(../Validation/checkValidation.sh "$storage3_ip" 1)
while [ "passed" != "$val" ];
do
        echo "Unexpected Response: expected IP address"
        echo "Enter the ip address for Storage3: "
        read -r storage3_ip
        # validation
        val=$(../Validation/checkValidation.sh "$storage3_ip" 1)
done
```

script, found at "ClientNodeSetup/ glusterfsClientSetup.sh," to make sure that the Worker Nodes are connected to the newly added storage nodes. This script is shown in figure 3. Similar to the "GlusterInstallationMaster.sh" script, this script just needs to have the extra IP address and login information added to it to allow developers to increase the number of storage nodes.

2.2 Secondary Service Stateful Storage

For secondary service stateful storage RaaSI uses GlusterFS. The reason behind this is that GlusterFS is a lightweight, easy to integrate, file system that allows users to manage with speed and simplicity. However, this does not have to be the storage used for secondary services in RaaSI. Developers can disregard GlusterFs and use their own form of storage if they so choose.

Modification of the type of stateful storage for secondary services can be done by modifying the script "GlusterSetup/ GlusterInstallationMaster.sh," shown in figure 2, and replacing the installation of GlusterFS information with the installation process of the storage that the developer desires. If developers choose to change this storage, they will also need to modify the scripts "ClientNodeSetup/ glusterfsClientSetup.sh", shown in figure 3, and "ClientNodeSetup/ clientSideNodeSetup.sh", shown in figure 4. The script "clientSideNodeSetup.sh" is where

Figure 3: RaaSI Worker Node Storage Installation Script

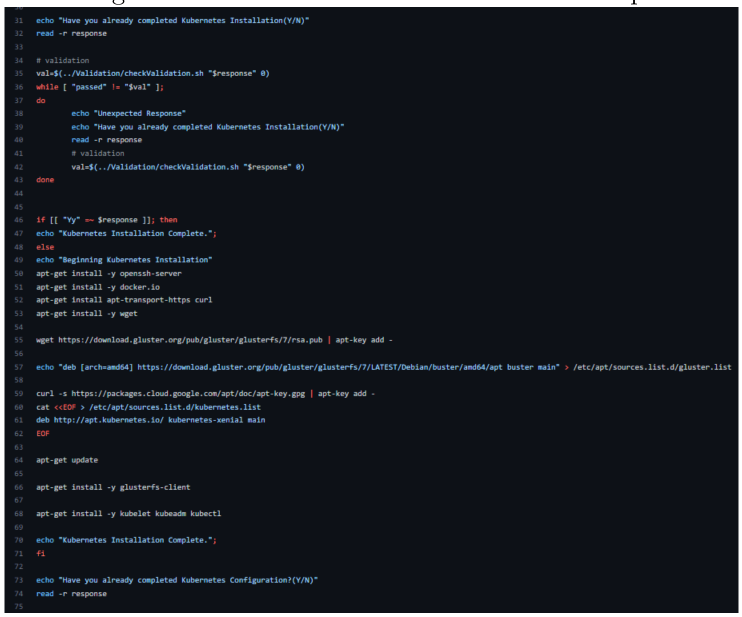
```bash
#!/bin/bash

client_ip=$1
client_user=$2

echo "Connecting the client node to GlusterFS..."

echo "Enter the IP address for Storage1:"
read -r storage1_ip

# validation
val=$(../Validation/checkValidation.sh "$storage1_ip" 1)
while [ "passed" != "$val" ];
do
	echo "Unexpected Response: expected IP address"
	echo "Enter the IP address for Storage1: "
	read -r storage1_ip
	# validation
	val=$(../Validation/checkValidation.sh "$storage1_ip" 1)
done

echo "Enter username to connect to on Storage1: "
read -r storage_user1

# validation
val=$(../Validation/checkValidation.sh "$storage_user1" 2)
while [ "passed" != "$val" ];
do
	echo "Unexpected Response: valid username was expected"
	echo "Enter username to connect to on Storage1: "
	read -r storage_user1
	# validation
	val=$(../Validation/checkValidation.sh "$storage_user1" 2)
done

echo "Enter the IP address for Storage2:"
read -r storage2_ip

# validation
val=$(../Validation/checkValidation.sh "$storage2_ip" 1)
while [ "passed" != "$val" ];
do
	echo "Unexpected Response: expected IP address"
	echo "Enter the IP address for Storage2: "
	read -r storage2_ip
	# validation
	val=$(../Validation/checkValidation.sh "$storage2_ip" 1)
done

echo "Enter username to connect to on Storage2: "
read -r storage_user2
```

GlusterFS is first installed to the Worker Nodes, this script must be modified to install the new storage to the Worker Nodes instead. The script "glusterfsClientSetup.sh" is where GlusterFs is configured on the Worker Nodes. This is where developers should place the configuration steps for the storage that they choose to use.

3 Configuration

Once RaaSI has been installed, developers can configure the environment to better fit their use. As discussed in the Users Guide, the Primary Service Monitoring API of RaaSI can be configured to monitor the health of any primary service through modifying the python script for monitoring services. RaaSI can also be configured through changing the dashboard and overlay network that is setup in the kubernetes configuration. Finally, secondary services can be managed from their creation and throughout their deployment with the use of the stateful storage system, kubernetes node communication, or by creating secondary services that support other forms of communication.

Figure 4: RaaSI Worker Node Installation Script



```
echo "Have you already completed Kubernetes Installation(Y/N)"
read -r response

# validation
val=$(../Validation/checkValidation.sh "$response" 0)
while [ "passed" != "$val" ];
do
        echo "Unexpected Response"
        echo "Have you already completed Kubernetes Installation(Y/N)"
        read -r response
        # validation
        val=$(../Validation/checkValidation.sh "$response" 0)
done


if [[ "Yy" =~ $response ]]; then
echo "Kubernetes Installation Complete.";
else
echo "Beginning Kubernetes Installation"
apt-get install -y openssh-server
apt-get install -y docker.io
apt-get install apt-transport-https curl
apt-get install -y wget

wget https://download.gluster.org/pub/gluster/glusterfs/7/rsa.pub | apt-key add -

echo "deb [arch=amd64] https://download.gluster.org/pub/gluster/glusterfs/7/LATEST/Debian/buster/amd64/apt buster main" > /etc/apt/sources.list.d/gluster.list

curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
cat <<EOF > /etc/apt/sources.list.d/kubernetes.list
deb http://apt.kubernetes.io/ kubernetes-xenial main
EOF

apt-get update

apt-get install -y glusterfs-client

apt-get install -y kubelet kubeadm kubectl

echo "Kubernetes Installation Complete.";
fi

echo "Have you already completed Kubernetes Configuration?(Y/N)"
read -r response

```

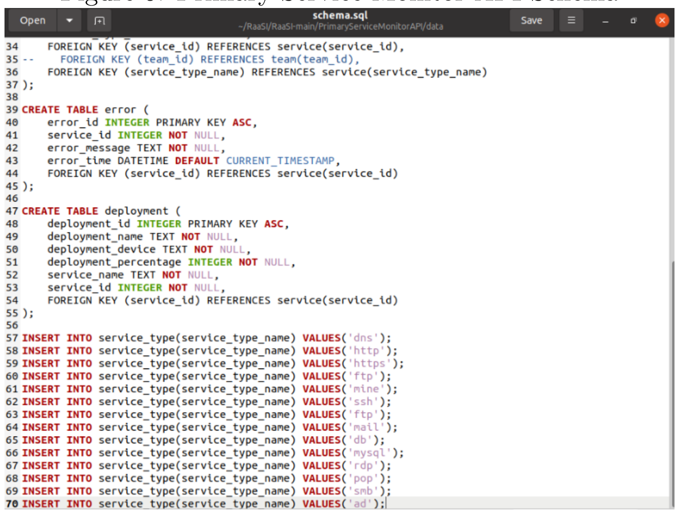
3.1 Primary Service Monitoring API

To monitor a new primary service with the primary service monitoring API, developers 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 developer should check the "PrimaryServiceMonitoringAPI/ data/ schema.sql" file to see if their service name is already in the database. If it is not, the developer 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 5.

After the new service has been added to the database, the developer should add a health monitoring method for the service. This can be done in the "PrimaryServiceMonitoringAPI/ judge/ tasks.py" file. In this file the developer 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 6. Once this method has been created, the developer 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 7.

Finally, the developer 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. In this file, the variable "service_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

Figure 5: Primary Service Monitor API Schema



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.

3.2 Kubernetes Configuration

The dashboard that is currently used for the default RaaSI installation [1] is a default dashboard used to display basic kubernetes metrics. This dashboard, shows the state of the kubernetes cluster; including node health, deployment status, cluster errors, and node resource limitations. However, if developers would prefer to use a different dashboard to show the metrics of the system they need to modify the script "MasterNodeSetup/ masterSetup.sh" shown in figure 8. To use a different dashboard, developers can swap out the current dashboard yaml location with their own custom one.

Developers can also change the type of overlay network that kubernetes uses for node communication. Currently, RaaSI uses the tigra calico overlay network, which integrates nicely with kubernetes. It is simple to change this overlay network, the developer just needs to change the source of the yaml file for the overlay network shown in the "MasterNodeSetup/ masterSetup.sh" script in figure 8. Once this change is complete, the developer can run the "masterSetup.sh" script and the new overlay network should be setup.

Figure 6: Primary Service Monitory API Health Monitor 2


```python


@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
```

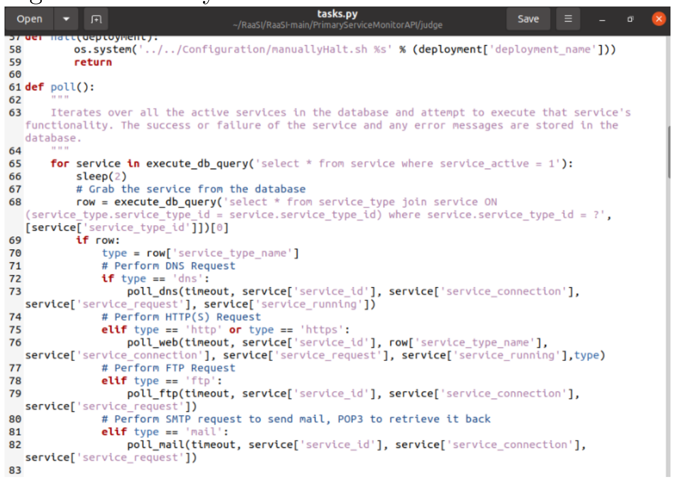
3.3 Developing Secondary Services

Secondary services can be deployed through the script presented in the RaaSI CLI; however, this deployment is only the beginning of managing the secondary services. Developers can use the stateful storage system to perform communication between secondary service instances. This can be done by passing data through the mounted drive on the secondary service. Developers can also send messages directly to secondary services through the "kubectl exec" command. This command allows developers to send commands to nodes belonging to a specific deployment or to individual nodes (the IP address of all nodes can be shown through viewing the dashboard). These forms of communication allow developers to manipulate their secondary services to best fit their scenarios.

4 Conclusion

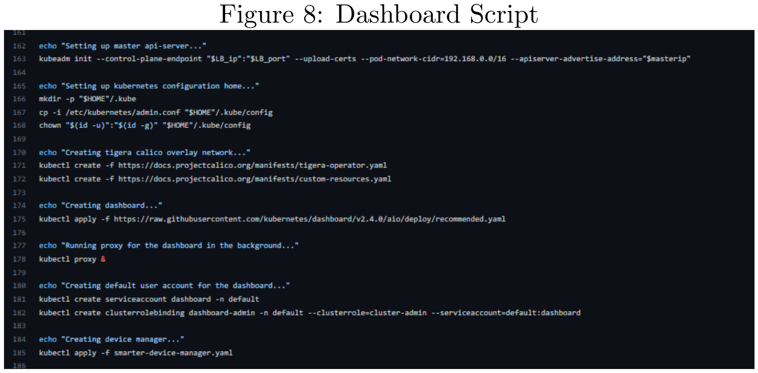
This concludes the RaaSI Developers Guide. RaaSI can be found at the following git repository [2]. For more generic information regarding the installation and configuration of RaaSI, users should read the RaaSI Users Guide. There is also a technical report that has been created demonstrating the need for RaaSI, works that are related to RaaSI, and discusses tests run on RaaSI. That report is titled RaaSI Technical Report.

Figure 7: Primary Service Monitor API Heath Monitor 1



```python
        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'])

```

Figure 8: Dashboard Script



```bash

echo "Setting up master api-server..."
kubeadm init --control-plane-endpoint "$LB_ip":"$LB_port" --upload-certs --pod-network-cidr=192.168.0.0/16 --apiserver-advertise-address="$masterip"

echo "Setting up kubernetes configuration home..."
mkdir -p "$HOME"/.kube
cp -i /etc/kubernetes/admin.conf "$HOME"/.kube/config
chown "$(id -u)":"$(id -g)" "$HOME"/.kube/config

echo "Creating tigera calico overlay network..."
kubectl create -f https://docs.projectcalico.org/manifests/tigera-operator.yaml
kubectl create -f https://docs.projectcalico.org/manifests/custom-resources.yaml

echo "Creating dashboard..."
kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.4.0/aio/deploy/recommended.yaml

echo "Running proxy for the dashboard in the background..."
kubectl proxy &

echo "Creating default user account for the dashboard..."
kubectl create serviceaccount dashboard -n default
kubectl create clusterrolebinding dashboard-admin -n default --clusterrole=cluster-admin --serviceaccount=default:dashboard

echo "Creating device manager..."
kubectl apply -f smarter-device-manager.yaml

```

References

[1] The Kubernetes Authors. *Dashboard*. Version 2.4.0. Mar. 2022. URL: `https://raw.githubusercontent.com/kubernetes/dashboard/v2.4.0/aio/deploy/recommended.yaml`.

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