

# Runlet: A Cross-platform IoT Tool for Interactive Job Execution Over Heterogeneous Devices with Reliable Message Delivery

Vandré Leal Cândido<sup>a</sup> and Flávio de Oliveira Silva<sup>b</sup>

*Faculty of Computing, Federal University of Uberlândia, Brazil*

**Keywords:** Runlet, Internet of Things, AMQP, RabbitMQ, Interactivity, Reliability.

**Abstract:** IoT uses different hardware and software components in a mixed environment, and for the management of the devices, interoperability and reliability are key issues. Interactive job execution is another important concept for the management in different scenarios. The literature lacks a tool with such characteristics. This work fills the gap in the state-of-the-art by introducing a tool that achieves interactive job execution over a network of heterogeneous devices with reliable message delivery. The tool leverages the power of the protocol Advanced Message Queuing Protocol (AMQP) and the message broker RabbitMQ. AMQP is an open standard Machine-to-Machine (M2M) publish/subscribe messaging protocol optimized for high-latency and unreliable networks that enables client applications to communicate with conforming messaging middleware brokers. RabbitMQ is an open-source message broker that supports various message protocols. The architecture of Runlet is discussed in detail, including the reasoning behind architectural decisions. The evaluation is conducted through an experimental approach that assesses interactivity and reliability on a testbed of devices composed of single-board ARM computers and laptop devices. The experimental results show that the application offers interactivity under different scenarios and provides reliable message delivery after node and server failures.

## 1 INTRODUCTION

The heterogeneous and dynamic nature of the Internet of Things (IoT) creates challenges that go beyond the traditional computer-based network model. These challenges are commonly related to the unpredictable mixture of devices with individual capabilities that may pose a barrier to achieving interoperability and managing devices in the context of IoT (Elkhodr et al., 2016).


The IEEE describes interoperability as “the ability of two or more systems or components to exchange information and to use the information that has been exchanged” (Geraci et al., 1991). Achieving interoperability is indispensable for devices across different networks and a difficult task given the IoT’s competitive nature and rapidly evolving wireless technologies. This commonly results in integration issues where heterogeneous devices cannot communicate with one another (Elkhodr et al., 2016).


In addition to that, IoT solutions need to handle unique management scenarios that go beyond the tra-

ditional capabilities of remote control, monitoring, and maintenance of devices. For example, a system that supports remote monitoring of tasks across a fleet of heterogeneous devices may highly benefit from a tool that offers the ability to monitor and troubleshooting errors in real-time.

This work fills the gap in the state of the art by presenting a tool that achieves interactive job execution over a network of heterogeneous devices with reliable message delivery. The tool, named Runlet, grants the user the capability of interacting with the shell from any connected device during job execution, which is particularly useful in a scenario that requires user input for successful script execution. A few potential use areas include Continuous Integration (CI), Continuous Delivery (CD), and remote monitoring and management of devices.

Runlet is a cross-platform application that runs across many architectures and operating systems. It uses both the protocol AMQP and the broker RabbitMQ for reliable message delivery. The protocol AMQP is an open standard M2M publish/subscribe messaging protocol optimized for high-latency and unreliable networks that enables client applications to

<sup>a</sup>  <https://orcid.org/0000-0001-7737-8878>

<sup>b</sup>  <https://orcid.org/0000-0001-7051-7396>

communicate with middleware brokers. RabbitMQ is an open-source lightweight message broker that supports various messaging protocols and can be deployed on-premises and in the cloud.

An experimental evaluation assesses interactivity and reliability on a testbed of heterogeneous devices ranging from single-board ARM computers to laptop devices. The results show that Runlet is interactive under different use cases and that message delivery is reliable after node and server failures.

This paper is organized as follows. Section 2 evaluates the related work. Section 3 describes the application and the reasoning behind architectural decisions. Section 4 describes the experimental evaluation used to evaluate interactivity and reliability. Finally, Section 5 presents our conclusions.

## 2 RELATED WORK

There are several studies that evaluate messaging protocols and distributed message brokers. However, to the best of our knowledge, there is no other academic study pursuing the goal of performing interactive job execution across a fleet of network devices with reliable message delivery.

A selection of academic studies is presented next, ranging from monitoring systems to surveys that evaluate messaging protocols and message brokers. However, no mention is made of interactive job execution in none of the studies.

(Krishna and Sasikala, 2019) introduces a healthcare monitoring system that monitors patients' vitals in real-time and sends the information to a web server for persistence and decision-making. Sensors are connected to a Raspberry Pi that acts as a gateway and sends the data to a server using AMQP through the board's Wi-Fi.

(Liang and Chen, 2018) introduces the design of a real-time data acquisition and monitoring system for medical data on smart campuses. The design includes both AMQP and RabbitMQ for data exchanging and the HL7 protocol to facilitate information management via standard-compliant messages. The HL7 standard applies to the exchange of medical records before different medical systems, which is also the main difference between this study and (Krishna and Sasikala, 2019) given that both introduce a similar concept for healthcare monitoring.

(Kostromina et al., 2018) presents a concept for a resilient meteorological monitoring system using AMQP. The system design considers the need for stable and resilient delivery of data without losses, even in case of network outages and a predictive monitor-

ing system that monitors memory usage on single-board computers to prevent system crashes caused by lack of available memory. RabbitMQ is the selected message broker since it implements the protocol AMQP and has features like federation, clustering, and persistence.

(Krishna and Sasikala, 2019) is the only study that presents the prototype's screen captures alongside this study. Interactive job execution is not applicable in none of the studies apart from Runlet, which is further motivation for conducting this study.

## 3 RUNLET

This study aims to fill the lack of a tool for interactive job execution by introducing an interactive job execution approach with reliable message delivery across heterogeneous devices. The queuing protocol and the message broker are two key elements discussed in this section, which also details the architecture and reasoning behind technology choices.

Runlet is a cross-platform application that features a daemon and a desktop manager. The desktop manager includes both the daemon and a full-featured GUI that provides an easy to use interface for managing jobs across a fleet of connected devices. The Advanced RISC Machine (ARM) distribution supports ARMv6, ARMv7, and ARM64.

Next, we present the reasoning and choices for the queuing protocol and the message broker. In the sequence, we present Runlet's architecture.

### 3.1 The Queuing Protocol

CoAP and XMPP protocols are discarded due to some of their characteristics described in the previous chapter. XMPP does not provide any Quality of Service (QoS) options (Dhas and Jeyanthi, 2019), which makes the protocol not suitable for Runlet, ensuring reliable message delivery is a crucial aspect of the application. CoAP seems like a great option from a resource usage and network bandwidth standpoint. However, it is discarded due to its lack of support for the publish/subscribe model (Karagiannis et al., 2015). The publish/subscribe model is a must-have for reaching optimal performance as job logs are updated continuously.

MQTT is an excellent option and very similar to AMQP in many aspects. They both have brokered architectures, support the publish/subscribe model, use TCP for transport and TLS/SSL for security, offer QoS levels for reliability, with MQTT having the advantage of smaller header size (Dhas and Jeyanthi,

2019). However, extra features provided by AMQP made the trade-off advantageous despite the increased overhead and latency. AMQP has the advantage of having flexible message routing, reliable message queues with fine-grained control over bindings, and multiple types of exchanges (AMQP, 2020). After a detailed analysis of the different protocols, we selected the AMQP protocol.

### 3.2 The Message Broker

RabbitMQ is the selected broker for the following reasons (RabbitMQ, 2020):

- Web management UI for short-term metric collection and monitoring offers many administrative features convenient for development.
- Built-in support for using long-term metric visualization tools such as Prometheus and Grafana in production.
- Concise documentation and tutorials.
- Feature flag subsystem for upgradability.

ActiveMQ Artemis also comes with a web management console, supports metric tools, and has well-organized documentation and active community (ActiveMQ, 2020). Nonetheless, the decision ultimately came down to a personal preference for RabbitMQ’s management tool and built-in support to Prometheus and Grafana in production. Qpid Broker-J and Qpid C++ Broker, on the other hand, fall short on development tools and lack support for metrics (Qpid, 2020). The Qpid community is also not as active and engaged as both RabbitMQ and ActiveMQ communities.

### 3.3 Architecture

This section details the conceptual architecture composed of the following components: GUI, daemon, server, database, and cloud storage. Fig. 1 shows the conceptual architecture and communication flow between components.

#### 3.3.1 GUI

The Graphical User Interface (GUI) is a cross-platform application that runs on all major operating systems such as Mac, Linux, and Windows. It provides a friendly interface for the management and monitoring of devices and applies reactive programming concepts to observe, compute, and react to changes.

The GUI is built with JavaScript, HTML, and CSS on top of well-adopted open-source libraries such as Electron, React, TypeScript, MobX, Blueprint,

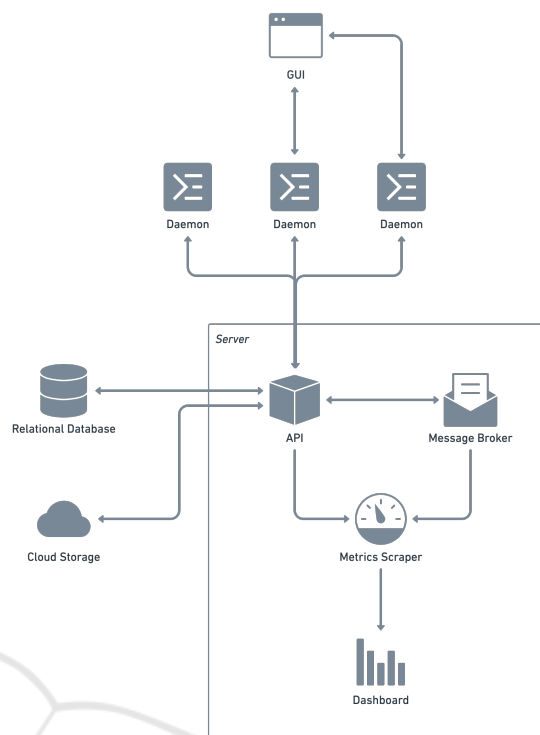


Figure 1: Runlet Conceptual Architecture.

Xterm.js, and Jest. Electron is a library developed by GitHub for building cross-platform desktop applications by combining Chromium and Node.js into a single runtime environment system (Electron, 2020). Chromium is used for rendering pages, while the Node.js API is used for lower-level system actions such as managing process events and interacting with the file system.

#### 3.3.2 Daemon

The daemon, written in Golang, is responsible for job orchestration and has an internal queue that executes jobs in the received order, 'First In, First Out' (FIFO), as well as a parallelism controller that helps to prevent CPU throttling by limiting the number of jobs that run in parallel. It runs on machine startup as a background process capable of recovering from crashes through a process restart.

Each daemon communicates with the server directly. It acts both as a producer and a consumer simultaneously, meaning that jobs are triggered and executed from any active node in the network. It supports the execution of selected methods via Command-Line Interface (CLI) by appending the term `runlet` to method names, which is useful for script automation and remote access on devices without a graphical interface.

The daemon is also responsible for End-to-End Encryption (E2EE) of logs, local disk persistence, and data dispatch for cloud storage. Logs are first encrypted and persisted on the local disk for quick access and then submitted to the cloud storage server. Logs are only retrieved from the cloud when the local copy is outdated, according to the rules:

1. **If the Log Does Not Exist Locally:** a local copy is created from the remote copy and used for reading operations.
2. **If Log Exists Locally:** the hash of the local copy is sent to the server and compared with the hash present in the metadata of the remote copy. The remote copy replaces the hash's local copy if it is different, and the remote copy has a more recent modification date. Otherwise, the local copy is used for reading operations, avoiding unnecessary updates.

### 3.3.3 Database

Runlet uses PostgreSQL, an open-source object-relational database that uses the Structured Query Language (SQL) language and is highly scalable both in the amount of data it can manage and in the number of concurrent users it accommodates (PostgreSQL, 2020). The database stores data from users and jobs. It also provides relational operators to manipulate data, is highly scalable in terms of data it can store and supports many concurrent users.

However, the database does not store job logs. They are shared between devices using RabbitMQ's messaging system. This decision reduces security concerns related to leakage of sensitive information that records may have and improves performance by ensuring that updates are queued and delivered reliably without the need for querying and storing large amounts of data from a database.

### 3.3.4 Cloud Storage

A cloud-based solution is used exclusively for log storing. This eliminates ownership costs associated with managing a data storage infrastructure and ensures data durability, availability, and security.

Log storing is accomplished with MinIO, an open-source distributed object storage server designed to be cloud-native, performant, scalable, and lightweight. The server runs as lightweight containers by external orchestration services and is highly efficient in its use of CPU and memory resources (MinIO, 2020).

### 3.3.5 Server

The server is the central point of the architecture and consists of a NestJS API, a RabbitMQ message broker, and metric tools Prometheus and Grafana for monitoring. It orchestrates the communication between daemons and the message broker through an API, persists data in the database, and files in the cloud.

RabbitMQ is the message broker that handles message exchanging between network devices. It has two subcomponents: (1) a server operator where all broker features are configurable, such as clustering, high availability, persistence, and access control; and (2) a web-based tool for external management, monitoring, and short-term visualization of broker events.

A *topic* exchange is created for each user. This type of exchange is used for multicast routing of messages as it routes messages to one or many queues bound with a matching binding key. All exchanges are durable and survive broker restart. On the other hand, queues are temporary and short-lived to reduce workload and avoid leaving unused queues behind failed connections. They are both *exclusive*, and *auto-deleted*, meaning that they are only used by their declaring connections and are automatically deleted when the last consumer is gone or cancels its subscription.

Queues are also created following the single responsibility principle that dictates that each queue has a single concern. That means that jobs end up having multiple queues to control execution rather than having a single queue for everything.

## 4 EXPERIMENTAL EVALUATION

This section describes the experiments that evaluate Runlet in regards to interactivity and reliability. The server, currently hosted on DigitalOcean, is a standard droplet with two (2) vCPUs, four (4) GB RAM, fifty (50) GB SSD storage, and four (4) TB of data transfer. A vCPU is a processing power unit that corresponds to a single hyperthread on a processor core (DigitalOcean, 2020). The testbed of devices used for the experiments is presented in table 1, including device model, CPU, RAM, operating system, and a nickname used for study reference.

### 4.1 Interactivity

Two (2) experiments are conducted to evaluate the user's capability to interact with job executions and make decisions when requested.





```

21/46 keybase 5.2.0-20200130011203,cf82db8320 5.2.0-20200130011203,c
22/46 launchcontrol 1.50 1.50
23/46 libreoffice 6.4.0 6.4.0
24/46 logitech-options 8.10.64 8.10.64
25/46 macs-fan-control 1.5.4 1.5.4
26/46 magicprefs 2.4.3 2.4.3
27/46 mounty latest latest
28/46 nordvpn 5.3.6 5.4.2
29/46 openemu 2.2.1 2.2.1
30/46 plex-media-server 1.18.6.2368-97add474d 1.18.7.2457-77cb9455c
31/46 sequel-pro 1.1.2 1.1.2
32/46 sip 2.2.5 2.2.5
33/46 skype 8.56.0.106 8.56.0.106
34/46 slack 4.3.3 4.3.3
35/46 spectacle 1.2 1.2
36/46 spotify latest latest
37/46 the-unarchiver 4.1.0.121:1549634528 4.1.0.121:1549634528
38/46 transmission 2.94 2.94
39/46 tunnelblick 3.8.1,5400 3.8.1,5400
40/46 vagrant 2.2.7 2.2.7
41/46 virtualbox 6.0.14,133895 6.1.4,136177
42/46 virtualbox-extension-pack 6.0.14 6.1.4
43/46 visual-studio-code 1.42.1 1.42.1
44/46 vlc 3.0.8 3.0.8
45/46 vnc-viewer 6.20.113 6.20.113
46/46 vox 3398.3,1580250435 3398.3,1580250435

==> Found outdated apps
Cask Current Latest A/U Result
1/7 balenaetcher 1.5.76 1.5.79 [OUTDATED]
2/7 boostnote 0.14.0 0.15.0 Y [ FORCED ]
3/7 brave-browser 80.1.3.115,103.115 80.1.3.118,103.118 Y [ FORCED ]
4/7 gitkraken 6.5.1 6.5.3 Y [ FORCED ]
5/7 google-chrome 80.0.3987.106 80.0.3987.122 Y [ FORCED ]
6/7 nordvpn 5.3.6 5.4.2 Y [ FORCED ]
7/7 plex-media-server 1.18.6.2368-97add474d 1.18.7.2457-77cb9455c Y [ FORCED ]

Do you want to upgrade 7 apps or enter [i]nteractive mode [y/i/N]? y
    
```

Figure 3: Output of (2) brew cu -a before confirmation [y].

```

Do you want to upgrade 7 apps or enter [i]nteractive mode [y/i/N]? y
==> Upgrading balenaetcher to 1.5.79
--> Downloading https://github.com/balena-io/etcher/releases/download/v1.5.79/ba
--> Downloading from https://github-production-release-asset-2e65be.s3.amazonaws
##### 100.0%
--> Verifying SHA-256 checksum for Cask 'balenaetcher'.
==> Installing Cask balenaetcher
Warning: It seems there is already an App at '/Applications/balenaEtcher.app'; overwriting.
--> Removing App '/Applications/balenaEtcher.app'.
--> Moving App 'balenaEtcher.app' to '/Applications/balenaEtcher.app'.
balenaetcher was successfully installed!
==> Upgrading boostnote to 0.15.0
--> Downloading https://github.com/BoostIO/boost-releases/releases/download/v0.1
--> Downloading from https://github-production-release-asset-2e65be.s3.amazonaws
##### 100.0%
--> Verifying SHA-256 checksum for Cask 'boostnote'.
==> Installing Cask boostnote
Warning: It seems there is already an App at '/Applications/Boostnote.app'; overwriting.
--> Removing App '/Applications/Boostnote.app'.
--> Moving App 'Boostnote.app' to '/Applications/Boostnote.app'.
boostnote was successfully installed!
==> Upgrading brave-browser to 80.1.3.118,103.118
--> Downloading https://updates-cdn.bravesoftware.com/sparkle/Brave-Browser/stab
##### 100.0%
--> Verifying SHA-256 checksum for Cask 'brave-browser'.
==> Installing Cask brave-browser
Warning: It seems there is already an App at '/Applications/Brave Browser.app'; overwriting.
--> Removing App '/Applications/Brave Browser.app'.
--> Moving App 'Brave Browser.app' to '/Applications/Brave Browser.app'.
brave-browser was successfully installed!
==> Upgrading gitkraken to 6.5.3
--> Downloading https://release.gitkraken.com/darwin/installGitKraken.dmg
--> Downloading from https://release.axocdn.com/darwin/installGitKraken.dmg
##### 100.0%
--> Verifying SHA-256 checksum for Cask 'gitkraken'.
==> Installing Cask gitkraken
    
```

Figure 4: Output of (2) brew cu -a after confirmation [y].

plications, including current and latest versions, and a list of outdated apps. The option -a passed with brew cu indicates that applications with the auto-update functionality are also listed. The question "Do you want to upgrade 7 apps or enter [i]nteractive mode [y/i/N]?" is shown after the list of outdated apps

and requires user interaction. Fig. 4 shows an excerpt of the upgrading process that takes place after user confirmation [y]. The progress of each upgrade is displayed individually, which helps to visualize the overall progress.

```

Avg[|] 1.6% Hostname: vlc-rpi
Mem[|] 358M/842M Tasks: 68, 189 thr; 1 running
Swp[|] 0K/0K Load average: 0.18 0.15 0.10
Uptime: 5 days, 00:15:12

PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
3806 root 20 0 5092 3472 2268 R 3.9 0.4 0:08.19 htop
404 root 20 0 922M 36312 13092 S 1.9 4.2 56:03.14 /opt/runlet/runle
459 root 20 0 922M 36312 13092 S 1.3 4.2 4:22.31 /opt/runlet/runle
440 root 20 0 922M 36312 13092 S 0.0 4.2 7:51.72 /opt/runlet/runle
473 root 20 0 922M 36312 13092 S 0.0 4.2 4:00.71 /opt/runlet/runle
467 root 20 0 922M 36312 13092 S 0.0 4.2 3:51.46 /opt/runlet/runle
6866 root 20 0 922M 36312 13092 S 0.0 4.2 4:01.81 /opt/runlet/runle
468 root 20 0 922M 36312 13092 S 0.0 4.2 4:14.52 /opt/runlet/runle
442 root 20 0 922M 36312 13092 S 0.0 4.2 0:15.37 /opt/runlet/runle
2229 root 20 0 922M 36312 13092 S 0.0 4.2 4:32.88 /opt/runlet/runle
458 root 20 0 907M 7748 3872 S 0.0 0.9 0:30.93 /usr/sbin/express
731 pi 20 0 162M 46336 16688 S 0.0 5.4 28:29.87 lxpanel --profile
1169 root 20 0 922M 36312 13092 S 0.0 4.2 4:20.73 /opt/runlet/runle
555 root 39 19 54116 9736 2668 S 0.0 1.1 6h19:27 /usr/bin/perl /us
1 root 20 0 28144 5168 3920 S 0.0 0.6 53:41.86 lib/systemd/syste
444 root 20 0 922M 36312 13092 S 0.0 4.2 4:40.95 /opt/runlet/runle
477 root 20 0 176M 61432 22212 S 0.0 7.1 6:17.43 /usr/lib/xorg/Xor
472 root 20 0 922M 36312 13092 S 0.0 4.2 4:10.85 /opt/runlet/runle
24074 pi 20 0 643M 102M 62424 S 0.0 12.1 1:15.97 /usr/lib/chromium
24343 pi 20 0 391M 46332 27092 S 0.0 5.4 0:10.33 /usr/lib/chromium
320 root 20 0 907M 7748 3872 S 0.0 0.9 5:12.25 /usr/sbin/express
24126 pi 20 0 643M 102M 62424 S 0.0 12.1 0:29.83 /usr/lib/chromium
Filehelp F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice F8Nice F9Kill F10Quit
    
```

Figure 5: Initial output of htop.

```

Avg[|] 1.3% Hostname: vlc-rpi
Mem[|] 358M/842M Tasks: 68, 189 thr; 1 running
Swp[|] 0K/0K Load average: 0.21 0.16 0.11
Uptime: 5 days, 00:15:29

Setup Left column Right column Available meters
Meters CPU [Bar] Hostname [Text] Clock
Display options Memory [Bar] Task counter [Text] Load averages: 1 minu
Colors Swap [Bar] Load average [Text] Load: average of read
Columns Uptime [Text] Memory
Swap
Task counter
Uptime
Battery
Hostname
CPUs (1/1): all CPUs
CPUs (162/2): all CPU
CPUs (1/2): first ha
CPUs (2/2): second ha
CPUs (162/4): first h
CPUs (364/4): second
Blank
CPU average
CPU 1
CPU 2
CPU 3
CPU 4

F1 Done
    
```

Figure 6: htop setup results for keyword 'runlet'.

The second experiment triggers a job on a Windows desktop manager to display running processes of a Raspberry PI using htop. Fig. 5 shows the initial output of htop, which is a list of system processes includ-

ing PID, user, percentage of CPU and memory used, virtual memory, and time in execution. The bottom bar has all the options that are available through function keys.

Fig. 6 shows all the options available under setup when the key F2 is pressed. These options can customize meters shown at the top and change display options, colors, and active columns. Navigation using arrow keys is done with no hassle as Runlet can capture all keyboard strokes. The key F10 can be pressed at any time to indicate that the setup is done and return to the previous screen.

### 4.3.2 Reliability

Reliability is investigated by observing the broker’s behavior considering two failure events. The impact is observed through a Grafana dashboard that monitors the following metrics using Prometheus:

- Total number of nodes, queues, connections, and channels.
- Per-node memory available.
- Messages published, delivered, and routed to queues in a per-node and per-second granularity.
- Total number of queues, channels, and connections per node per second.

The first experiment investigates how the broker behaves to node failure by changing the cluster composition from four (4) nodes to a single node at about 16:15. Figure 7 shows that all messages, queues, channels, and connections are kept in the remaining node after removing three (3) nodes, meaning no messages are lost.

Grafana may use different colors to indicate the same node across graph panels. For example, green is designated for node one (1) on messages published, delivered, and routed to queues, while yellow is designated to the same node for total queues, channels, and connections.

A slight increase in memory consumption happens on the remaining node as it starts to handle all the load. The difference between the number of messages published and routed to queues for delivery may be attributed to messages routed to multiple queues. The number of queues, channels, and connections varies according to the workload of messages.

The second experiment investigates how the broker reacts to a server fail by removing the only active node at about 16:25. Figure 8 shows that messages stop being transmitted for about 5 minutes during the fail, but the process resumes as soon as the node joins the cluster again to take over. The node establishes all the old connections and recovers persisted messages



Figure 7: Cluster after node failure.

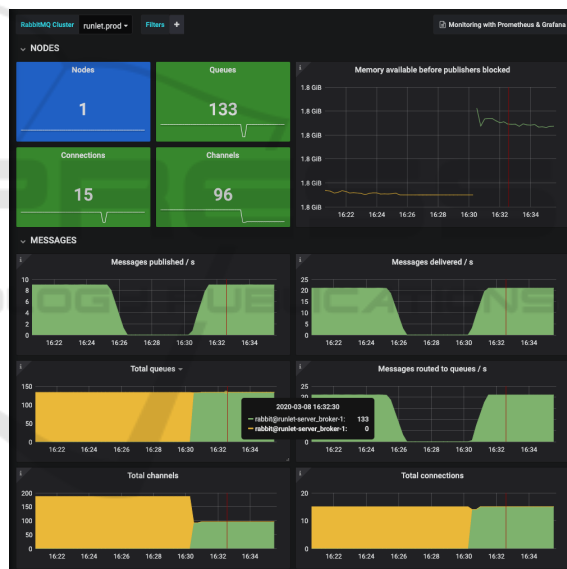


Figure 8: Cluster after server failure.

and queues, resulting in no lost messages. The number of channels drops significantly as previous connections and channels are closed. Grafana changes the color from yellow to green to differentiate instances of the same node.

## 5 CONCLUSIONS

This study introduces a tool that achieves interactive job execution across heterogeneous devices using the protocol AMQP and the message broker RabbitMQ.

The approach fills a gap in the state-of-the-art, as confirmed by the findings in section 2. A few potential use areas include Continuous Integration (CI), Continuous Delivery (CD), and remote monitoring and management of devices.

The protocol AMQP was selected due to its flexible message routing, reliable message queues, and multiple exchange types. The message broker RabbitMQ was selected based on a personal preference for the built-in web management tool and support for Prometheus and Grafana. This combination proved very useful in the experimental evaluation as the broker could recover messages and queues after node and server failures with no messages lost.

The architecture and motivation behind architectural decisions are discussed in detail in section 3, including a conceptual view that introduces all the components. The experimental evaluation is conducted in section 4. It shows that interactive job execution on heterogeneous devices is achieved under various scenarios and that message delivery is reliable even after node and server failures.

The application has been released on GitHub and is easily found over the internet. The website with more information is available at <https://runlet.app>.

The roadmap for future work includes an experimental evaluation to benchmark scalability by measuring message throughput and latency as the network grows. Also, the analysis of the security measures that are put in place to ensure data protection from cyberattacks.

## REFERENCES

- ActiveMQ, A. (2020). Apache activemq. <https://activemq.apache.org>. [Online; accessed 15-December-2020].
- AMQP (2020). Advanced message queuing protocol. <https://www.amqp.org>. [Online; accessed 15-December-2020].
- buo (2020). homebrew-cask-upgrade. <https://github.com/buo/homebrew-cask-upgrade>. [Online; accessed 15-December-2020].
- Dhas, Y. J. and Jeyanthi, P. (2019). A review on internet of things protocol and service oriented middleware. In *2019 International Conference on Communication and Signal Processing (ICCSP)*, pages 0104–0108. IEEE.
- DigitalOcean (2020). Droplets. <https://www.digitalocean.com/products/droplets>. [Online; accessed 15-December-2020].
- dylanaraps (2020). neofetch. <https://github.com/dylanaraps/neofetch>. [Online; accessed 15-December-2020].
- Electron (2020). Electron. <https://electronjs.org>. [Online; accessed 15-December-2020].
- Elkhodr, M., Shahrestani, S., and Cheung, H. (2016). The internet of things: new interoperability, management and security challenges. *arXiv preprint arXiv:1604.04824*.
- Geraci, A., Katki, F., McMonegal, L., Meyer, B., Lane, J., Wilson, P., Radatz, J., Yee, M., Porteous, H., and Springsteel, F. (1991). *IEEE standard computer dictionary: Compilation of IEEE standard computer glossaries*. IEEE Press.
- Homebrew (2020). homebrew-cask. <https://github.com/Homebrew/homebrew-cask>. [Online; accessed 15-December-2020].
- Karagiannis, V., Chatzimisios, P., Vazquez-Gallego, F., and Alonso-Zarate, J. (2015). A survey on application layer protocols for the internet of things. *Transaction on IoT and Cloud computing*, 3(1):11–17.
- Kostromina, A., Siemens, E., and Yurii, B. (2018). A concept for a high-reliability meteorological monitoring system using amqp. In *Titel: Proceedings of the 6th International Conference on Applied Innovations in IT*. Bibliothek, Hochschule Anhalt.
- Krishna, C. and Sasikala, T. (2019). Healthcare monitoring system based on iot using amqp protocol. In *International Conference on Computer Networks and Communication Technologies*, pages 305–319. Springer.
- Liang, Y. and Chen, Z. (2018). Intelligent and real-time data acquisition for medical monitoring in smart campus. *IEEE Access*, 6:74836–74846.
- MinIO (2020). Minio. <https://min.io>. [Online; accessed 15-December-2020].
- Muhammad, H. (2020). htop. <https://hisham.hm/htop>. [Online; accessed 15-December-2020].
- PostgreSQL (2020). PostgreSQL. <https://www.postgresql.org>. [Online; accessed 15-December-2020].
- Qpid, A. (2020). Apache qpid. <https://qpid.apache.org>. [Online; accessed 15-December-2020].
- RabbitMQ (2020). Rabbitmq. <https://www.rabbitmq.com>. [Online; accessed 15-December-2020].