CHALLENGES IN RESEARCH EXPERIMENTATION WITH OPEN-SOURCE PLATFORMS FOR SERVERLESS COMPUTING

An Undergraduate Research Scholars Thesis

by

KALISTA K. BAILEY

Submitted to the LAUNCH: Undergraduate Research office at
Texas A&M University
in partial fulfillment of requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Faculty Research Advisor: Dr. Dilma Da Silva

May 2022

Major: Computer Engineering – Computer Science Track

RESEARCH COMPLIANCE CERTIFICATION

Research activities involving the use of human subjects, vertebrate animals, and/or biohazards must be reviewed and approved by the appropriate Texas A&M University regulatory research committee (i.e., IRB, IACUC, IBC) before the activity can commence. This requirement applies to activities conducted at Texas A&M and to activities conducted at non-Texas A&M facilities or institutions. In both cases, students are responsible for working with the relevant Texas A&M research compliance program to ensure and document that all Texas A&M compliance obligations are met before the study begins.

I, Kalista K. Bailey, certify that all research compliance requirements related to this Undergraduate Research Scholars thesis have been addressed with my Research Faculty Advisor prior to the collection of any data used in this final thesis submission.

This project did not require approval from the Texas A&M University Research Compliance & Biosafety office.

TABLE OF CONTENTS

ABSTRACT

Challenges in Research Experimentation with Open-source Platforms for Serverless Computing

Kalista K. Bailey
Department of Computer Science & Engineering
Texas A&M University

Research Faculty Advisor: Dr. Dilma Da Silva
Department of Computer Science & Engineering
Texas A&M University

Cloud computing technology has enabled enterprises in all societal domains to achieve remarkable growth by harnessing data and computation power without an upfront investment in computer infrastructure or any expense in maintaining hardware. In the Infrastructure-as-a-Service model, cloud customers acquire virtual servers, i.e., they gain remote access to the operating system of their choice. Free of any concerns or responsibilities regarding the hardware, customers can focus more on the business and services offered by their applications. However, there is still the need to manage their software stacks, which is often quite complex. Serverless computing is a new cloud paradigm that aims to free software developers from the burden of maintaining their software infrastructure, i.e., the operating system, libraries, and runtime execution environments supporting their applications.

The serverless computing paradigm has gained traction and, for many, it represents the future of cloud computing. Academic researchers, industry leaders, and software developers are exploring the opportunities and challenges introduced by serverless computing. This new model presents challenges for all stakeholders. For the service providers leading this space currently

(namely, Amazon, Microsoft, Google, and IBM), the new technology requires changes in how computing resources (CPU, memory, network, and storage) are managed in order to achieve the targeted efficiency and economy-of-scale.

Academic researchers have been innovating in resource management for decades, having succeeded in improving the efficiency of supercomputers, mobile devices, private enterprise servers, and cloud computing providers. Industry researchers can experiment with new ideas in the context of a company's implementation of serverless computing, having access to the company’s intellectual property. Academic researchers need to rely on open-source implementations of serverless computing infrastructure, often with their progress hindered by deficiencies and limitations in the publicly available alternatives. Even for experienced system software researchers, the initial learning curve is quite steep.

In this work, we aim at making the experimentation with serverless computing frameworks practical for researchers with expertise in computer science but lacking experience in distributed systems and container technology. We assume the level of proficiency of a senior undergraduate or an early graduate student. Our research experiments with a popular open-source framework, OpenFaaS, to identify possible pitfalls in its deployment and find the components involved in its resource management strategy. The results of our exploration will simplify the path for introducing research management optimizations.

DEDICATION

This research is dedicated to Annabel Perry, a brilliant scientist and a dear friend.

ACKNOWLEDGEMENTS

Contributors

I would like to thank my faculty advisor, Dr. Dilma Da Silva for her guidance and support throughout the course of this research.

Thanks also go to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University a great experience.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

This work was made possible in part by the National Science Foundation under Grant Number NSF-SGX- 1919181. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the National Science Foundation.

NOMENCLATURE

CLI	Command Line Interface
CPU	Central Processing Unit
GB	Gigabyte
IT	Information Technology
RAM	Random-access Memory
Serverful	The traditional first phase of cloud computing that focuses on servers
vCPU	Virtual CPU
VM	Virtual Machine

1. INTRODUCTION

Cloud computing had a tremendous impact on the computing industry and radically changed how companies acquire computing resources [1]-[4]. With cloud computing, the physical infrastructure (computers, networks, storage units, powering and cooling systems) is maintained by cloud service providers in hyper-scale datacenters. A new paradigm in cloud computing, serverless computing, goes beyond hardware management. Serverless computing aims to manage virtually all system operations, including software updates, making it easier for the programmer to use cloud services [5], [6].

Serverless introduces a radical change in how developers can request cloud resources. The existing cloud resource management infrastructure, built around virtual machines and containers, needs to change in order to deliver services with efficiency on two fronts: low-cost for the provider and efficient performance (i.e., latency, throughput, and energy consumption) for the users [7]-[9]. Academic researchers have been proposing several approaches for service optimization or enhancement (for example, [10], [11], [12]). A thorough experimental evaluation of any proposed resource management algorithms is required in order to demonstrate benefits. Academic researchers rely on open-source implementations of the serverless paradigm to conduct their experiments.

This chapter introduces serverless computing and describes existing open-source implementations of serverless platforms.

1.1 Serverless Computing

In contrast with its name, serverless computing does in fact utilize servers to operate. Servers act as the invisible foundation for serverless computing, hidden from the application builder but managed by the cloud provider. Considered by many to be the next step in cloud computing, the serverless approach aims to shift the operational responsibilities to the cloud provider so as to allow the programmer to focus on their specific application [13].

1.1.1 Serverless vs. "Serverful" Computing

There are some distinct differences between serverless and "serverful" computing. Throughout the initial development and expansion of cloud computing, serverful computing has been the standard. "Serverful" computing may be viewed as the first phase of cloud computing, essentially operating as a service for businesses and developers to rent remote servers for specific blocks of time (x number of servers for y number of hours). Due to the fact that costs for these servers are determined based on the number of servers and how long these servers are utilized by the developer, details such as what percentage of that time are resources actively being used are not accounted for. During the dawn of cloud computing, serverful computing became common practice, with a majority of enterprise IT spending for software and infrastructure based in the cloud by May of 2021 [13].

Serverless computing promises to be the next stage of cloud computing, offering improvements to methods employed under the serverful framework. Serverless computing seeks to change the nature of cloud computing, specifically the relationship between cloud providers and programmers. Serverless seeks to remove the responsibility of server operations from the end-user, as servers often have complex needs with regards to maintenance and security. Instead of the reservation model provided by serverful computing, serverless computing establishes a pay-as-you-go model, with programmers only paying for resources that are actively in use, rather than for an allotted amount of time. This more cost-efficient method allows for better management of resources by both the provider and the developer.

An example that best describes the difference between serverless and serverful computing is the contrast between a car rental service and a taxi service [13]. If a customer rents a car, they are paying for the entire rental period, regardless of what is occurring during that period. The customer is responsible for all maintenance the car requires during that time and continues to pay for the use of the car even when it is parked. In contrast, when a person hails a cab, they are only paying for the time they are actively being taken to their destination. The customer is in no way responsible for the taxi cab's maintenance as this is a responsibility handled by the company that owns the taxi. The moment this customer exits the vehicle and is no longer actively using the taxi service, they are under no obligation to continue paying for the cab and the resource is free for new customers to utilize. This simple analogy illustrates how the responsibility of server upkeep is shifted from the programmer (the customer either renting the car or hailing the taxi) to the operator/cloud provider (the company that owns the vehicle).

1.1.2 Benefits of Serverless Computing

The transition between serverful to serverless cloud computing is a promising development, as it introduces new opportunities for programmers to utilize the cloud. From these opportunities, the realm of cloud computing can be further developed.

1.1.2.1 Simplicity

By removing the server maintenance responsibility from the programmer, developers are free to focus their time and energy on the application they are working on rather than the requirements of the server. Effectually, these developers can simply provide an event-driven function and, in response, the cloud-provider can meet the demands of the function as the event triggers occur [14]. This increased simplicity would make cloud-based applications a more feasible goal for programmers. Additionally, operators and server providers would also have the increased benefit of having more simplified scalability and language flexibility as provided by many serverless platforms. This simplicity for both developer and operator provides a benefit to the interests of progress in computer science and engineering, with more programmers able to work on more projects and develop more solutions.

1.1.2.2 Cost Efficiency

Another major benefit of the serverless computing approach is improvements in cost management. With serverless computing's "pay-as-you-go" method, programmers will not be paying for "idle time" during specified reservations but only paying for the resources they are consuming. This further makes the development of cloud-based applications more available for programmers who may not have the resources to fund a project using the serverful approach. The benefits of cost-efficiency are not exclusive to the programmer, but may also be enjoyed by the operator offering the servers. Serverless platforms allow operators to better manage their servers so that functions only consume resources when necessary.

1.1.3 Management of Serverless Workloads

Despite promising rewards, the continued evolution of serverless computing is dependent on further research. As demand for serverless applications begins to grow, the effective management of serverless workloads becomes an expanding area of interest. Many existing scheduling mechanisms for serverless platforms fail to cater to the unique characteristics of the applications that utilize them [15]. By investigating the current scheduling practices of cloud providers, areas of potential improvement may be discovered. By addressing these areas, the effects of undocumented and unexpected behavior can be counteracted [16]. Some such issues that warrant further research in improving current serverless platform scheduling methods are inflated application run times, function drops (i.e., unavailable or unreliable services), and inefficient allocations that result in additional energy consumption and increased hardware costs. Inflated application run times, specifically, can occur due to extended start-up times caused by cold starts. Cold starts may be defined as the time required to set up a serverless application's environment when it has been invoked for the first time within a defined period. These and other performance-related issues may have minimal consequences for smaller scale projects but lead to great cost as the scale of the application is increased. For the emerging workloads targeted by serverless platforms (such as virtual reality, vehicular communication, smart manufacturing, and smart cities), the performance overhead may be unacceptable, precluding the use of serverless platforms.

1.2 Open-source Serverless Platforms

There are currently many options for programmers to harness serverless technology to fit their needs. Besides commercially available platforms offered by providers such as AWS Lambda [17], Microsoft Azure Functions [18], Google Functions [19], and IBM Functions [20], several communities invested in creating open-source offerings. Open-source projects for building serverless platforms, specifically, present the opportunity for unique innovations within a format accessible to the individual developer. The public availability of these projects allows more programmers to become exposed to serverless frameworks and pushes serverless projects to the forefront of cloud computing development. There is a selection of open-source serverless platforms that are specifically friendly for those with no experience in serverless computing and act as good resources for programmers who have started experimenting with serverless computing or academic researchers who seek to investigate serverless computing but lack access to industry-level tools. Nonetheless, changing these open-source frameworks to optimize their services still requires a high level of expertise in system software.

The following sub-sections present the most prominent open-source serverless frameworks currently available.

1.2.1 Apache OpenWhisk

The Apache OpenWhisk framework is notable for its large number of contributors and powerful features. It is an event-driven programming model that supports any functional programming language [21]. The OpenWhisk programming model responds to events such as changes to a database or web application interactions by creating channels from these events. These channels are referred to as “triggers,” and rules associate these channels with specific actions. These actions encapsulate the application logic to be executed in response to a specific event [21]. Using this logic, OpenWhisk is capable of executing code in response to events at whatever scale the developer may need.

The easiest and most straightforward setup for OpenWhisk requires Docker, Java, and Node.js to be available on the local machine. This effectually installs an OpenWhisk stack which runs as a Java process. Using Docker Compose in order to use Docker directly is another quick start alternative that lacks the full set of OpenWhisk features but allows for the execution of functions [21].

Compared to other alternatives, the OpenWhisk framework tends to have a steeper learning curve, and its large tools can pose difficulties for developers. Another drawback is that the resource management algorithms available in OpenWhisk are too simple to be considered realistic representations of the state-of-art commercial offerings; therefore, comparing a new proposed scheme with the original OpenWhisk is not sufficient to demonstrate the value of the proposed changes.

1.2.2 Fission

Fission is an open-source Kubernetes-native serverless framework. Similarly to OpenWhisk, Fission supports any language. It accomplishes this by isolating language-specific sections in parts of Fission called “environments,” which contain just enough software to run functions. Function invocations are triggered by HTTP requests, and can also be managed with KEDA based message queue triggers [22].

A major advantage of the Fission framework is that it has a specific method by which it counteracts the workload management issue discussed previously: cold start. The method employed by the Fission framework is to maintain a number of “warm” containers, containers already up and running in preparation for an incoming function call. Each of these “warm” containers has a small dynamic loader. Thus, when a function is called, one of these running containers is loaded, so the delay typically caused by a cold start is minimized. By using this method, the Fission framework is able to reduce cold-start latencies to an average of 100msec [22]. This allows Fission’s median response time to average less than that of other alternatives, but only within a certain number of concurrent users [23].

Overall, Fission provides a framework where programmers can write short-lived functions, building them within the Kubernetes framework.

1.2.3 IronFunctions

IronFunctions is a platform that harnesses the power of microservices for developers seeking an easy and cohesive resource for serverless computing projects. This platform relies on Docker to operate and can be run on single server mode to quickly get started and work on functions. The platform itself is written in Go [24], but like the other serverless platforms discussed, it allows for language flexibility on the part of the developer. This platform uses routes to define paths in applications that map to a particular function. There are fewer contributors to this platform than the previous platforms, but this particular option primarily focuses on ease of use and simplicity.

1.2.4 OpenFaaS

OpenFaaS is a platform that provides a simple framework for deploying to Kubernetes but can also deploy to OpenShift or to a single host with faasd. The primary focus of OpenFaaS is to avoid repetitive boiler-plate coding, offering a developer-friendly abstraction on top of Kubernetes. This is accomplished by installing OpenFaaS to a Kubernetes cluster using the native CLI installer, or other options such as Helm chart, Flux or ArgoCD. OpenFaaS is designed to run on any form of Kubernetes cluster without restriction. A major benefit of this particular framework is the ability to auto-scale according to demand.

1.2.4.1 Faasd

OpenFaaS offers a lightweight, cost-conscious version of the platform called faasd. This version of OpenFaaS is designed to be beginner-friendly and to allow developers to start deploying their functions as quickly as possible. This particular option specifically targets programmers who do not have the time or background knowledge to manage a Kubernetes cluster and/or have limited resources for their project. Faasd can run on a single host with modest requirements for operation, making it an optimal choice for personal use. Much of how it is used remains consistent with the full OpenFaaS framework, including the same tooling, ecosystem, templates, and container, but not requiring cluster management. Due to this, resources available for OpenFaaS remain relevant for a faasd user. Faasd has greater limits on its functionality, but for projects that require few operations, these limits are no issue.

1.3 The Process of Choosing a Platform

Choosing a platform with which to pursue a serverless computing project is highly dependent on the specific needs and context of that individual project. This choice will also depend on the experience level of the developer and what tools they are already familiar with. Considering the context and allotted timeframe of this research, it was found that the lightweight version of OpenFaaS, faasd, was the best option for the purposes of this particular study. Faasd's portability and minimal requirements, in addition to its independence from Kubernetes, make it optimal for individual research.

2. METHODS

A common challenge for research areas evolving while the ideas are also undergoing widespread adoption in the industry, as is the case with serverless computing, is to demonstrate that proposed ideas can be impactful under real-world conditions. For researchers in industry, there is usually the possibility of creating an experimentation environment that uses a software infrastructure and architecture very similar, if not the same, to the in-production deployment. Academic researchers have to take an indirect route when validating their research results, working on existing open-source software infrastructure, and finding ways to demonstrate that the experimental environment reflects some aspects of the real, in-production one.

Because industry offerings for serverless technology are already widespread, even though the research area is quite new, there have been efforts to create open-source implementations of serverless platforms, as listed in Chapter 1. These projects aim at a role similar to what Xen [25], [26] and KVM [27], [28] had on research in virtualization or what Eucalyptus [29] and OpenStack [30] had on research on Infrastructure-as-a-Service (IaaS). Among the many alternatives, there is no clear winner yet.

This chapter describes our process for selecting an open-source platform and identifies other components or tools needed in this research project. It also documents the method we deployed to build our experimental environment.

2.1 Laying the Foundation

In pursuing further knowledge regarding open-source platforms for serverless computing, it is first necessary to develop a foundational understanding of the role serverless computing holds in both industry and research. Having studied the differences between serverful and serverless computing and completing a basic literature overview for the purposes of establishing the necessary background information, hands-on hardware-based experimentation with a serverless platform is the next logical step. Unlike serverless research conducted within the realm of industry, academic research does not have access to tools guarded by intellectual property regulations, and therefore is highly dependent on open-source platforms in order to pursue new solutions. Another challenge for the research community is the choice of workloads to use when assessing new ideas. Serverless providers have access to real-world workloads, thereby targeting optimizations designed to improve their services. The research community has to rely on benchmarks such as [31] and [32]. Recently, one provider released datasets revealing high-level information on the usage of their serverless platforms [33], enabling researchers to deploy benchmarks that mimic the patterns observed in production.

2.2 Choosing an Open-source Platform

Of the open-source platforms described in section 1.2, the initial choice for further investigation was the open-source cloud framework OpenWhisk. This choice is quickly reconsidered, the reason being that while a popular framework and possessing incredibly powerful tools, the complexity of the OpenWhisk framework may require a considerable amount of time for programmers to learn how to use its features to their full advantage. Therefore, it is necessary to pursue an option that is better suited the time frame allotted for this research. OpenFaaS proves to be a more fitting option, specifically the lightweight version offered by the platform developers: faasd.

Instructions and documentation detailing setup and uses for faasd can be found on the GitHub repo where the platform is stored [34], including an official handbook. Most resources related to the use of OpenFaaS are also transferable to operating within the faasd framework, as the two platforms are highly similar with a few exceptions. Most notably, faasd is far more limited with regards to scale and tools. This is due to the fact that faasd is intended to be a light-weight alternative to OpenFaaS, designed to run on a single host and requiring fewer resources. This makes it a suitable platform to focus on for this research.

Having identified the suitable framework, this research tackles the operational steps required to build a runtime environment for the experimental evaluation of the selected framework.

3. RESULTS

The deployment and refinement of open-source cloud frameworks presents significant technical challenges. Often, such research requires multiple years of effort, being carried out in the context of Ph.D. dissertations [35, 36, 37] or in industry research labs [33, 38, 39, 40]. Another path that has been adopted in serverless research is to build from scratch a runtime environment to demonstrate the value of a proposed design or mechanism (as done, for example, in [41], [42]). As described in Chapter 2, our work identifies and documents the operational steps in setting up an open-source serverless framework for research experimentation.

3.1 AWS Lightsail Instance

It is not recommended that faasd be operated on the same host on which Docker [43] is installed, as faasd utilizes containerd for its runtime and may use a different version than a given Docker installation. Additionally, Docker's networking tools can disrupt faasd's networking, causing issues with deployment. This being understood, an Amazon Web Service (AWS) Lightsail instance is used to act as the host for faasd, while Docker is installed on a local machine.

An AWS Lightsail instance is a virtual private server (VPS) that lives in the AWS Cloud. A VPS, or virtual private server, is a method for cloud hosting in which virtualized server resources are provided to an end-user via a cloud or hosting provider. AWS is one such hosting provider, and its advertised goal is to make the process of creating virtual private server instances, containers and databases quick and cost-effective. Preceding the first steps in this research, a Lightsail instance was set up for the purpose of exploring serverless open-source platforms. Through the AWS console, this Lightsail instance can be accessed using a browser-based SSH client. When connected, a new browser window is opened to allow access to the instance's console.

After connecting to the AWS console, the first step in experimenting with faasd is to deploy faasd within the instance. The following commands are entered into the console:

```
git clone https://github.com/openfaas/faasd --depth=1
cd faasd
./hack/install.sh
```

This installation script installs the required libraries to begin using faasd within the Lightsail instance. Within the OpenFaaS GitHub repository, there are suggested numbered documents titled "labs" that allow a user to experiment with the basics of using the OpenFaaS framework [44]. While some of these labs contain content that is irrelevant to the lightweight version of OpenFaaS, they allow for familiarization with some basic commands. Initially, a roadblock is encountered when attempting to run the sample functions due to an unauthorized access error. This is solved with the following command:

```
sudo cat /var/lib/faasd/secrets/basic-auth-password | faas-cli
login -s
```

To deploy sample functions, the following command is called:

```
faas-cli deploy -f
https://raw.githubusercontent.com/openfaas/faas/master/stack.y
ml
```

To list these deployed sample functions, the following command is used:

```
faas-cli list
```

A similar command achieves the same list, with the added information regarding the corresponding docker image:

```
faas-cli list --verbose
```

In order to invoke an example function named “markdown” the following command is used, and sample text is entered into the console:

```
faas-cli invoke markdown
```

A final introductory command allows for the generation of an HTML file from the corresponding markdown file of the instructional lab document within the OpenFaaS GitHub repository:

```
git clone https://github.com/openfaas/workshop \
        && cd workshop
      cat lab2.md | faas-cli invoke markdown
```

Having obtained a rudimentary understanding of basic commands to deploy and list sample functions as well as generate HTML files from markdowns, it is now time to attempt the creation and deployment of a basic function for testing purposes. Faasd allows you to view the full range of official potential template options by running:

```
faas-cli template store list
```

Unofficial templates can also be found on GitHub that are compatible with faasd, but this command will only list the official templates, many of which are sourced from OpenFaaS directly. For the purposes of simplicity, the template named “python3” with the description “Classic Python 3.6 template” is selected for testing. This new function is started by running:

```
faas-cli new test-function --lang python3
```

Having been created, the function can be developed within the handler.py file generated within the newly created test-function folder. Within our handler.py file we write:

```
def handle(req):
    """handle a request to the function
    Args:
        req (str): request body
    """
    print("Hello World")
    return req
```

In editing this file, we have created an extremely simple function called "test-function" that prints the phrase "Hello World."

After this we seek to run the following command in order to deploy the function:

```
faas-cli up -f test-function.yml
```

This is where we encounter another roadblock. After attempting to deploy the function, we receive an error indicating that the "docker" executable file is not found within the path. This makes sense, as we have proceeded with attempting to deploy a function using faasd without installing Docker on the instance or pushing any Docker images to the Lightsail service. As suggested in the README.mb file stored on the faasd GitHub repository, the faasd server should be maintained as a faasd server, while a container image may be built on a local machine or in a CI pipeline. We will choose the former option as we proceed.

3.2 Pushing Docker Images to Lightsail Service

Throughout this next section, we will be using the Amazon Lightsail documentation to inform our process of creating a container image for our Amazon Lightsail container service. We will first create a Docker image on our local machine. Next, through the AWS console, we will setup a container with specifications that suit the needs of this research. Finally, we will then push the docker image to our Lightsail service.

3.2.1 Creating the Docker Image

The first step is the create the Docker image on a local machine. Docker is a platform for developing, shipping, and running applications. For our purposes specifically, Docker can be used to create container images, which are an executable package of software that includes everything needed to run an application, such as code, system libraries and settings. This package of software is what will be pushed to the AWS Cloud where it can be used by the serverless platform.

The installation prerequisites required for this process are Docker, the AWS Command Line Interface (AWS CLI), and the Lightsail Control plugin. The AWS CLI is a required prerequisite as it allows for the parameters of the container images to be specified and enables the process of pushing the image to the Lightsail container service. The Lightsail Control (lightsailctl) plugin allows the AWS CLI to access container images on the local machine.

After installing the AWS CLI and saving the lightsailctl executable to the C:\Temp\lightsailctl\ directory on the local machine and setting the path, the Dockerfile can be created. For this initial test, it is only necessary to create a simple static website, so a folder named “mystaticwebsite” is created, and a Dockerfile (named “Dockerfile” without a file extension) is placed in the folder. A simple static website container image with the message “Hello World” can then be copied into the Dockerfile. The container image is then built by issuing the following command through the command prompt terminal (before executing the build command, we must make sure that Docker is already running on our local machine):

```
docker build -t mystaticwebsite .
```

Having built the image, we can verify that the image was properly created by ensuring that it is listed when running the command:

```
docker images --filter reference=mystaticwebsite
```

In order to run the new container image, the following command can be issued in the console:

```
docker container run -d -p 8080:80 --name mystaticwebsite
mystaticwebsite:latest
```

This command maps port 80 on the container to port 8080 on the local machine where the container has been built. The following command allows us to view all running containers:

```
docker container ls -a
```

By using this command, we can ensure that our container is running, further confirming this by visiting the URL “http://localhost:8080” within our browser.

3.2.2 Creating a Lightsail Container

Having created a Docker image on our local machine, we need to push it to an AWS Lightsail service. The AWS CLI and Lightsail Control plugin are set up to push Docker images to Lightsail containers. Therefore, we need to return to the AWS console and create a container to receive the static website image that we created. Within the AWS Lightsail console, we choose the option to create a new container service. When choosing our container service capacity, we select the 2 GB RAM, 1vCPU option so as not to be restricted. Our requirements will depend on our research needs, and we don’t want to create an unnecessary bottle neck as we proceed. We set our scale to a single node. Having set these specifications, we have created our container service.

3.2.3 Pushing the Image that We Created to Our Container Service

We are now ready to push our Docker image to our newly created container service. First, we can run the following command to view all images currently on our local machine:

```
docker images
```

This command allows us to verify the repository name, tag, image ID, time of creation, and size of every image on our machine. For our purposes, the tag is the most important piece of information to focus on. To push the image to our container service, we run the following command:

```
aws lightsail push-container-image --region us-east-2 --
service-name kali-bailey-container-1 --label mystaticwebsite -
-image mystaticwebsite:latest
```

The portions indicated in red within the command above are the sections that are unique to the individual user, while the rest is the standard specifications of the command. For *region*, we must view the container service that we've created within the AWS Lightsail console and take note of the state listed underneath the name of the container service. We can then cross-reference this with the list of regions provided within the AWS user guide to determine the region [45]. The service name should be the exact same name we previously determined as part of the process of creating our container service as described in section 2.4.3. The label should be the repository name of the Docker image. Lastly, the image should be input as the repository name of the Docker image, followed by a colon, followed by the tag of the Docker image.

During the execution of this command, another obstacle to progress reveals itself. Despite having set the path to the Lightsail Control plugin with the following command, the AWS CLI is unable to locate the plugin:

```
setx PATH "%PATH%;C:\Temp\lightsailctl" /M
```

After reinstalling the plugin and resetting the path, it is found that the best temporary solution is to navigate directly into the folder where the plugin was stored and then activate the

AWS command to push the container. While not an ideal solution, it allows the command to operate as intended.

After receiving the confirmation that the image was pushed following the input of the previously described command, this image can be viewed within the Lightsail console. Navigating to the Lightsail console and clicking on the container service we created, our newly pushed container image can be viewed under the “Images” tab.

3.3 Deploying Containers

We can create a deployment that utilizes our container image. Under the “Deployments” tab, we can choose to create a new deployment. We can then specify the name of our container service, and because we pushed our Docker image locally, choose the option “Choose stored image.” Other configuration options are available, but for a basic deployment, these are the only steps necessary.

Note that all methods described have been accomplished using a Windows local machine. Replicating results using a macOS or Linux machine may require altered methodologies to produce the same results.

4. CONCLUSION

Throughout the process of this research, we have had the opportunity to explore AWS Lightsail services, the functionality of Docker, and the creation of functions utilizing faasd tools. This process has exposed us to the world of containerized applications as well as many open-source platforms for serverless computing. We've also had the opportunity to compare and contrast multiple open-source platforms during the course of our research.

After an initial exploration to identify and assess existing open-source frameworks in terms of complexity, the main goal has been to get faasd to operate by using AWS services to host the platform and using Docker on the local machine to provide images. We identified a process to achieve our goal and documented its steps so that other researchers (in particular, the ones without extensive experience in system software) can replicate it.

Besides identifying a successful sequence of installation steps, throughout our exploration we also learned that an alternative method that may be more effective than our proposed procedure. A method, not attempted but suggested by the documentation provided on the OpenFaaS GitHub repository, was to use multipass to create a VM which would host faasd. Multipass is a lightweight VM manager that can be used by Linux, Windows and macOS operating systems. It is designed for programmers who need to create an Ubuntu environment quickly. This option may have allowed for a quicker setup and less roadblocks in pursuing experimentation with faasd.

A surface-level investigation of additional faasd introductory material [46] provided a more detailed guideline on how to proceed with further experimentation with the framework. The material mentioned multiple methods by which hosting could be accomplished (including cloud hosting), but the outlined steps focused specifically on the method previously described: using Multipass to set up a virtual machine with which to host the faasd framework. Additionally, while installation instructions for Windows machines were provided, the material and accompanying tutorials were performed on a macOS.

4.1 Challenges in Research

Serverless computing is a very active area of research in both academia and industry. It is an attractive topic due to its high demand in furthering the capabilities of the cloud, satisfying industry needs, and providing new opportunities for individual programmers. There are, however, some challenges to pursuing this area of research.

One of the first challenges that presents itself is in the overwhelming number of options available for platforms and tools. Because this research area is quite new, there is not a lot of experience documented and disseminated through publications yet. Not being able to build upon the operational methods of other researchers, we had to try out several alternatives and quickly settle on a single choice. When introducing this report, four different open-source platforms for serverless computing were described, but this is far from an exhaustive list. There are many more options, all of which have unique strengths and weaknesses. Choosing an option is highly dependent on the optimization being pursued and the individual needs of the programmer, as well as the programmer's background and experience level.

Another challenge that arises when pursuing this research is a lack of resources for troubleshooting errors. While OpenFaaS provides documentation for potential uses for the platform, it can be difficult to find answers to specific issues or questions. Additionally, when specific errors arise when interacting with the AWS CLI, there seems to be very few resources to resolve them.

Another roadblock that is of increased concern for beginner researchers (e.g., undergraduate students or graduate students with little experience in cloud computing technology) is that certain terms are used differently by different sources, which may lead to confusion. Individuals pursuing research in this area should ensure that they have a very clear understanding of the differences between terms such as “instance,” “container,” and the purposes of these frameworks. Our work contributes to this ecosystem by documenting a process derived through time-consuming trial-and-error.

REFERENCES

[1] M. Armbrust et al., “A view of cloud computing,” *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] S. J. Berman, L. Kesterson-Townes, A. Marshall, and R. Srivathsa, “How cloud computing enables process and business model innovation,” *Strategy & Leadership*, 2012.

[3] K. E. Kushida, J. Murray, and J. Zysman, “Cloud Computing: From Scarcity to Abundance,” *Journal of Industry, Competition and Trade*, vol. 15, no. 1, pp. 5–19, 2015.

[4] A. Lamkin, “How the Cloud Transforms Industries,” *Forbes Technology Council*, 09-Jul-2018. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2018/07/09/how-the-cloud-transforms-industries. [Accessed: 05-Mar-2022].

[5] E. Jonas et al., “Cloud Programming Simplified: A Berkeley View on Serverless Computing,” *arXiv preprint arXiv:1902.03383*, 2019.

[6] I. Pietri and R. Sakellariou, “Mapping Virtual Machines onto Physical Machines in Cloud Computing: A Survey,” *ACM Computing Surveys*, vol. 49, no. 3, pp. 1-30, Sep. 2017.

[7] D. Merkel, “Docker: lightweight Linux containers for consistent development and deployment,” *Linux Journal*, Mar. 2014.

[8] L. Wang, M. Li, and Y. Zhang, “Peeking Behind the Curtains of Serverless Platforms,” *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pp. 133–146, 2018.

[9] I. E. Akkus, “SAND: Towards High-Performance Serverless Computing,” *2018 Usenix Annual Technical Conference (USENIX ATC 18)*, pp. 923–935, 2018.

[10] S. Shillaker and P. Pietzuch, “Faasm: Lightweight Isolation for Efficient Stateful Serverless Computing,” *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pp. 419–433, 2020.

[11] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, “The rise of serverless computing,” *Communications of the ACM*, vol. 62, no. 12, pp. 44–54, Nov. 2019.

[12] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, “Cloud Container Technologies: A State-of-the-Art Review,” *IEEE Transactions on Cloud Computing*, vol. 7, no. 3, pp. 677–692, 2017.

[13] J. Schleier-Smith *et al.*, "What serverless computing is and should become: The next phase of cloud computing," *Communications of the ACM*, vol. 64, ed, pp. 76-84.

[14] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking Behind the Curtains of Serverless Platforms," *2018 USENIX Annual Technical Conference (USENIX ATC 18)*: USENIX Association, 2018, pp. 133-146. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/wang-liang. [Accessed: 24-Jan-2022]

[15] K. Kaffes, N. J. Yadwadkar, and C. Kozyrakis, "Centralized Core-granular Scheduling for Serverless Functions," in *SoCC '19: Proceedings of the ACM Symposium on Cloud Computing*, 2019, pp. 158-164.

[16] A. Tariq, A. Pahl, S. Nimmagadda, E. Rozner, and S. Lanka, "Sequoia: enabling quality-of-service in serverless computing," in *SoCC '20: Proceedings of the 11th ACM Symposium on Cloud Computing*, pp. 311-327, 2020.

[17] “AWS Lambda,” *Amazon*. [Online]. Available: https://aws.amazon.com/lambda/. [Accessed: 31-Mar-2022].

[18] “Azure Functions,” *Microsoft*. [Online]. Available: https://azure.microsoft.com/en-us/services/functions/. [Accessed: 31-Mar-2022].

[19] “Cloud Functions,” *Google*. [Online]. Available: https://cloud.google.com/functions. [Accessed: 31-Mar-2022].

[20] "IBM Cloud Functions," *IBM*. [Online]. Available: https://cloud.ibm.com/functions/. [Accessed: 31-Mar-2022].

[21] Documentation. [Online]. Available: https://openwhisk.apache.org/documentation.html. [Accessed: 24-Jan-2022].

[22] Docs. [Online]. Available: https://fission.io/docs/. [Accessed: 24-Jan-2022].

[23] S. K. Mohanty, G. Premsankar, and M. Di Francesco, "An evaluation of open source serverless computing frameworks," *Proceedings - IEEE 10th International Conference on Cloud Computing Technology and Science, CloudCom 2018*, pp. 115-120, 2018.

[24] "The Go Programming Language," *Go*. [Online]. Available: https://go.dev/. [Accessed: 31-Mar-2022].

[25] "XenProject," *The Linux Foundation*. [Online]. Available: https://xenproject.org/.

[26] P. Barham et al., "Xen and the art of virtualization," *ACM SIGOPS operating systems review*, vol. 37, no. 5, pp. 164–177, Dec. 2003.

[27] "KVM Project," *RedHat Openshift Online*. [Online]. Available: https://www.linux-kvm.org/.

[28] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the Linux virtual machine monitor," *Proceedings of the Linux symposium*, vol. 1, no. 8, pp. 225–230, 2007.

[29] A. Kivity et al., "The eucalyptus open-source cloud-computing system," *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pp. 124–131, 2009.

[30] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "OpenStack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, pp. 38–42, 2012.

[31] M. Copik et al., “Sebs: A serverless benchmark suite for function-as-a-service computing,” *Proceedings of the 22nd International Middleware Conference*, pp. 64–78, 2021.

[32] P. Maissen et al., “Faasdom: A benchmark suite for serverless computing.,” *Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems*, pp. 73–84, 2020.

[33] M. Shahrad et al., “Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider.,” *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pp. 205–218, 2020.

[34] A. Ellis, "faasd," GitHub, Available: https://github.com/openfaas/faasd. [Accessed: 31-Mar-2022]

[35] E. Feller and C. Morin, “Autonomous and energy-aware management of large-scale cloud infrastructures,” *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pp. 2542–2545, 2012.

[36] K. M. Maiyama, "Performance Analysis of Virtualisation in a Cloud Computing Platform. An application driven investigation into modelling and analysis of performance vs security trade-offs for virtualisation in OpenStack infrastructure as a service (IaaS) cloud computing platform architectures," PhD Dissertation, Univ. of Bradford, 2019.

[37] Md. A. Ullah, "Design and Implementation of a framework for Multi-Cloud Service Broker." PhD Dissertation, PhD. Thesis, Ryerson Univ., Canada, 2015.

[38] X. Ju et al., "On fault resilience of OpenStack," *Proceedings of the 4th annual Symposium on Cloud Computing*, pp. 1-16, 2013.

[39] E. Cortez et al.,"Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," *Proceedings of the 26th Symposium on Operating Systems Principles*, pp. 153-167, 2017.

[40] A. Fuerst et al., "Memory-harvesting VMs in cloud platforms," *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 583-594, 2022.

[41] Z. Jia and E. Witchel, "Nightcore: efficient and scalable serverless computing for latency-sensitive, interactive microservices," *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 152-166, 2021.

[42] Z. Jia and E. Witchel, "Boki: Stateful Serverless Computing with Shared Logs," *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pp. 691-707, 2021.

[43] “Get Started with Docker,” *Docker*. [Online]. Available: https://www.docker.com/get-started/. [Accessed: 31-Mar-2022].

[44] A. Ellis, "openfaas," *Github*. [Online]. Available: https://github.com/openfaas/workshop. [Accessed: 31-Mar-2022]

[45] "Regions, Availability Zones, and Local Zones" Available: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.RegionsAndAvailabilityZones.html

[46] A. Ellis, “Serverless for Everyone Else,” *OpenFaaS Store.* [Online]. Available: https://openfaas.gumroad.com/l/serverless-for-everyone-else