Towards Low-Cost Global Highly Available Large Container-Based Serverless Functions

Jasleen Kaur, Robert Cordingly, Ling-Hong Hung, Wes Lloyd
School of Engineering and Technology
University of Washington
Tacoma, Washington USA
jaslkaur, rcording, lhhung, wlloyd@uw.edu

***Abstract*—Serverless computing and the Function-as-a-Service (FaaS) paradigm have transformed how cloud-hosted applications are deployed, providing remarkable scalability and geographical distribution capabilities. AWS Lambda's support for containerized functions has further enhanced the packaging and deployment process, enabling seamless portability and consistency across various operational environments. However, a fundamental challenge remains: the replication of Docker container images across multiple container registries in different regions to ensure high availability of globally deployed functions with low replication latency to support geographically dispersed clients.**

While multi-region deployment minimizes latency and enhances service availability for diverse clients, the associated costs and operational complexities are non-trivial. The replication process, particularly for large container images, is both expensive and slow, potentially hindering rapid scalability and responsiveness. Public cloud container image replication support such as that offered by Amazon's Cross Region Replication (CRR) represents a significant stride forward, automating the replication of container images across regions within container repositories. Despite its advantages, CRR is constrained by several limitations: all images in a container repository are replicated after this feature is enabled, and there is an absence of fine-grained filtering at the image level within repositories. This can result in superfluous replications and increased costs.

This paper investigates alternatives for cost-effective and efficient global deployment of container-based serverless functions on AWS Lambda. We introduce on-demand container image replication mechanisms with promise to reduce storage costs by avoiding unnecessary replication. We investigate multiple approaches to replicate container images only when needed to minimizedata transfer latency and storage overhead. We provide a comprehensive evaluation of these approaches, assessing their impact on cost, replication time, and turnaround time for function availability. Our findings demonstrate potential for substantial cost savings while maintaining high availability and consistent performance for container-based Lambda functions deployed worldwide.

***Index Terms*—Serverless Computing, AWS Lambda, Elastic Container Repository, S3, Docker, CRR, On-demand Replication, Cost Optimization**

I. Introduction

Serverless computing, particularly Function-as-a-Service (FaaS), has gained popularity for cloud application hosting due to its abstraction of infrastructure management [1]. Cloud providers such as Amazon Web Services, Microsoft Azure, and IBM Cloud each offer serverless computing platforms. With serverless computing, software packaged as functions are deployed to individual cloud regions. The number of cloud regions, which are sets of geographically local data centers has grown substantially, offering opportunities to leverage the best resources based on various objectives [2]–[4]. For example, users may choose regions based on client proximity to minimize latency or select regions with low resource contention during off-business hours for better performance and cost savings [5] [6]. Cloud providers, however, lack mechanisms to automatically scale serverless function deployments globally.

Recently Function-as-a-Service serverless platforms have introduced support for hosting functions based on application containers like Docker, simplifying deployment by packaging function code, libraries, and data into containers [7]. AWS Lambda supports container-based functions using container images stored on Amazon S3 or the Amazon Elastic Container Registry (ECR). However, container-based Lambda functions can only use container images from private regional ECR repositories, not public repositories. This design requires duplication of container images **in every region** where functions are deployed. To store container images in private ECR repositories costs $0.10/GB/month/region [8], which totals $12/year/region for a each function with a 10GB container image. If looking to replicate a function across all AWS regions globally this results in a cost greater than $\tilde{\$}$300/year simply for image storage. This does not consider the cost of executing functions. Thus, providing high availability for container-based functions globally, especially those with large images, is expensive and challenging [9].

This paper investigates alternative approaches to reduce the costs of providing global high availability for container-based serverless functions, particularly those requiring large container images. These approaches can support creation of a just-in-time deployment tool to enable global container-based function deployment on demand. We compare our approaches to deploy large container based functions on demand with the Elastic Container Replication (ECR) service as a baseline. We investigate tradeoffs between container image storage cost and data transfer latency. Our envisioned workflow involves: creating container-based functions in one region, triggering container image replication in target region(s) on demand, and deploying functions using the transferred images from local ECRs. We focus on challenges associated with moving larger container images (e.g. ˜10 GB) for serverless function execution.

A. Problems with cloud provider managed image replication

Achieving high availability for containerized applications often requires replicating container images across multiple regions, which can be both costly and time-consuming. Amazon addressed this challenge with the launch of ECR Cross Region Replication (CRR) [10], a feature designed to automatically replicate container images across regions in the Amazon Elastic Container Registry (ECR). ECR CRR allows users to filter repositories using a regular expression to match the prefix of the repository name to configure Cross Region Replication [11]. When configuring a replication rule in a private registry, specifying a repository filter offers a way to manage which repositories are replicated. Without any filters, all repositories' contents are replicated by default. This filtering capability allows for granular control over replication, ensuring that only the desired repositories are replicated across regions. However, there are limitations to the current implementation. Only repository content pushed to a repository after CRR is configured is replicated, pre-existing content is not replicated. While filtering based on the repository name is supported, there is no image level filtering, leading to potential issues where CRR may replicate more images than necessary. These limitations are highlighted in a reported issue on GitHub [12], indicating the need for further refinement of ECR CRR to optimize image replication across regions.

B. Research Questions

This paper investigates the following research questions:

RQ-1: How should container images be replicated for on-demand use to support multi-region high availability of container-based serverless functions?

For RQ-1, For viable solutions, we calculate the ensuing storage costs in S3 or ECR, and transfer costs to make the data available for compute. We compare various methods for image replication and find the most cost-effective approach.

RQ-2: : How should container-based serverless functions be best created on-demand once a container image is made available locally in the region for execution?

For RQ-2, we propose using a proxy function to initiate deployment of container-based functions in regions lacking the container image. This proxy function manages image transfer and function creation, with control transferred to the new function post-deployment. We create a synchronous blocking proxy function, which waits for the container-based function to become available before calling it and returns results [13].

RQ-3: What are the cost implications for container image replication for different image retention policies and image replication methods?

For RQ-3, we determine worst-case costs to replicate images under different usage scenarios and image retention policies. Usage scenarios define different serverless function invocation frequencies, (e.g. 3x/hour, 3x/day). Image retention policies define how long an image should be retained in a regional repository (e.g. one hour, one day, one week)

C. Contributions

This paper makes the following research contributions:

1. Investigation of alternatives to maintaining continuous container image replicas across every region for global high availability of container-based serverless function deployments.
2. Investigation of tradeoffs for replicating container images on demand with an aim of reducing storage costs while also attempting to minimize time taken for on demand image replication to support global function high availability across regions, including the implementation of image-level filtering mechanisms.
3. Testing workloads different function invocations and container image retention policies to evaluate worst case cost of replication.

II. Background and Related Work

A. Benchmarking Functions-as-a-Service (FaaS) globally on any cloud

Multi-region FaaS involves deploying serverless functions in multiple regions to achieve better performance, high availability, and fault tolerance. It allows clients to run their applications closer to the users, reducing latency and improving user experience. Prior studies have investigated hosting FaaS functions across multiple regions or across multiple clouds. In one of the studies [4], performance of a Natural Language Processing (NLP) pipeline built using serverless FaaS functions is compared on two different CPU architectures (x86_64 and ARM64), and across four cloud regions on three continents over twenty-four hours. To minimize the function cold start latency, required data in S3 and associated Docker container images were stored in the same region as their dependent Lambda function. The cloud regions were selected across three continents to include a variety of times zones. Performance of the NLP pipeline was shown to be negatively impacted during busy times of a typical 9am to 5pm workday as compared to off hours. Another related effort on FaaS federation is rAFCL [14], a middleware platform that maintains the reliability of complex Function Choreographies (FCs), serverless workflow applications which connect serverless functions by data and control flow. The platform creates alternative strategies for each function of a FC based on user-specified availability, which are not restricted to the same cloud region. For example, FCs can automatically distribute alternative functions to multiple regions across the top five cloud providers without vendor lock-in, thereby ensuring that FCs continue to execute even when individual functions fail. The experimental results demonstrate that rAFCL outperforms the resilience of AWS Step Functions. Serverless functions can leverage deployments across multiple regions or even multiple clouds to take advantage of their capabilities, pricing, and geographical locations, but deploying functions across multiple clouds increases the risk of configuration and consistency errors due to the increased complexity and distributed nature.

B. Container-based serverless functions

Containers are isolated environments that provide a consistent runtime environment for applications. Serverless container images are typically small and lightweight. Cloud providers, like Google [15], Alibaba [16], and Amazon [7] support cloud functions to be deployed using container images. FaaSNET [17], a middleware system for accelerating container provisioning in Function-as-a-Service (FaaS) platforms was integrated into Alibaba Cloud Function Compute, where it proved to be scalable and fast. An experimental evaluation of FaaSNET found it to be extremely fast and highly scalable as it could initiate 2,500 function containers on 1,000 virtual machines in just 8.3 seconds. With FaaSNET integrated into Alibaba's Cloud Platform, it scaled container-based serverless functions approximately 13x faster than the original baseline approach used by Alibaba. Alongside serverless container provisioning tools, prior work has sought to improve the cold start latency of hosting container-based serverless functions in the cloud. Pigeon [18], a private cloud serverless FaaS framework built on top of Kubernetes, helps reduce function start up latency by using oversubscribed static pre-warmed containers. Compared to a Kubernete based serverless platform using the native scheduler, Pigeon improved throughput threefold while handling short-lived functions. Pigeon enhanced function cold trigger rates by 26% to 80% as compared to the AWS Lambda serverless platform.

C. FaaS deployment tools for pushing updates to different cloud/regions

GlobalFlow [17] is a workflow orchestration service that operates across multiple regions to effectively coordinate the execution of geographically distributed AWS Lambda functions that are necessary for cloud-based workflows. One of its strategies is the copy-based approach, where Lambda functions are copied from various regions to a target region to generate and execute a new workflow job in the target region. To aid FaaS application deployment across multiple regions, data may also need to be transferred or replicated across regions. The Performance and Profit Oriented Data Replication Strategy (PEPR) for cloud systems can evaluate whether data replication would be profitable for the cloud provider to help satisfy a cloud service level agreement (SLA) [19]. But to ensure profitability for customers or cloud users, focus should be made on minimizing costs without compromising the availability of serverless functions. To achieve better performance of FaaS functions, especially for minimizing cold start latency, data should be close to the compute. Approaches to ensure co-location of code and data have been previously discussed. The FaDO [20] orchestrator helps in viable placement of data dependencies across multi-serverless compute clusters in different regions. It uses HTTP reverse proxying and load balancers to direct function invocation requests to appropriate storage clusters offering distribution transparency to the clients in bringing data to compute. However, FaDO used Amazon's S3 storage solutions to interact with the data and is not concerned with container-based function images. In our research, we focus on moving data to compute to support container-based function invocations. The idea is to investigate the potential to copy container images to the region they are executed on demand to lower storage costs. To support container image replication, AWS provides [10] the Cross Region Replication (CRR) service that automatically triggers data replication of desired S3 buckets. It has been extended to Elastic Container Registries (ECR) to support replication of Docker container images in the specified regions automatically, when images are pushed to the private registry with the CRR feature.

D. Support for AWS Lambda container based functions

Container images are executable packages that contain the necessary software components to run an application or service, including code, libraries, and system tools. Container images are widely used by containerization technologies like Docker to enable the efficient and portable deployment of applications. Amazon announced support of container-based Lambda functions in December 2020 [7], [21]. Containers up to 10 GB can be packaged and deployed for use in serverless functions simplifying the development and deployment of large data-intensive functions that depend on dependencies like machine learning or bioinformatics datasets. The same operational simplicity, automatic scaling, high availability, and native integrations with various services available to functions packaged as zip archives also apply to functions based on container images. AWS provides publicly available base images, that include a runtime interface client to make the function code compatible for all supported Lambda runtimes, allowing developers to build images from the open-source images.

III. METHODOLOGY

A. Systems and tools

Multiple tools supported our study, as summarized in Table I: Docker container images and container based lambda functions for packaging and deployment of code, the Elastic Container Registry, and Amazon S3 for image storage.

TABLE I: Systems and tools summary

Function	Description
Docker Container Image	A 10 GB container image used to package function code.
Container-based Lambda	Serverless event-driven computer service to deploy container images.
Elastic Container Registry	Repositories to regionally store container images
Amazon S3	Object store service that has a lower pricing model as compared to ECR.
AWS CLI	Accessing AWS services using command line.

B. Tasks

We describe below preparatory and development tasks performed to support investigation of our research questions.

1) Implementation of alternative approaches to replicate container images across regions on demand: To address **(RQ-1)**, we evaluated the following methods to move container images to regions on demand:

a) Trigger to replicate container images in private ECR registries: Starting with one master copy of the container image stored in one ECR region, we used a Bash script embedded into a build to pull the container image from the master ECR. We then created private registries in local regions as needed and pushed the downloaded image to those registries to make the container image available in the desired regions using the same script. We used evaluation metrics listed in Section III-D to investigate the tradeoffs to copying the image ourselves versus triggering AWS CRR for replication using evaluation.

b) Store the Docker image in S3 and trigger replication in ECR: Since the storage cost of container images in ECR is high, we considered storing the container image data in S3 bucket(s) to reduce the cost. We created a master copy in one region's S3 bucket. On instantiating a build trigger, a Docker image was created using the raw data present in S3 and published in a local private ECR making the container image available for a container based Lambda function. To further reduce storage costs and build time for creating a Docker image, we also considered storing an image in S3 in compressed form [22] and then pushing it to the ECR registry.

2) Implement triggered Lambda function creation: For high availability of container-based functions, we prototyped a mechanism to transfer control from one Lambda function to another to evaluate how much additional time was required using this *Proxy Function Approach*. We triggered creation of a new container-based Lambda function once image replication was complete. We investigated passing-off control of the current function to the newly created function. The new function's calling information (i.e., function name) is already available to this proxy function and the responsibility for calling the new function was delegated to the proxy function. After the image was pushed to the desired ECR region, we triggered the creation of the new container-based function and then invoked the function. Figure 1 summarizes the regional replication steps for container based functions.

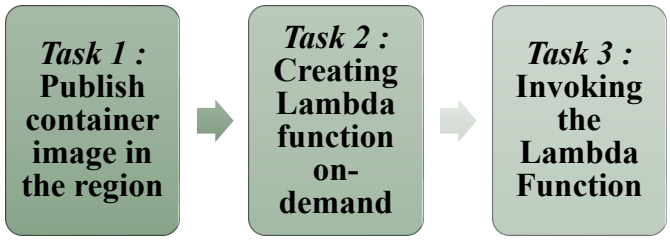

Fig. 1: Regional container based function replication steps

3) Implementing and testing static retention policies of container images: We investigated different static retention policies to evaluate their implications and to help understand how long we should retain container images in the ECR after a period of inactivity. This addressed **(RQ-3)** in terms of evaluating different schemes of purging to minimize costs when an ECR image is no longer in use. We proposed a variety of ways to call container-based functions based on different retention policies and invocation frequencies.

C. Experiments

1) Evaluating performance overhead to replicate data across regions on demand: This experiment leverages Task 1, which addresses **(RQ-1)**. We compared each of the methods to perform data replication discussed in Task 1 against the overall cost, and time to transfer the data, using the AWS ECR CRR service as a baseline (AWS CRR performs automatic cross region replication of container images to remote regions). We ran tests using a 10GB container image and performed image replication. To conduct Experiment 1, we used three different build services CodeBuild, GitHub Actions, and EC2 to transfer data to destination region in the following three ways:

1. **Build-from-S3**: Stored supporting source files to build Docker image in S3 to then push the image to ECR
2. **Transfer-using-ECR**: Implemented a trigger to replicate container images in private ECR registries from one region to another
3. **Transfer-from-S3**: Stored the Docker image in S3 and triggered replication in ECR

2) Evaluating time taken to pass of control to a Lambda function: In Task 2, we triggered the creation of a container-based Lambda function in the target region immediately after the image was successfully uploaded using the proxy function approach discussed before. To evaluate this, we measured the time taken to complete the transfer of control between the two Lambda functions. We also assessed the latency involved in invoking the newly created function, which directly impacts the overall responsiveness of the system. This experiment addresses **(RQ-2)**, crucial for understanding the operational characteristics and performance implications of our approach. It provided valuable insights into the practicality and feasibility of using container-based serverless functions for achieving high availability and low latency for multi-region serverless functions.

3) Evaluating the container image and container function retention policies: This experiment, associated with Task 3 and **(RQ-3)**, focused on evaluating the cost implications of different container image retention policies for various function usage scenarios. We considered a range of scenarios that vary in the frequency of function invocation within a given time window, such as X invocations per Y time (e.g., 3 calls per week, 4 calls per day). For each scenario, we evaluated different image retention policies, such as retaining the image for 1 hour, 4 hours, 1 day, etc., after use with an aim to determine the optimal image retention policy that balances cost and performance for different usage patterns. To calculate the cost, we considered the number of image replications required for each usage scenario with different retention policies, assuming that every function invocation results in a 'cache MISS' (i.e. when the image is not present in the desired region when the function is called), representing the worst-case scenario for cost calculation. The cost metrics for this experiment include image storage in Elastic Container

Registry (ECR) for a specified time period, data transfer for image replications across regions, and the cost to run the build services (e.g., EC2, AWS CodeBuild, GitHub Actions). By assuming the widest distribution of calls to examine the worst-case cost outcome, we provide insights into cost-effective strategies for managing container image storage and replication in a serverless environment.

TABLE II: Comparing tradeoffs between costs and build execution times for container image replication

Method	Build-From-S3		Transfer-Using-ECR		Transfer-From-S3	
	Time(s)	Cost(c)	Time(s)	Cost(c)	Time s	Cost(c)
CodeBuild	480	16	460	18	470	15.8
Github Actions	580	22	520	99	600	31
EC2 m5zn.2xlarge	503.6	9.2	588.5	14.1	541.88	10.1

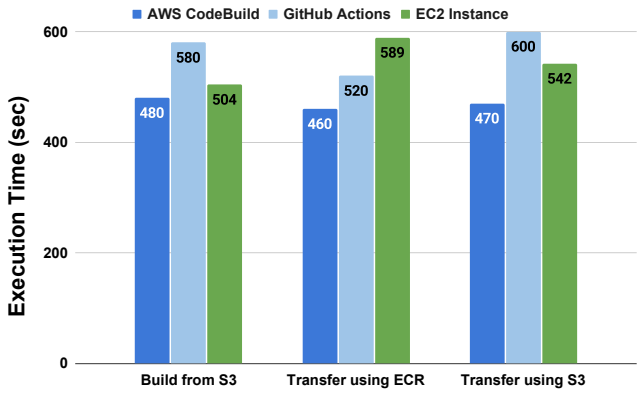

Fig. 2: Container image build and deployment times

D. Evaluation Criteria / Metrics

The evaluation for all the experiments was performed on a 10GB container image using the following metrics:

1. Cost
   - Storing container image in ECR
   - Storing container or data in S3
   - Transfer costs of container images/data from S3 to ECR or in between cloud regions
   - Cost of running the build using CodeBuild, Github Actions, or EC2 instance
   - AWS Lambda function runtime
2. Data replication or data transfer latency among AWS regions
3. Function turnaround time when the image is finally deployed.

IV. Results

A. RQ-1: Cost and time estimation of container image replication

Table II summarizes the experimental results from Experiment 1 comparing three different ways to transfer container images using AWS CodeBuild, Github Actions, and Amazon EC2. Figure 2 compares the build and deployment times between the three data transfer methods using three different services and Figure 3 compares the cost. AWS CodeBuild emerged as the fastest option across all scenarios, with data transfer times for large images ranging from 460 to 480 seconds. Conversely, EC2 proved to be the most cost-effective choice, with costs for large images ranging from 9.2 to 14.1 cents. These results highlight the trade-offs between time and cost when selecting a method for building and transferring Docker images across regions for use in a serverless environment.

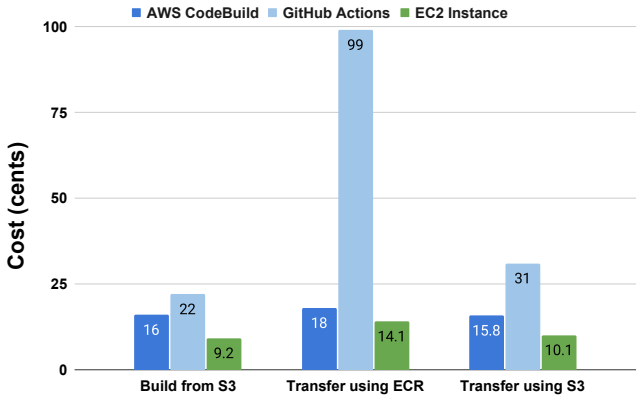

Fig. 3: Comparing Execution Costs (1 day retention)

B. RQ-2: Additional time for container-based Lambda function to be available once the image is replicated to destination region

After a container image is deployed to the destination region, in Experiment 2 we triggered the creation of a container based Lambda function with the new image and then invoked the newly created function. A proxy function was used to trigger image replication, create the container based Lambda function with the image, and pass control to the newly created Lambda function. Table III highlights the total time before a new container based function was available and ready for use. The additional turnaround time observed was 17.56 seconds.

TABLE III: Serverless function response time after image replication and function creation in the destination region

Scenario	Time (s)
Creating the container based Lambda function with the image	17.02
Invoking the lambda function	0.540

C. RQ-3: Worst case cost analysis of container image retention against invocation policies

This section describes the experimental results for Experiment 3 in which we examined the worst case costs of making a function available after completing the steps to replicate the container image and make the function available in the new region explained in figure 1. Table IV summarizes for different function invocation frequencies, how often a function is not available when invoked. For example, when there is a 'cache miss' for a one day image retention policy, how long will the image have to be stored in the ECR repository. For example, if we invoke the function 3x/day, perhaps once every 8 hours, and retain the image only for 1 hour, we would need to replicate

the image 3x/day in the worst case scenario (i.e. 3 hours in a day), resulting in 1.5 months/year.

TABLE IV: Retention time and annual ECR storage time of a container image based on function invocation frequency

Invocation Rate	Retention time (%) and annual ECR storage time			
	1 hr	1 day	1 week	1 month
Sc A: 3x / day	12.5% (1.5 mo/yr)	100% (12 mo/yr)	100% (12 mo/yr)	100% (12 mo/yr)
Sc B: 3x / week	1.79% (.21 mo/yr)	42.85 % (5.13 mo/yr)	100% (12 mo/yr)	100% (12 mo/yr)
Sc C: 3x / month	0.42% (.05 mo/yr)	10% (1.18 mo/yr)	69.90% (8.3 mo/yr)	100% (12 mo/yr)

Further, in Tables V, VI, and VII we show the cost of replicating an image using Github actions, an EC2 instance (m5zn.2xlarge), and AWS Codebuild respectively against different retention policies when storing the image in ECR for various lengths (e.g. 1 hour, 1 day, 1 week and 1 month).

TABLE V: Yearly cost of replication if an image is replicated using Github Actions

Invocation Rate	Image retention policies and ECR storage time			
	1 hr	1 day	1 week	1 month
Sc A : 3x/day	$1,059.38	$12.97	$12.97	$12.97
Sc B : 3x/week	$150.93	$155.84	$12.97	$12.97
Sc C : 3x/month	$34.83	$35.96	$11.67	$12.97

TABLE VI: Yearly cost of replication if an image is replicated using an m5zn.2xlarge EC2 instance

Invocation Rate	Image retention policies and ECR storage time			
	1 hr	1 day	1 week	1 month
Sc A: 3x / day	$119.54	$12.11	$12.11	$12.11
Sc B: 3x / week	$17.38	$22.29	$12.11	$12.11
Sc C: 3x / month	$4.01	$5.14	$12.26	$12.11

TABLE VII: Yearly cost of replication if an image is replicated using AWS CodeBuild

Invocation Rate	Image retention policies and ECR storage time			
	1 hr	1 day	1 week	1 month
Sc A: 3x / day	$163.56	$12.148	$12.148	$12.148
Sc B: 3x / week	$23.30	$28.22	$12.148	$12.148
Sc C: 3x / month	$5.38	$6.51	$13.63	$12.148

V. Conclusion

This paper has described our investigation of alternate approaches to replicate container-based images on demand and the associated time and costs. These findings support addressing the issue with ECR Cross Region Replication, specifically focusing on the problem of image-level filtering for high availability of container images across regions. Our experiments revealed that AWS CodeBuild was the fastest service for transferring large container images, with a container image transfer time of 460 seconds using the Transfer-Using-ECR method, which triggers replication of the desired image present in one private registry to the destination ECR region using the CodeBuild service. Further, we found replicating images using EC2 m5zn.2xlarge VMs to be the most cost-effective method of replication, with a cost of $0.5 for replicating one image using the Build-From-S3 method with a 1-day retention policy. This approach creates a container image using the source files present in S3 and uses an EC2 instance as a medium to build and push the image to the desired ECR repository. Additionally, our analysis of the worst-case cost of replicating an image using different retention policies and invocation frequencies showed that the cheapest approach is to use an EC2 instance, especially when invoking the function call 3x per month and retaining the image for an hour. In conclusion, this paper has demonstrated cost vs. performance trade-offs for three alternate approaches for just-in-time container image replication to support global high availability of container-based serverless functions.

References

[1] M. Yan *et al.*, "Building a chatbot with serverless computing," pp. 1–4, 2016.

[2] S. Ginzburg and M. J. Freedman, "Serverless isn't server-less: Measuring and exploiting resource variability on cloud faas platforms," pp. 43–48, 2020.

[3] R. Cordingly, J. Kaur, D. Dwivedi, and W. Lloyd, "Towards serverless sky computing: An investigation on global workload distribution to mitigate carbon intensity, network latency, and cost," IEEE, pp. 59–69, 2023.

[4] D. Lambion *et al.*, "Characterizing x86 and arm serverless performance variation: A natural language processing case study," pp. 69–75, 2022.

[5] P. Patros *et al.*, "Toward sustainable serverless computing," *IEEE Internet Computing*, vol. 25, no. 6, pp. 42–50, 2021.

[6] M. Shahrad *et al.*, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," pp. 205–218, 2020.

[7] "Aws container based lambda." [Online]. Available: https://aws.amazon.com/blogs/aws/new-for-aws-lambda-container-image-support/

[8] "Ecr pricing." [Online]. Available: https://aws.amazon.com/ecr/pricing/

[9] J. M. Hellerstein *et al.*, "Serverless computing: One step forward, two steps back," *arXiv preprint arXiv:1812.03651*, 2018.

[10] "Aws crr." [Online]. Available: https://aws.amazon.com/about-aws/whats-new/2015/03/amazon-s3-introduces-cross-region-replication/

[11] "Tag level filtering in aws ecr crr." [Online]. Available: https://github.com/aws/containers-roadmap/issues/1737/

[12] "Repository level filtering in aws ecr crr." [Online]. Available: https://aws.amazon.com/blogs/containers/using-amazon-ecr-replication-rules-to-optimize-your-application-delivery-process/

[13] S. Quinn, R. Cordingly, and W. Lloyd, "Implications of alternative serverless application control flow methods," pp. 17–22, 2021.

[14] S. Ristov *et al.*, "Faascinating resilience for serverless function choreographies in federated clouds," *IEEE Trans on Network and Service Management*, vol. 19, no. 3, pp. 2440–2452, 2022.

[15] "Google cloud functions." [Online]. Available: https://cloud.google.com/functions/

[16] "Alibaba cloud functions." [Online]. Available: https://www.alibabacloud.com/en/product/function-compute/

[17] A. Wang *et al.*, "{FaaSNet}: Scalable and fast provisioning of custom serverless container runtimes at alibaba cloud function compute," pp. 443–457, 2021.

[18] W. Ling *et al.*, "Pigeon: A dynamic and efficient serverless and faas framework for private cloud," IEEE, pp. 1416–1421, 2019.

[19] U. Tos *et al.*, "A performance and profit oriented data replication strategy for cloud systems," IEEE, pp. 780–787, 2016.

[20] C. P. Smith *et al.*, "Fado: Faas functions and data orchestrator for multiple serverless edge-cloud clusters," IEEE, pp. 17–25, 2022.

[21] M. Brooker *et al.*, "On-demand container loading in {AWS} lambda," in *2023 USENIX Annual Tech Conf (USENIX ATC 23)*, 2023, pp. 315–328.

[22] "Docker save image." [Online]. Available: https://docs.docker.com/reference/cli/docker/image/save/