# Steel: Simplified Development and Deployment of Edge-Cloud Applications

Shadi A. Noghabi
*Univeristy of Illinois (UIUC)*
*abdolla2@illinois.edu*

John Kolb
*UC Berkeley*
*jkolb@berkeley.edu*

Peter Bodik, Eduardo Cuervo
*Microsoft Research*
*{peterb,cuervo}@microsoft.com*

## Abstract

The rapid growth in both the number and variety of cloud services has led to the emergence of complex applications using multiple cloud services (typically $> 5$). Although building cloud-based applications is relatively simple, extending them to an edge-cloud environment is complicated, error-prone, and time-consuming. Effectively using the edge requires dynamic adaptation and movement of services between edges and the cloud, e.g., in presence of failures or load spikes. However, current platforms do not support this. We propose Steel, a high-level abstraction that simplifies development, enables transparent adaptation and migration, and automates deployment and monitoring across the entire edge-cloud environment. Additionally, Steel enables modular and pluggable optimizations, such as placement, load balancing, and dynamic communication, with minimal effort. Based on our evaluation, we reduce the initial development effort (1.7x – 3.5x reduction in lines of config), support dynamic moves with minimal changes ($\sim$2 lines of config per move, reducing 95% of the overhead), and support easy development of optimizations (e.g., a placement optimizer requires $\sim$500 lines of code).

## 1 Introduction

Cloud services have seen a disruptive growth, becoming the de facto solution for building applications at large scale. Gartner Inc. estimates that cloud services will become a $383 Billion market by 2020, a 20% annual growth rate [11,15]. Simultaneously, there has been continuous growth in the number and variety of services provided by all major cloud providers. Azure, Amazon AWS, and Google currently have more than 170, 130, and 70 different services, respectively [6, 9, 12]. These services cover a variety of categories, e.g., compute, communication, storage, and analytics. There are multiple services within each category, each tailored for a specific use-case. For example, for analytics alone, Amazon has Kinesis (streaming), EMR (batch), and seven other services [6]. New categories also constantly emerge to cover new domains such as AI and the Internet of Things.

Because of the diverse services and large-scale processing capabilities of the cloud, most cloud applications use *multiple cloud services*, in various complex ways. We expect this complexity to grow as cloud services become increasingly diverse. One main category of these applications is *data processing* applications, where large quantities of data are piped from one service to another while being transformed at each point. This can be viewed as a Directed Acyclic Graph (DAG) of services. IoT applications, stream processing jobs, and image/video processing are typical examples of such applications, e.g., a simple monitoring and maintenance IoT application consists of more than eight different services (for computation, communication, and storage) [14].

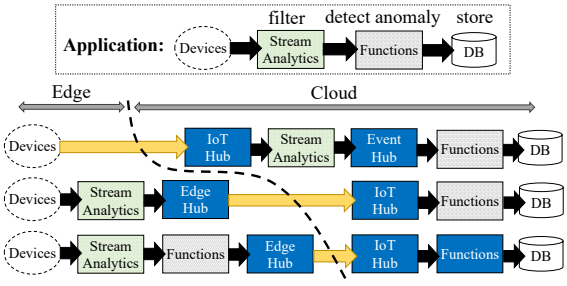



Figure 1: Sample anomaly detector IoT application.

In parallel, there is a growing interest in pushing computation to the edge. Processing on the edge has several advantages including: reducing end-to-end latency, providing continuous service despite intermittent connectivity to the cloud, reducing network bandwidth consumption, and reducing the monetary cost of using cloud services [20, 26, 30, 32, 36–39, 42]. This has led to cloud services offering solutions that can be ported to the edge, e.g., AWS Greengrass for edge computation, Azure Edge Hub for edge-cloud communication, or Azure Functions and AWS Lambda's extension to the edge [3, 5, 8].

When using the edge, many optimizations, that are usually optional in the cloud, become crucial, especially to make applications robust and correct. Examples include placement, load balancing, and adaptive communication [34]. Individual edge deployments and their networks are heterogeneous and diverse in terms of cost, performance, availability, etc., generating a large search space of potential placements. Edges also have limited resources (for economic and practical reasons) while workloads are dynamic, and failures are common at the edge. Thus, the only cost-effective, peak-tolerant option is to adaptively balance and dynamically move services. Furthermore, the connection from the edge is poor, costly, and intermittent, demanding dynamic batching and compression [27, 35, 44]. Towards these optimizations, it is essential to be able to *readily move* services back and forth between edges/cloud and *dynamically adapt* to the environment.

Even though the construction of applications from cloud-only services has become well-established and relatively straightforward, the industry is still in its infancy in terms of *supporting multi-service applications that are dynamic and movable across the full edge-cloud environ-*

*ment*. There are several challenges in the development, deployment, monitoring, and optimization of these applications. First, configuring and connecting various cloud services is cumbersome and error-prone. Deployment through a web portal is time-consuming ($\sim$25 minutes for a simple four-service pipeline) and not repeatable. Building automated scripts requires specifying several configurations (typically 100s of lines of config), and in many cases, these configurations are not properly documented or maintained over time. The configurations also change depending on the location of the service (edge or cloud). Although this overhead can be acceptable as a one-time effort, the conventional case for cloud-only applications, it becomes a major barrier to dynamic movement of services between the cloud and the edge.

Furthermore, most cloud services have each developed organically and independently. Thus, they exhibit wide diversity and suffer from inter-service compatibility constraints, e.g., Azure Stream Analytics cannot connect directly to Azure Functions. These compatibility issues can be resolved by adding intermediate *glue services*. However, the glue services depend on the location of services (edge or cloud). Figure 1 shows a simple anomaly detector application built using Azure services. This application reads sensor data, filters values (Stream Analytics), detects anomalies (Functions), and writes the results to a database (Document DB). Depending on the location of the different services, the glue services required (shown in blue) change substantially. Manually configuring these glue services to support migration is intractable because it is error-prone and cannot be done in real time. Although the example above is based on Azure, it is a fundamental problem for other platforms as well.

Finally, the complexities and dynamism of edges require applying many optimizations, common across different applications. Ideally, these should be pluggable features. Currently however, application developers have to manually incorporate these optimizations, without the ability to reuse them. Besides, many of these optimizations rely on end-to-end monitoring across services, e.g., their performance, cost, and availability, which is not unified or consistent across the many diverse services.

To bridge this gap and ease the burden of manual, redundant, and error-prone multi-service application development, we argue for a *high-level abstraction*. This abstraction should support dynamically adapting and moving services (between edges and cloud) along with the ability to *build pluggable, reusable optimization modules*. The main challenge is finding the right level of abstraction: it should hide the complexities of building applications without negatively impacting their flexibility (i.e., users should be able to build the same applications they could build before). To reach this sweet spot, we argue that we only need to abstract the *connections* and *locations* of services. We should not impose any limitations on the internals of each service, e.g., using a SQL query to build a streaming job.

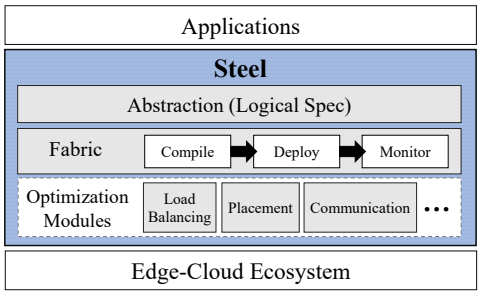



Figure 2: High level architecture of Steel, a middle layer between the applications and edge-cloud environment.

In this work, we propose Steel, a high-level abstraction and deployment engine tailored specifically for building multi-service applications in the emerging edge-cloud environment. Steel abstracts the logical connection of services in an application. Based on the location of services (edge or cloud), it automatically resolves compatibility constraints by adding required glue components, builds the detailed deployment scripts/templates, and deploys the application across the entire environment. Steel provides full flexibility to dynamically adapt and migrate services while handling any internal complexities. In addition, Steel provides a framework to build modular, pluggable and reusable optimizations.

We implemented a prototype version of Steel on top of Azure, one of the leading cloud service providers. We evaluate our solution with a number of real-world applications. Based on our results, we reduce the number of lines of config (*loc*) for the initial application development by 1.7x–4.8x, and in presence of a change or move, we reduce the changes to $\sim$2 *loc* per move, a $>$95% reduction. We also show how our abstraction is essential for simplifying and building pluggable optimizations. As an example, we develop a placement optimizer as a pluggable module with $\sim$500 lines of code.

## 2 Design of Steel

Steel targets data processing applications constructed of multiple cloud services, connected to each other in form of a DAG. Manually developing, deploying, and optimizing these multi-service applications is hard, time-consuming and error-prone. For example, deploying the anomaly detector application (Figure 1) through the web portal takes roughly 25 minutes for an expert, and building a deployment template requires $>$250 lines of config.

Listing 1 shows a sample Azure template required just for deploying (without fully connecting) a Document DB service. The template requires a verbose specification of the service's properties, such as its detailed location and unique name, on lines 1–16. Lines 17–27 specify the output values that should be passed back to the user after the service is deployed and cannot be generated beforehand,

Listing 1: "Sample Azure Config for deploying a simple Document DB service"

```
{  "$schema": ``http://schema.management.azure.com/...'',
   ``contentVersion'': ``1.0.0.0'',
   ``resources'': [{
      ``apiVersion'': ``2015-04-08'',
      ``kind'': ``GlobalDocumentDB'',
      ``type'': ``Microsoft.DocumentDb/databaseAccounts'',
      ``name'': ``test-name'',
      ``location'': ``westus'',
      ``properties'': {
         ``databaseAccountOfferType'': ``Standard'',
         ``locations'': [
         {``id'': ``test-id'',
            ``failoverPriority'': 0,
            ``locationName'': ``West Us''}]},
      ``tags'': {``defaultExperience'': ``DocumentDB''}
   }],
   ``outputs'': {
      ``EndpointUri'': {
         ``value'': ``[reference('...').documentEndpoint]'',
         ``type'': ``string''},
      ``PrimaryKey'': {``value'': ``[listKeys(...)']',
                       ``type'': ``string''},
      ``resourceId'': {
         "value": ``[concat('subscriptions/',
               subscription().subscriptionId,'...')]'',
         ``type'': ``string''}
}  }
```

e.g., the unique ID and URI that must be known to applications and services that will connect to the database.

Steel's goal is to simplify a) developing multi-service applications, b) deploying them across the entire environment, c) dynamically adapting and moving parts of them, and d) applying common optimizations to them. Steel acts as middle layer between application developers and the edge-cloud, hiding the underlying complexities. As shown in Figure 2, Steel consists of three main layers: abstraction, fabric, and optimization modules.

## 2.1 Abstraction

The abstraction includes: a) the *logical DAG* of the application, i.e., the main services and their connections excluding the glue services and b) the *location* of each service. We argue this information is sufficient to support adaptable and movable services across the entire environment. Listing 2 shows the anomaly detector (Figure 1) in this abstraction. As shown, it provides full flexibility and extensibility of the internals of each service. Services can be defined similar to using the typical templates, e.g., using a SQL query as a streaming job or setting service specific configs (e.g., "Streaming Units"). To move a service, simply, the location mapping needs to be updated.

## 2.2 Fabric

The Fabric materializes the abstraction of an application into an actual physical deployment. The Fabric *compiles* the abstraction, *deploys* it across the edge-cloud environment, and *monitors* it end-to-end.

**Compiler:** The compiler converts the abstraction to detailed deployment templates, based on the location of each service. First, it verifies the location settings are valid. Some services cannot move to the edge or cloud, e.g, a permanent storage on the edge, and there can be user-specified constrains, e.g., service A and B should be placed together. Second, glue services are added to fix the compatibility constraints across services. Figure 1 shows a simple 3-service application, and how the glue components (shown in blue) change based on the location settings. Finally, the routes and connections are configured to connect services, including glue services, and the actual deployable templates are built. These template differ based on whether they are targeted for the edge or the cloud, where the compiler masks these complexities.

Listing 2: "Sample Application in Steel's Abstraction"

```
{  "Name": "test job",
   "Services": [
   {"Name": "filter",
      "Type": "AsaJob",
      "Query": "SELECT * FROM input0 WHERE tmp > 60"},
      "Streaming Units": 10 }
    ...
   ],
   "Connections": [
   {"From": "filter", "To": "anomaly detector"},
   {"From": "anomaly detector", "To": "anomaly db"} ],
   "Locations": {
      "filter": {"Type": "edge","Id": "dev1"},
      "anomaly detector": {"Type": "cloud", "Id": "westus"},
      "anomaly db": {"Type": "cloud","Id": "westus"}
}  }
```

**Deployer:** The deployer deploys the templates across the entire environment. Using the application's DAG of services, it automatically detects dependencies among services, and deploys them stage-by-stage, with multiple parallel deployments within each stage. For example, an Functions instance connected to a Document DB needs to know the endpoint URI of the DB service (which is only available once the DB has been deployed), and therefore, must be instantiated in a stage after the DB.

**Monitoring:** Similar to cloud-based tools [1, 2, 7, 16, 31], the monitoring gathers real-time metrics for performance, resource usage, and cost, but across the entire edge-cloud environment. It digests and unifies the metrics in an end-to-end application level, rather than per service. The monitoring hides the diversity of cloud services (with their own metrics and models), and heterogeneity of the edge devices. For example, it masks pricing structures diversities such as the different metrics (bytes, CPU, IO), granularities, reserve costs, etc. Monitoring is crucial for checking the health of the program, and also a major piece for further analysis such as for bottleneck detection, allocation and placement optimizations, load balancing, adaptive communication, etc.

## 2.3 Optimization Modules

Inherently, the edge comes with many complexities such as increased chances/causes of failures, poor and variable network links, extreme heterogeneity among resources,

and unbalanced distribution of workloads over space and time. These complexities rise the need for additional, but common, optimizations. Steel's design enables the modular, pluggable, and extensible implementation of these optimization which manipulate the logical DAG representation (Section 2.1). Below, we list a few of the common optimizations, and how they benefit from Steel's Fabric, abstraction, and design.

**Placement:** At the edge, the search space of potential placements grows significantly. Edges are heterogeneous (from Raspberry Pi to multi-GPU machines), large in number ($\sim$ 1000s in a typical factory), and with diverse network bandwidths. The optimal placement depends on the workload, available capacity, cost of bandwidth, etc. Manually deciding the placement is tedious. In presence of a change (e.g., a new application), the procedure has to be repeated. Thus, a common optimization is automated placement. Decoupling an application and its location, along with the end-to-end monitoring provided by Steel, is essential to make placement automated and smart.

**Load balancing:** Edges decentralize computation across a spatial area, where ideally each edge processes workloads closest to it. However, in practice, workloads and edges are not uniformly distributed across the area, e.g., a cluster of sensors close to one edge, machines and links fail, and the load changes over time. Since edges do not have the luxury of gathering and distributing data in a centralized location, as in the cloud, they need to dynamically divert load and move services to other locations (or bring them back). The ability to transparently and easily move services provided by Steel, is an important feature for load balancing.

**Communication:** Edges are often characterized by limited bandwidth or intermittent connectivity to the larger Internet, making the network a critical factor for performance. Also, both the network and the entry point at the cloud have a high cost, e.g., charging per message/Byte. To this end, it is common to optimize the communication by using smart batching, compression, and compaction. Steel's connection abstraction enables these optimizations to be done behind the scene, in a pluggable and configurable manner, and its unified monitoring enables intelligent optimizations for both cost and performance.

# 3 Implementation

The main challenge in building Steel is extending systems originally designed for static configurations to dynamic and automated ones. We implemented a prototype of Steel on top of Azure, a leading cloud service provider. To enable edge-cloud communication, we use the Edge Hub [5] on the edge and IoT Hub [4] in the cloud (as an entry point). Services deployed on the edge talk with the Edge Hub, which in turn talks to the IoT Hub, that then routes data to other cloud services. We use the IoT Hub because of its ease of use. However, other entry points, such as a publish-subscribe queue, can be used as well.

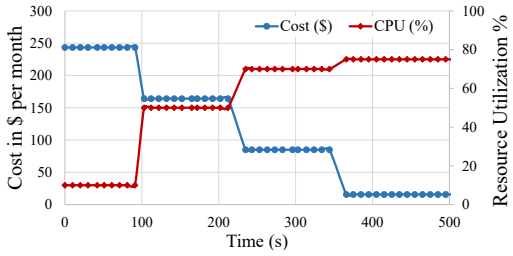


Figure 3: Over time reduction of price and improvement of edge resource utilization via the placement optimizer.

The prototype compiles an application (in Steel's abstraction) to edge and cloud templates (based on locations), and in parallel deploys them across the entire environment. At the edge, for portability, we use Docker [10] and containerized versions of the services (supported by the services [8]). Next, the prototype sets the routes in the Edge Hub and IoT Hub for the data to flow through accordingly. Finally, it starts monitoring and publishing metrics to a unified source. Metrics are gathered from multiple sources: existent service metrics, resource metrics, and our additional instrumented metrics.

## 3.1 Prototype Placement Optimizer

To demonstrate Steel's extensibility at adding optimizations, we implemented a basic placement optimizer. The optimizer automatically decides service placements with the goal of a) reducing the cost of running services in the cloud, and b) increasing the resource utilization at the edge. The optimizer starts with a config where all services are deployed in the cloud. Then, in a greedy fashion, it finds the most expensive services on the boundary of the edge and the cloud. It tries moving these services; ensures the moves do not hurt performance; and continues this loop until the edge capacities are full.

We implemented this optimizer on top of the Fabric, as a pluggable module, with only $\sim$500 lines of C# code. Building the optimizer with such low development effort was only possible due to the movable abstraction and readily available monitoring provided by Steel. Figure 3 shows a sample run of a multi-stage IoT application. At each iteration, the overall cost is reduced as services move from the cloud to the edge. Concurrently, the edge's resource utilization increases from 10% to 75%.

# 4 Experimental Evaluation

The main promise of Steel is ease of development, and adaption of services. To compare our abstraction (*Steel*) to conventional templates/scripts currently required by Azure to deploy applications (*current*), we implement real world applications in both systems on top of Azure. We measure the development effort in terms of lines of config (*loc*), i.e., number of configs required by Azure/Steel. We evaluate both the initial development and the changes required for moving services.

We chose 6 diverse applications from a combination

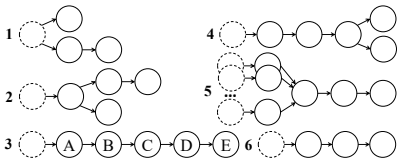



Figure 4: DAG of studied applications. Dotted and full circles show the data source and services, respectively.

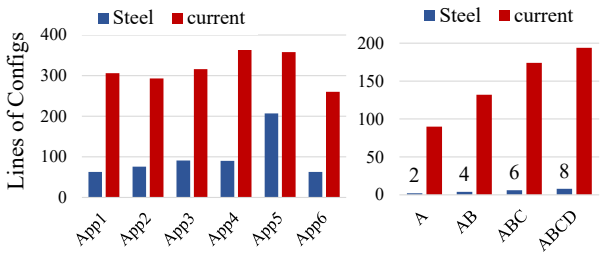
of common and pre-configured applications [17]. Figure 4 shows the logical DAG (without glue services) of the applications: 1. a simple data persisting application (two stores, raw and filtered), 2. a predictive maintenance application using machine learning, 3. Bluetooth sensor connector and analyzer (convert format, add meta-data, filter, average, and store), 4. a factory remote monitoring and alert generator, 5. a campus-wide statistics generator (aggregate locally, join globally, build statistics, and store), and 6. an anomaly detector (Figure 1)

**Initial Development** We compare the initial development effort in terms of *loc* in both Steel and the current cloud environments. Figure 5-a shows the results for the 6 chosen applications. Our abstraction reduces the *loc* between 1.7x to 4.8x across the different applications compared to the current system. This improvement is due to two main factors: 1. the hidden glue services added automatically by the system, and 2. the substantially reduced configurations ($\approx$ 1–2 *loc*) required for connecting services. Our abstraction works best when there are multiple services per location. For examples, Application 5 is spread across many edges, each edge performing one service and the cloud joining the results (followed by additional computation). Since the edge part is simple (single stage), there is little improvement there, and the improvement mostly comes from the cloud part, resulting in an overall smaller improvement (1.7x).

**Modifications** Being able to dynamically move services is a crucial feature when going to the edge. We evaluate the changes required when making a move. We start with an all-cloud deployment of Application 3 (the longest chain), and move its services {A,B,C,D} one by one to the edge. We measure the *loc* changes (add, modify, or delete) required at each step, compared to the all-cloud deployment. As shown in Figure 5-b, Steel provides a constant and small over head of around 2 *loc* per change of a service location, as only the location map needs to be updated (in Figure 2.1). However, current systems need 100s of *loc* changes, between 260–360 *loc* for Application 3, both to add glue components and to reconnect the services. Steel reduces over 95% of this overhead.

## 5 Related Work

**Abstractions:** Mobile fog [29], Beam [41], and [13] propose abstractions to hide the complexities and heterogeneity of the edge. General purpose abstractions [21], have also emerged for building large-scale, elastic, and reliable applications. However, none integrate with existing cloud services (despite the need for doing so [18]).




(a) Initial Development (b) Movement of App3

Figure 5: Comparison of lines of config required in Steel and current systems for a) developing applications, and b) moving a service {A,B,C,D} within an application.

**Edge-Cloud Frameworks:** To leverage the edge, many end-to-end edge-cloud frameworks have emerged, e.g., FarmBeats for agricultural IoT, Race, and GigaSight [22, 40, 43]. Although these systems handle abstraction, deployment, and optimizations, they are each tailored for a specific usecase (e.g, remote agriculture fields), and do not generalize to custom data-processing applications.

**Edge Deployment:** Dynamic and inter-operable deployment across heterogeneous devices is essential for the edge. Micro-cloud, Cloudlets, Openstack++, and others [13, 19, 25, 28, 30, 33, 38] have proposed portable and dynamic infrastructures using either containers or virtual machines. However, this is orthogonal to the need of simplifying the development of edge-cloud applications.

**Migration and Placement:** Dynamically placing and moving computation is not a new concept. Maui and CloneCloud [23, 24] have explored this with mobiles as the edge. MigCEP [35], VM Handoff [27], and [44] optimize the dynamic migration between edges, based on different constraints, e.g., latency. While these approaches can be added as optimizations modules to Steel, they do not support current cloud services.

## 6 Conclusion

This paper describes Steel, a high-level framework designed specifically for building complex data processing applications in the emerging edge-cloud environment. We design Steel to hide the complexities of developing, deploying, and monitoring data processing applications using many cloud services, and to support dynamically adapting and easily moving services back and forth between the edge and the cloud. Steel is an extensible framework where common but crucial optimizations for the edge (e.g., placement, load balancing, communication) can be built as pluggable and configurable modules.

As part of future work, we plan to further investigate optimizations and requirements in an edge-cloud environment, including supporting security/privacy, and develop efficient edge-oriented solutions as pluggable modules in Steel. Moreover, we are working on extending the abstraction to incorporate both resource allocations, ideally being set automatically by Steel, and service level agreements for building better optimizations.

## References


[1] Amazon CloudWatch. https://aws.amazon.com/cloudwatch/. Accessed: 08/2017.

[2] Applciation Insights. https://azure.microsoft.com/en-us/services/application-insights/. Accessed: 08/2017.

[3] AWS Greengrass. https://aws.amazon.com/greengrass/. Accessed: 08/2017.

[4] Azure Internet of Things. https://azure.microsoft.com/en-us/suites/iot-suite/. Accessed: 02/2018.

[5] Azure IoT Edge. https://azure.microsoft.com/en-us/services/iot-edge/. Accessed: 08/2017.

[6] Cloud Products & Services - Amazon Web Services (AWS). https://aws.amazon.com/products/. Accessed: 02/2018.

[7] Datadog – Modern monitoring & analytics. https://www.datadoghq.com/. Accessed: 08/2017.

[8] Deploy Azure Function as an IoT Edge module - preview. https://docs.microsoft.com/en-us/azure/iot-edge/tutorial-deploy-function. Accessed: 08/2017.

[9] Directory of Azure Cloud Services. https://azure.microsoft.com/en-us/services/. Accessed: 02/2018.

[10] Docker. https://www.docker.com/. Accessed: 08/2017.

[11] Gartner Forecasts Worldwide Public Cloud Services Revenue to Reach $260 Billion in 2017. https://www.gartner.com/newsroom/id/3815165. Accessed: 02/2018.

[12] Google Cloud Platform – Products & Services. https://cloud.google.com/products/. Accessed: 02/2018.

[13] Multi-access Edge Computing. http://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing. Accessed: 08/2017.

[14] Predictive maintenance preconfigured solution walkthrough. https://docs.microsoft.com/en-us/azure/iot-suite/iot-suite-predictive-walkthrough. Accessed: 08/2017.

[15] Roundup Of Cloud Computing Forecasts, 2017. https://www.forbes.com/sites/louiscolumbus/2017/04/29/roundup-of-cloud-computing-forecasts-2017/. Accessed: 02/2018.

[16] Stackdriver Monitoring. https://cloud.google.com/monitoring/. Accessed: 08/2017.

[17] What is Azure IoT Suite. https://docs.microsoft.com/en-us/azure/iot-suite/iot-suite-what-are-preconfigured-solutions. Accessed: 08/2017.

[18] BAHL, P., HAN, R. Y., LI, L. E., AND SATYANARAYANAN, M. Advancing the state of mobile cloud computing. In *Proceedings of the third ACM workshop on Mobile cloud computing and services* (2012), ACM, pp. 21–28.

[19] BHARDWAJ, K., SREEPATHY, S., GAVRILOVSKA, A., AND SCHWAN, K. Ecc: Edge cloud composites. In *Proceedings of the International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)* (2014), IEEE.

[20] BONOMI, F., MILITO, R., ZHU, J., AND ADDEPALLI, S. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing* (2012), ACM.

[21] BYKOV, S., GELLER, A., KLIOT, G., LARUS, J. R., PANDYA, R., AND THELIN, J. Orleans: cloud computing for everyone. In *Proceedings of the Symposium on Cloud Computing (SoCC)* (2011), ACM.

[22] CHANDRAMOULI, B., CLAESSENS, J., NATH, S., SANTOS, I., AND ZHOU, W. Race: Real-time applications over cloud-edge. In *Proceedings of the International Conference on Management of Data SIGMOD)* (2012), ACM.

[23] CHUN, B.-G., IHM, S., MANIATIS, P., NAIK, M., AND PATTI, A. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the conference on Computer systems* (2011), ACM.

[24] CUERVO, E., BALASUBRAMANIAN, A., CHO, D.-K., WOLMAN, A., SAROIU, S., CHANDRA, R., AND BAHL, P. Maui: making smartphones last longer with code offload. In *Proceedings of the international conference on Mobile systems, applications, and services (MobiSys)* (2010), ACM.

[25] ELKHATIB, Y., PORTER, B., RIBEIRO, H. B., ZHANI, M. F., QADIR, J., AND RIVIÈRE, E. On using micro-clouds to deliver the fog. *Internet Computing 21*, 2 (2017).

[26] GARCIA LOPEZ, P., MONTRESOR, A., EPEMA, D., DATTA, A., HIGASHINO, T., IAMNITCHI, A., BARCELLOS, M., FELBER, P., AND RIVIERE, E. Edge-centric computing: Vision and challenges. *SIGCOMM Computer Communication Review 45*, 5 (2015).

[27] HA, K., ABE, Y., CHEN, Z., HU, W., AMOS, B., PILLAI, P., AND SATYANARAYANAN, M. Adaptive vm handoff across cloudlets. *Technical Report CMU-CS-15–113, CMU School of Computer Science* (2015).

[28] HA, K., AND SATYANARAYANAN, M. Openstack++ for cloudlet deployment. *School of Computer Science Carnegie Mellon University Pittsburgh* (2015).

[29] HONG, K., LILLETHUN, D., RAMACHANDRAN, U., OTTENWÄLDER, B., AND KOLDEHOFE, B. Mobile fog: A programming model for large-scale applications on the internet of things. In *Proceedings of the SIGCOMM workshop on Mobile cloud computing* (2013), ACM.

[30] HU, Y. C., PATEL, M., SABELLA, D., SPRECHER, N., AND YOUNG, V. Mobile edge computing—a key technology towards 5g. *ETSI white paper 11*, 11 (2015).

[31] KALDOR, J., MACE, J., BEJDA, M., GAO, E., KUROPATWA, W., O'NEILL, J., ONG, K. W., SCHALLER, B., SHAN, P., VISCOMI, B., ET AL. Canopy: An end-to-end performance tracing and analysis system. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)* (2017), ACM.

[32] KALIM, F., NOGHABI, S. A., AND VERMA, S. To edge or not to edge? In *Proceedings of the Symposium on Cloud Computing (SoCC)* (2017), ACM.

[33] LEE, E. A., HARTMANN, B., KUBIATOWICZ, J., ROSING, T. S., WAWRZYNEK, J., WESSEL, D., RABAEY, J., PISTER, K., SANGIOVANNI-VINCENTELLI, A., SESHIA, S. A., ET AL. The swarm at the edge of the cloud. *Design & Test 31*, 3 (2014).

[34] NOGHABI, S. A., KOLB, J., BODIK, P., AND CUERVO, E. Unified management and optimization of edge-cloud iot applications. *arXiv preprint arXiv:1805.02305* (2018).

[35] OTTENWÄLDER, B., KOLDEHOFE, B., ROTHERMEL, K., AND RAMACHANDRAN, U. Migcep: operator migration for mobility driven distributed complex event processing. In *Proceedings of the international conference on Distributed event-based systems* (2013), ACM.

[36] RYDEN, M., OH, K., CHANDRA, A., AND WEISSMAN, J. Nebula: Distributed edge cloud for data intensive computing. In *Proceedings of the International Conference on Cloud Engineering (IC2E)* (2014), IEEE.

[37] SATYANARAYANAN, M. The emergence of edge computing. *Computer 50*, 1 (2017).

[38] SATYANARAYANAN, M., BAHL, P., CACERES, R., AND DAVIES, N. The case for vm-based cloudlets in mobile computing. *Pervasive Computing 8*, 4 (2009).

[39] SATYANARAYANAN, M., LEWIS, G., MORRIS, E., SIMANTA, S., BOLENG, J., AND HA, K. The role of cloudlets in hostile environments. *Pervasive Computing 12*, 4 (2013).

[40] SATYANARAYANAN, M., SIMOENS, P., XIAO, Y., PILLAI, P., CHEN, Z., HA, K., HU, W., AND AMOS, B. Edge analytics in the internet of things. *Pervasive Computing 14*, 2 (2015).

[41] SHEN, C., SINGH, R. P., PHANISHAYEE, A., KANSAL, A., AND MAHAJAN, R. Beam: Ending monolithic applications for connected devices. In *Proceedings of the Annual Technical Conference (ATC)* (2016), USENIX, pp. 143–157.

[42] SHI, W., CAO, J., ZHANG, Q., LI, Y., AND XU, L. Edge computing: Vision and challenges. *Internet of Things Journal 3*, 5 (2016).

[43] VASISHT, D., KAPETANOVIC, Z., WON, J., JIN, X., CHANDRA, R., SINHA, S. N., KAPOOR, A., SUDARSHAN, M., AND STRATMAN, S. FarmBeats: An iot platform for data-driven agriculture. In *Proceedings of the Symposium on Networked Systems Design and Implementation (NSDI)* (2017), USENIX.

[44] WANG, S., URGAONKAR, R., ZAFER, M., HE, T., CHAN, K., AND LEUNG, K. K. Dynamic service migration in mobile edge-clouds. In *Proceedings of the IFIP Networking Conference (IFIP Networking)* (2015), IEEE.