SERVERLESS COMPUTING: STATE-OF-THE-ART

Tomy GUICHARD
ISEN Yncréa Ouest
email: tomy.guichard@isen-ouest.yncrea.fr

ABSTRACT
Serverless computing is a relatively new popular cloud computing model to run applications in the cloud. It has a few advantages such as cost reduction and automatic scaling. However, the design of these serverless applications is dramatically impacted, and existing applications may need to be redesigned so that it can run under a serverless model. It is important to note that the term "serverless" does not imply that no servers are involved, it only means application developers do not need to provision and manage servers.

The aim of this paper is to provide a better understanding of the design challenges of serverless applications, as well as an overview of the existing commercial and non-commercial solutions to run serverless workloads.

KEY WORDS
Serverless, Cloud computing

1 Introduction

Cloud computing is now progressively replacing traditional data centers [1], as companies delegate the management of their infrastructures to cloud providers such as Google, Amazon and Microsoft.

Instead of managing their infrastructures and having to deal with all its associated costs (hardware purchases, server cooling, maintenance, security, etc.), companies can now choose to run their applications in the cloud, with different levels of control over an infrastructure hosted by their cloud providers. These levels of controls are associated to different commercial products called "cloud services". There currently exists 4 different services [2], as described in the figure 1:

- Infrastructure as a Service (IaaS): This service allows companies to rent servers, either virtual machines (VM) or bare-metal servers. They usually instruct the provider to install a specific OS on the server and then manage the server through an SSH connection or a remote desktop connection.
- Platform as a Service (PaaS): This service allows companies to deploy their application without having to manage servers. The application is automatically scaled and load balanced by the cloud provider [3].
- Serverless: Companies develop an app client (Android application, web application, etc.). This client communicates with an app backend. The app backend is either entirely managed by the cloud provider, or the company's custom software can run as the app backend. The cloud provider creates the necessary resources (servers) when the app backend is called. When the backend is not currently in use, no compute resources are allocated to it so that the company is not billed when their backend is not used.
- Software as a Service (SaaS): Companies pay to use an application that is entirely developed, deployed and managed by the provider. Examples of SaaS include Dropbox, Slack and GitLab.

Companies that develop their own applications based on cloud services will usually have to choose between IaaS, PaaS and Serverless. The choice of service is usually done in early stages of new projects as it directly impacts the design of the application. Serverless is a service that is particularly adapted for developing stateless event-driven applications [4], as we will see in the rest of this paper.

In this paper we first introduce serverless computing and its two most common implementations. In section 3, a state-of-the-art will detail existing commercial products and open source software to run serverless workloads. A solution to run serverless workload will be detailed in section 4 and we will conclude the paper by discussing the advantages and limitations of serverless computing.

2 Overview of Serverless computing

2.1 Definition

The goal of serverless computing is to allow developers to think about operating systems instead of operating servers [5]. In serverless computing, applications are considered as workflows, distributed logic and externally managed data stores [5]. Developers no longer need to manage infrastructure, as the cloud service provider automatically provisions, scales, and manages the infrastructure required to run the code [6].

When their application is deployed, developers are only charged for the resources they use. Because of the autoscaling abilities of serverless, no resources are necessary when the application is not used. This is usually called scaling to zero. If the application is heavily used, the resources will be scaled appropriately so that it can handle the high load.

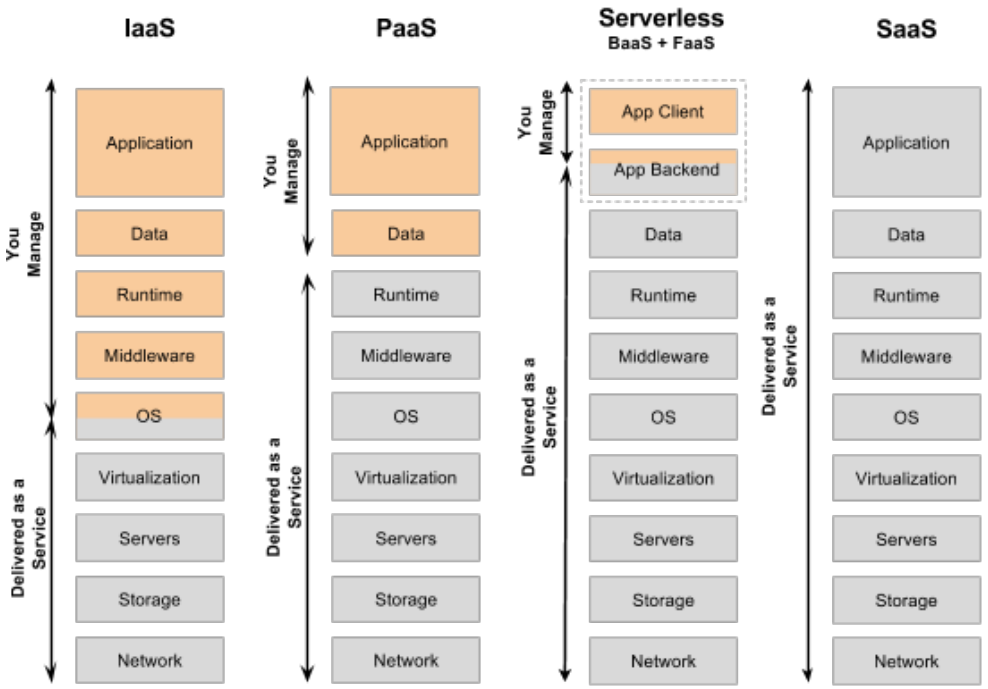

Figure 1. Cloud services

Source: specify.io

As a result, serverless computing is particularly suitable for event driven applications. Event driven applications are programs specially designed to respond to actions generated by users or systems [7]. Examples of events can be the creation of a new user, or a file added in a storage bucket. The flow of incoming events is usually not constant, so costs can fluctuate accordingly. For applications with very few events, serverless computing is even more suitable as costs will be as low as a few cents per month [8].

2.2 Functions as a Service (FaaS)

FaaS is a serverless environment used for running software [5]. Developers need to push their code to the serverless platform offered by their cloud provider. The process of creating a new cloud function is usually done via an easy-to-use web graphical user interface (GUI), a command line interface (CLI) or a REST application programming interface (API), depending on the cloud provider.

Because the code is only invoked when the function is called, it must be stateless. However, the function can still query external databases, managed by the same or another cloud provider, to persist data or fetch existing data.

Once created, there are usually multiple ways of invoking these functions, depending on the cloud provider:

- By calling a specific HTTP endpoint dedicated to the cloud function. The function can be called via HTTP by another cloud function, or simply by a web client.
- By defining a cron schedule to call the function automatically on a regular basis. This is useful when a function performs some backups.
- By configuring the cloud provider's platform to call the function on specific events (e.g., creation of a new file in a storage bucket).
- By calling it manually using the cloud provider's platform interface. This is usually done to test and debug the function.

The function is often limited to a few minutes of execution. If the function exceeds the maximum time of execution, it is automatically killed by the platform. As a result, heavy tasks such as processing videos should be avoided.

Functions can also be limited to some programming languages. This depends on the runtimes available on the serverless platform. This is usually not a problem as most popular languages such as Java, Python and PHP runtimes are almost always available. There is however one way for platforms to circumvent the runtime limitations while also keeping isolation: by running Docker containers that contain the required environment to run the code. The required runtime and software dependencies are packaged in a Docker image by the developers so that the cloud provider only needs this image, instead of code, to run the function.

2.3 Backend as a Service (BaaS)

BaaS is a serverless environment entirely managed by the cloud provider. It allows developers to focus entirely on the frontend application as the backend is already available and ready to use. The development of the backend is usually what requires the most time and effort. Not having to develop the backend significantly reduces time to market. In some cases, it is possible to use BaaS in combination with a custom developed backend (hosted on a FaaS or IaaS platform).

Here is a list of common features available in most backend by service providers [9]:

- Data management
- APIs (REST and GraphQL)
- File storage
- Databases
- Email notifications
- Push notifications
- Login authentication
- Social media integration
- Infrastructure
- Software usage analytics

Developers only pay for resources they use, making BaaS very cost efficient for small applications.

3 State-of-the-art

In this section, we present the state-of-the-art implementations of serverless computing. We first describe the key technologies currently used for serverless computing. Then, we survey the popular commercial and open-source serverless computing products.

3.1 Key technologies

This section provides a review of technologies used in serverless computing environments.

3.1.1 Hypertext transfer protocol (HTTP)

HTTP is a stateless protocol used to transfer files such as HTML pages and media files. It uses a client-server model. It is mostly used in web browsers to load web pages and web assets (images, scripts, etc.). It is also used in service-to-service communications, using dedicated data exchange formats such as JSON or XML and software architectures such as REST to communicate. Some remote procedure call frameworks such as gRPC are based on HTTP.

Serverless services mostly use HTTP to communicate with the external world. They can be accessed synchronously using a script in a web browser via AJAX (Asynchronous JavaScript and XML), or from mobile application clients. This allows decoupling the client and the infrastructure as the client does not need to know how the backend infrastructure works. The backend infrastructure will automatically adapt itself to the incoming requests by scaling up or down the servers so that clients always get a response.

3.1.2 Docker containers

Docker is a set of PaaS products that use OS-level virtualization to deliver software in packages called containers [10]. Containers allow running multiple isolated Linux systems (containers) on a control host using a single Linux kernel. Containers can be orchestrated by platforms such as Kubernetes and Docker Swarm. These platforms automate the placement of containers on a fleet of hosts.

Serverless makes heavy use of containers as this is currently one of the best and most secure method for lightweight virtualization [11]. This allows running multiple isolated serverless workload on the same host to optimize the usage of servers. Containers give a lot of flexibility to both developers and platform operators as they contain an entire environment with all dependencies and tools to run software.

3.1.3 Publish/Subscribe

A publish/subscribe (pub/sub) system connects together information providers and consumers by delivering events from sources to interested users [12]. This is a form of asynchronous service-to-service communication used in serverless and microservices architectures. In a pub/sub model, any message published to a topic is immediately received by all of the subscribers to the topic. Pub/sub messaging can be used to enable event-driven architectures, or to decouple applications in order to increase performance, reliability and scalability [13].

3.2 Commercial products

This section provides a review of existing commercial products to run serverless applications in the cloud.

3.2.1 Google Cloud Run (GCR)

GCR is a serverless platform based on Knative to deploy containers [14]. It is generally available on Google Cloud Platform (GCP) since 2019 [15]. Using containers gives developers a high level of freedom as any programming language can be used. Using containers also makes GCR compatible with a lot of use-cases, such as hosting microservices, making it more than a FaaS platform. Even though any programming language can be used, there are a

still a few requirements, called container runtime contract, to make containers work with GCR [16]:

- Executables in the container image must be compiled for Linux 64-bit. Cloud Run specifically supports the Linux x86_64 ABI format.
- The container must listen for requests on 0.0.0.0 on the port to which requests are sent. By default, the port 8080 is used.
- The container should not implement any transport layer security (TLS) directly. Security is already handled by Cloud Run.
- Container instances must send a response within the time specified in the request timeout setting after it receives a request, including the container instance startup time. Otherwise, the request is ended and a 504 error is returned.
- Computations must be done during the processing a request as containers are automatically stopped when no requests are coming.
- Because containers are regularly started and stopped, and scaling is done horizontally, a state must not be maintained in the container. External databases and APIs should be used for this purpose.

GCR allows the configuration of a minimum number of instances so that the service never scales to zero [17]. Setting the minimum number of instances above zero removes some advantages of using serverless computing, but it can be useful in some situations when calls to the service is expected very frequently.

Scaling to zero also brings new issues such as cold starts. A cold start is the scaling from zero container to one container due to a new incoming request. As requests cannot be processed when there is no container running, this first request will have to wait additional time as a new container instance goes up. To avoid high latencies when processing such requests, containers should start fast and avoid loading too many components at startup.

Services can be configured to be accessed publicly, or only privately using bearer tokens. They can be triggered using HTTPS, gRPC, WebSocket, scheduled tasks and more [18].

The process of deploying and configuring new GCR services is usually done via the GCP console or using the gcloud command line tool.

3.2.2 Google Cloud Functions (GCF)

GCF is a FaaS platform available since 2016 [19] that supports Node.js, Python, Go, Java, .NET, Ruby and PHP [20]. Cloud functions can be managed using the GCP console, the gcloud command line tool or specific client libraries designed to consume GCP's API [21]. Internally, it uses the Functions Framework that is designed to create runnable stateless container [22]. Container images being portable, it allows the created functions to run in many environments, including other Knative-based environments such as Google Cloud Run. Because GCF uses containers internally, the first call of the functions will require a container cold start, making it a lot slower that successive calls.

There are two types of cloud functions [23]:

- HTTP functions, invoked from standard HTTP requests.
- Event-driven functions, used to handle events from the cloud infrastructure, such as messages on a Cloud Pub/Sub topic, or changes in a Cloud Storage bucket.

Cloud functions can be secured using identity-based or network-based access control. They are also isolated from each other, making it impossible for functions to access other functions running in the same environment.

Functions are allocated a pre-defined amount of RAM. This amount is set at the creation of the function, but it can be changed later.

3.2.3 Firebase

Firebase is a BaaS platform created in 2011 and acquired by Google in 2014 [24]. The platform is positioned as a mobile BaaS platform. It allows mobile application developers to replace an entire backend by using only Firebase products. For easy integration in applications, a Firebase software development kit (SDK) is available for most mobile platforms. Firebase offers products such as Realtime Database and Firestore Database, NoSQL databases API that synchronize application data across multiple platforms (iOS, Android, etc.), and store it on Firebase's cloud. There are also other products to handle authentication with Firebase Authentication, file storage with Firebase Storage, cloud functions with Cloud Functions for Firebase, machine learning, analytics, push notifications and a lot more other features. Most Firebase products do not require cold starts. For these products, the infrastructure never scales to zero as they are shared between all the customers. This is a significant advantage over most of the other FaaS approaches.

3.2.4 Amazon Web Services (AWS) Lambda

AWS Lamdba is a FaaS platform developed by Amazon and available since 2014 [25]. AWS Lambda natively supports Java, Go, PowerShell, Node.js, C#, Python, and Ruby code, and provides a runtime API which allows us to use any additional programming languages to create functions [26]. It is also compatible with containers [26]. Functions are isolated using the same virtualization technology as Amazon EC2 [26], an AWS product for creating VMs in the cloud. Functions can be executed directly using HTTP in combination with Amazon API Gateway [27], but also in

response to specific events such as changes to Amazon S3 buckets, updates to an Amazon DynamoDB table, or custom events generated by applications or devices [26]. The call of an inactive function will require a cold start because Lambda scales the infrastructure for these functions to zero after some time of inactivity.

3.2.5 Azure Functions

Azure Functions is a FaaS platform developed by Microsoft and available since 2016 [28]. The latest Azure Function runtime supports C#, JavaScript, F#, Java, PowerShell, Python and TypeScript [29]. Functions can be created using Azure's command line tool or Visual Studio Code with the Azure Functions extension. They can be deployed as containers or as code hosted on Azure Storage. Specific tools allow local development and debugging of new functions. Each function gets assigned a unique URL to trigger it. Functions can also be triggered on specific cloud events from other Azure products.

3.3 Open source software

This section provides a review of existing open-source software to run serverless applications in the cloud or on-premise. A focus was made on maintained and popular software to keep this review as short as possible.

3.3.1 Knative

Knative is an open-source project created by Google to deploy serverless workloads in a Kubernetes cluster. It can be installed on any Kubernetes cluster, whether on-premise or in the cloud.

Knative features two components that can be installed in the Kubernetes cluster [30]:

- Knative Serving to serve serverless services. Workload automatically scales to zero when no requests arrive for a predefined amount of time and scales up as more and more requests arrive. Services are packaged as container images and managed in Kubernetes by Knative Serving.

- Knative Eventing to handle a publish/subscribe model in Knative. Knative services can subscribe to specific events so that they are automatically triggered.

Services get attributed a unique URL to trigger them. They can be made public or private (only accessible in the Kubernetes cluster). Knative is very flexible, it can host function containers or any other workload container as long it is a web server application and it is stateless [31].

Services can be deployed using YAML templates or using Knative's command line client.

3.3.2 Kubeless

Kubeless is a Kubernetes native serverless framework created by Bitnami to build advanced applications with FaaS on top of Kubernetes [32]. It can be installed on any Kubernetes cluster, whether on-premise or in the cloud. Functions can be programmed in Ballerina, .NET, Go, Java, Node.js, PHP, Python and Ruby. They are deployed using Kubeless' command line client and are exposed internally in the cluster via Kubernetes services. Kubernetes ingresses can be used to expose functions outside the cluster and HTTP basic authentication can be used to control the access to the functions [33]. Kubeless is compatible with publish/subscribe frameworks such as Apache Kafka to trigger functions on specific events [34]. Autoscaling of functions is based on Kubernetes HorizontalPodAutoscaler [35]. As functions are hosted in containers, allowing functions to scale to zero will inevitably introduce cold starts when functions are idle. Kubeless features a web graphical user inface to create and manage functions in the cluster [36]. Monitoring of Kubeless is based on Prometheus [37].

3.3.3 OpenFaaS

OpenFaaS is an open-source framework to deploy serverless microservices and functions. Functions can be deployed in Kubernetes, OpenShift, Docker Swarm or to a single host with faasd [38]. They can use any programming language and are packaged into container images before being deployed. OpenFaaS offers a command line tool to manage functions, manage secret and access function logs. Once deployed, functions are accessible through a specific HTTP endpoint. A single API Gateway routes traffic to the corresponding functions [39]. OpenFaaS has a built-in autoscaling mechanism. Monitoring of OpenFaaS is based on Prometheus.

A commercial distribution of OpenFaaS called OpenFaaS PRO introduces more features such as single sign-on (SSO), OpenID Connect (OIDC), scaling functions to zero and triggering functions via Apache Kafka.

3.3.4 Fission

Fission is a framework to run serverless functions on Kubernetes. Functions can be written in any language and mapped to HTTP requests or other event triggers [40]. Functions run internally in containers, but they do not require developers to build any image. A Fission router is available in the Kubernetes cluster to access functions. This router can be exposed outside the cluster by using a Kubernetes Ingress and creating an external route. To support a specific programming language, a Fission environment must be created with all the necessary dependencies to run the code. Monitoring of Fission is based on Prometheus for metrics and Loki for logs.

Fission supports multiple types of triggers [41]:

- HTTP Triggers invoke functions when receiving HTTP requests.
- Timer Triggers invoke functions based on time.
- Message Queue Triggers for Kafka, NATS, and Azure queues.
- Kubernetes Watch Triggers to invoke functions when something in the Kubernetes cluster changes.

An advantage of using Fission is that it supports low latency cold starts (as low as 100 milliseconds) by maintaining a pool of "warm" containers that each contain a small dynamic loader [42].

3.3.5 Fn project

The Fn project is an open-source container-native serverless platform developed by Oracle that can run anywhere (in the cloud or on-premise) [43]. Officially supported languages for writing functions are Go, Java, Node.js, Ruby and Python. Fn is made to run on Docker but it can also be installed in a Kubernetes cluster using the project's Helm chart. All the work can be done using Fn's command line tool, including starting and configuring the server, and creating and invoking functions. Created functions can be automatically packaged in a container image by the command line tool. An HTTP endpoint can be generated to invoke functions. Monitoring of the Fn server is based on Prometheus. A web GUI is also available to manage and monitor the functions.

3.3.6 Apache OpenWhisk

Apache OpenWhisk is an open source, distributed serverless platform that executes functions in response to events at any scale [44]. OpenWhisk manages the infrastructure, servers and scaling using Docker containers, making it compatible with any container platform such as Kubernetes, OpenShift, and Docker Compose. Functions can be written in Go, Java, NodeJS, .NET, PHP, Python, Ruby, Rust, Scala, Swift and Ballerina. Other languages can be supported by creating a Docker image that contains the new custom runtime and using it to execute the code. In OpenWhisk, functions are called actions. These actions are invoked on specific events that are matched using triggers and rules, as described in the figure 2.

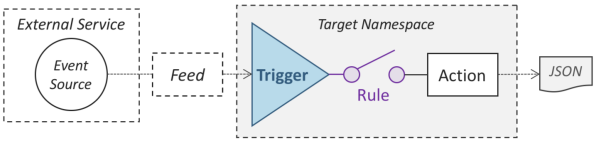

Figure 2. OpenWhisk programming model

Source: openwhisk.apache.org

To minimize cold start latency, OpenWhisk uses prewarm containers [45]. Runtime containers are started and always running in the background, waiting for the code to execute so that clients do not need to wait for containers to start.

OpenWhisk features a command line tool to quickly create and invoke functions in the cloud. Functions can also be called asynchronously or synchronously via HTTP. Synchronous calls will wait for the function to terminate and return the result while asynchronous calls will return an ActivationId to fetch results later [46].

4 Knative deployment

In this section, we describe the development and deployment of a simple synchronous cloud function to retrieve the current temperature in a given city using Knative and Google's Function Framework. Knative makes it easy to deploy serverless functions and microservices, it will be installed in a Kubernetes cluster managed by OVH's Public Cloud platform. The described solution will be as portable as possible, meaning it can be used in a variety of environments, including commercial products and on-premise container platforms.

4.1 Function creation

As Knative services must expose an HTTP server, we need to build an application that exposes one. This application will receive the name of a city and return the current temperature in this city, using an external weather API. We want to build this application using Node.js, as JavaScript is a simple language that is vastly used according to recent surveys [47]. To hide the complexity of building a web application, we use Google's Function Framework that only requires us to write the function that processes the input and returns the corresponding output. The framework allows us to build an application that will be containerized and deployable on any commercial and open-source container-based platform, whether it is serverless or not. The function will also be testable locally, by simply running it through a locally installed Node.js runtime.

The code for this function is available in a public repository on GitHub [48] and a built image can be pulled from the Docker Hub [49] with the name "tomy2e/weather-js-cloud-function".

The workflow of the function, deployed as a Docker container, is detailed in the figure 3. A client app sends an HTTP request to the container serving the function. The function fetches the temperature of the requested city using a public weather API and returns it to the client app. The entire process is done synchronously.

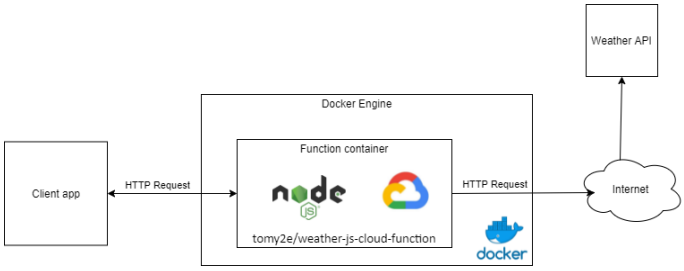

Figure 3. Cloud function running in Docker

4.2 Knative installation

To start serving functions via HTTP in an existing Kubernetes cluster, we only need to install the Knative Serving component. The process of installing Knative Serving is described in Knative's official documentation [50]. During installation, Knative also requires choosing networking layer. Architectures will slightly differ depending on the chosen networking layer. Kourier, the currently recommended networking layer, was chosen for this paper. In this setup, a load balancer provider is required. For self-managed Kubernetes clusters, MetalLB can be used to provide the required load balancer. The configuration of a wildcard domain name is also required, a subdomain will be automatically attributed to each created service.

Finally, to use Knative at its full potential and benefit from a real serverless experience, the "kn" command line tool should be installed and used to work with Knative.

4.3 Service deployment

Services are simply deployed using the "kn" command line tool. When deploying a service, the name of the service and its image name must be provided. Once the service is deployed, a unique URL is attributed to it. This unique URL contains the name of service.

The workflow of the serverless function is described in the figure 4. Requests coming from outside the Kubernetes cluster pass through the OVH load balancer to the Kourier Gateway. If the request's HTTP host header matches a known Knative service, it is forwarded to the activator service. The activator reports the number of requests to the Knative autoscaler service that is responsible for instantiating and scaling serverless services. If the serverless service was scaled to zero, the HTTP request is buffered by the activator while the autoscaler creates a new container. When the serverless service finally responds to the request, the activator forwards the response to the client and ends the connection.

5 Conclusion

To conclude, serverless computing is still a new technology that has not matured yet. Although most of the technologies presented in this paper can be considered production-ready, they are still quickly evolving and changing to adapt to more and more use-cases. For open-source software and some commercial solutions, Kubernetes seems to be the platform of choice to deploy serverless workloads on.

Most of the presented technologies suffer from a few limitations. Choosing the right tool and being aware of these limitations at the beginning of a project is critical. Some technologies are limited to a few programming languages, they often bring high latency cold starts and they may not be portable to other solutions. Testing serverless services also brings new challenges as developers are no longer aware of the underlying infrastructure their services run on.

Even though serverless computing has a lot to improve, the current advantages of using serverless computing such as lower operations costs for infrequently used services and highly reduced time to market are what makes serverless computing a very interesting technology to use.

References

[1] Steve Ranger, ZDNet. Cloud computing will virtually replace traditional data centers within three years, 2018. URL `https://www.zdnet.com/article/cloud-computing-will-virtually-replace-traditional-data-centers-within-three-years/`. [Online; accessed 29-June-2021].

[2] Microsoft. What is iaas?, 2021. URL `https://azure.microsoft.com/en-us/overview/what-is-iaas/`. [Online; accessed 29-June-2021].

[3] Amazon. Aws elastic beanstalk, 2021. URL `https://aws.amazon.com/elasticbeanstalk/`. [Online; accessed 29-June-2021].

[4] Erin Baez, Scalyr. Function as a service (faas) explained, simply, 2020. URL `https://www.scalyr.com/blog/function-as-a-service-faas/`. [Online; accessed 29-June-2021].

[5] Davide Taibi, Josef Spillner, and Konrad Wawruch. Serverless computing-where are we now, and where are we heading? *IEEE Software*, 38(1):25–31, 2021. doi: 10.1109/MS.2020.3028708.

[6] Microsoft. Serverless computing, 2021. URL `https://azure.microsoft.com/en-us/overview/serverless-computing/`. [Online; accessed 29-June-2021].

[7] TechTarget. Definition: event-driven application, 2021. URL `https://searchitoperations.techtarget.com/`

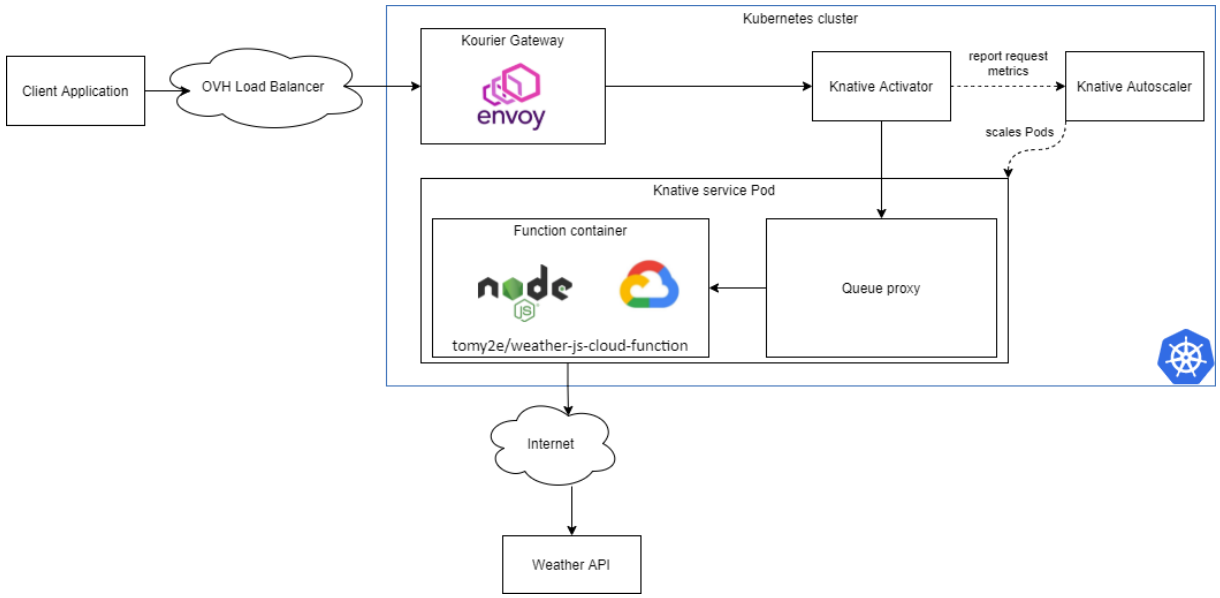

Figure 4. Cloud function running in Knative

definition/event-driven-application. [Online; accessed 29-June-2021].

[8] Peter Sbarski. Serverless cost calculator, 2017. URL http://serverlesscalc.com/. [Online; accessed 29-June-2021].

[9] George Batschinski. Backend as a service list, 2019. URL https://george-51059.medium.com/backend-as-a-service-list-9104bdce845e. [Online; accessed 29-June-2021].

[10] Wikipedia contributors. Docker (software) — Wikipedia, the free encyclopedia, 2021. URL https://en.wikipedia.org/w/index.php?title=Docker_(software)&oldid=1029236841. [Online; accessed 29-June-2021].

[11] Charles Anderson. Docker [software engineering]. *IEEE Software*, 32(3):102–c3, 2015. doi: 10.1109/MS.2015.62.

[12] Yongqiang Huang and Hector Garcia-Molina. Publish/subscribe in a mobile environment. *Wireless Networks*, 10(6):643–652, 11 2004.

[13] Amazon. Pub/sub messaging, 2021. URL https://aws.amazon.com/fr/pub-sub-messaging/. [Online; accessed 29-June-2021].

[14] Google. Cloud run, 2021. URL https://cloud.google.com/run?hl=en. [Online; accessed 29-June-2021].

[15] Oren Teich, Pali Bhat. Cloud run, a managed knative service, is ga, 2019. URL https://cloud.google.com/blog/products/serverless/knative-based-cloud-run-services-are-ga. [Online; accessed 29-June-2021].

[16] Google. Container runtime contract, 2021. URL https://cloud.google.com/run/docs/reference/container-contract. [Online; accessed 29-June-2021].

[17] Google. Using minimum instances, 2021. URL https://cloud.google.com/run/docs/configuring/min-instances. [Online; accessed 29-June-2021].

[18] Google. Triggering from pub/sub push, 2021. URL https://cloud.google.com/run/docs/triggering/pubsub-push. [Online; accessed 29-June-2021].

[19] Janakiram MSV, Forbes. Google brings serverless computing to its cloud platform, 2016. URL https://www.forbes.com/sites/janakirammsv/2016/02/09/google-brings-serverless-computing-to-its-cloud-platform/. [Online; accessed 29-June-2021].

[20] Google. Cloud functions documentation, 2021. URL https://cloud.google.com/functions/docs. [Online; accessed 29-June-2021].

[21] Google. Writing cloud functions, 2021. URL https://cloud.google.com/functions/docs/writing. [Online; accessed 29-June-2021].

[22] Google. Functions framework, 2021. URL https://github.com/GoogleCloudPlatform/functions-framework. [Online; accessed 29-June-2021].

[23] Google. Your first function: Go, 2021. URL https://cloud.google.com/functions/docs/first-go. [Online; accessed 29-June-2021].

[24] Wikipedia contributors. Firebase — Wikipedia, the free encyclopedia, 2021. URL https://en.wikipedia.org/w/index.php?title=Firebase&oldid=1029031665. [Online; accessed 29-June-2021].

[25] Ron Miller, TechCrunch. Amazon launches lambda, an event-driven compute service, 2014. URL https://techcrunch.com/2014/11/13/amazon-launches-lambda-an-event-driven-compute-service/. [Online; accessed 29-June-2021].

[26] Amazon. Aws lambda faqs, 2021. URL https://aws.amazon.com/lambda/faqs/. [Online; accessed 29-June-2021].

[27] Amazon. Tutorial: Build a hello world rest api with lambda proxy integration, 2020. URL https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-create-api-as-simple-proxy-for-lambda.html. [Online; accessed 29-June-2021].

[28] Yochay Kiriaty, Microsoft. Announcing general availability of azure functions, 2016. URL https://azure.microsoft.com/en-us/blog/announcing-general-availability-of-azure-functions/. [Online; accessed 29-June-2021].

[29] Microsoft. Supported languages in azure functions, 2019. URL https://docs.microsoft.com/en-us/azure/azure-functions/supported-languages. [Online; accessed 29-June-2021].

[30] The Knative Authors. Knative, 2021. URL https://knative.dev/. [Online; accessed 29-June-2021].

[31] KnativeTips Authors. What can be run on knative serving?, 2020. URL https://knative.tips/basics/suitable-applications/. [Online; accessed 29-June-2021].

[32] Kubeless. Kubeless, 2021. URL https://kubeless.io/. [Online; accessed 29-June-2021].

[33] Kubeless. Expose and secure kubeless functions, 2021. URL https://kubeless.io/docs/http-triggers/. [Online; accessed 29-June-2021].

[34] Kubeless. Pubsub events, 2021. URL https://kubeless.io/docs/pubsub-functions/. [Online; accessed 29-June-2021].

[35] Kubeless. Autoscaling function deployment in kubeless, 2021. URL https://kubeless.io/docs/autoscaling/. [Online; accessed 29-June-2021].

[36] Kubeless. kubeless-ui, 2021. URL https://github.com/kubeless/kubeless-ui. [Online; accessed 29-June-2021].

[37] Kubeless. Monitoring, 2021. URL https://kubeless.io/docs/monitoring/. [Online; accessed 29-June-2021].

[38] OpenFaaS Author(s). Deployment, 2021. URL https://docs.openfaas.com/deployment/. [Online; accessed 29-June-2021].

[39] OpenFaaS Author(s). Gateway, 2021. URL https://docs.openfaas.com/architecture/gateway/. [Online; accessed 29-June-2021].

[40] Fission Authors. Fission, 2021. URL https://fission.io/. [Online; accessed 29-June-2021].

[41] Fission Authors. Concepts, 2020. URL https://docs.fission.io/docs/concepts/. [Online; accessed 29-June-2021].

[42] Fission Authors. Fission, 2021. URL https://docs.fission.io/docs/. [Online; accessed 29-June-2021].

[43] Fn. Fn project, 2021. URL https://fnproject.io/. [Online; accessed 29-June-2021].

[44] The Apache Software Foundation. Apache openwhisk, 2021. URL https://openwhisk.apache.org/. [Online; accessed 29-June-2021].

[45] Markus Thömmes. Squeezing the milliseconds: How to make serverless platforms blazing fast!, 2017. URL https://medium.com/openwhisk/squeezing-the-milliseconds-how-to-make-serverless-platforms-blazing-fast-aea0e9951bd0. [Online; accessed 29-June-2021].

[46] The Apache Software Foundation. System overview, 2017. URL https://github.com/apache/openwhisk/blob/master/docs/about.md. [Online; accessed 29-June-2021].

[47] Stack Exchange Inc. Stack overflow developer survey 2020, 2020. URL https://insights.stackoverflow.com/survey/2020. [Online; accessed 29-June-2021].

[48] Tomy Guichard. Weather js cloud function (github repository), 2021. URL https://github.com/Tomy2e/weather-js-cloud-function. [Online; accessed 29-June-2021].

[49] Tomy Guichard. Weather js cloud function (docker image), 2021. URL https://hub.docker.com/r/tomy2e/weather-js-cloud-function. [Online; accessed 29-June-2021].

[50] The Knative Authors. Installing knative serving using yaml files, 2021. URL https://knative.dev/docs/install/install-serving-with-yaml/. [Online; accessed 29-June-2021].