iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 8 **Issue:** VI **Month of publication:** June 2020

DOI: http://doi.org/10.22214/ijraset.2020.6144

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 8 Issue VI June 2020- Available at www.ijraset.com

A Study of Serverless Architecture: An Overview

L Pavan Venkat[1], A. Akshay Raja Reddy[2], Priya D[3], Kavitha S. N[4]
[1, 2]*Student,* [3, 4]*Assistant Professor, R V College of Engineering, India*

Abstract: ***Serverless computing has become a compelling new model for the implementation of peripherals and utilities. It represents the emergence of computer systems, abstractions and interfaces, and pays tribute to the growing popularity and creation of software.***
Serverless architecture involves custom code that integrates with a third-party back-end as a service (BaaS) or runs in an intangible jar maintained on a "function as a service" platform (FaaS). Application prototype. These architectures contemplate using the same ideas and related topics such as single page applications to still get rid of most of the typical component uses on the server. Serverless architectures can benefit from a significant reduction in operating costs, sophistication and development time at the expense of increased supplier dependencies and increased dependence on relatively inexperienced support systems. .
Serverless is the next evolution of cloud computing. This helps with application development without having to worry about storage, hardware, and processing of applications. Developers must manage business logic. Useful when microservices execute business logic. It offers great flexibility at virtually any speed. The serverless architecture allows consumers to rethink cutting-edge products from concept to manufacturing without having to wait and sweat on the system.
Keywords: Serverless, microservices, PaaS, FaaS, Aws, Baas.

I. INTRODUCTION

Serverless is a cloud computing implementation paradigm where service providers manage server allocation and provisioning dynamically. Serverless applications run in stateless, event-triggered, transient (possibly persistent single call) IT containers and are fully managed by the cloud provider. The price is based on the number of executions and not on the computing power acquired in advance.
Serverless (or just serverless) computing has emerged as an attractive new paradigm for deploying cloud applications, largely due to recent changes in enterprise cloud and microservices architectures. Infrastructure as a Service (IaaS)
From the client's point of view, this paradigm shift represents both opportunities and possibilities. On the one hand, it provides developers with a simplified programming model for creating cloud applications that sums up most, if not all, of the operational concerns. Reduce the cost of deploying cloud code by charging time instead of allocating resources. It is also a rapid deployment platform for small snippets of native cloud code that respond to events such as changing microservice configurations. Otherwise, it runs in the client or dedicated middleware. Or, these applications are deployed on serverless disks.
From the cloud provider's perspective, serverless computing provides more capacity to manage the entire technology chain, reduces operating costs, makes it easier to use new environmental services, and eliminates the required commitment of cloud services. It provides a framework that facilitates efficient optimization and deployment for the creation and execution of cloud-wide programs.

II. ARCHITECTURE

There are some misconceptions about the serverless server starting with the name. All servers are still needed, but developers don't have to worry about managing these servers. The serverless architecture is responsible for decisions such as the number and capacity of servers and automatically provision server space on demand. This provides a separate environment from the one where calculations (for stateless functions) are performed.
A central feature of serverless platforms is that of an event management system. The service manages a set of user-defined events, retrieves events sent via HTTP or events collected from event sources, determines the function to which the event is sent and finds new iterations of existing functions. . The instance must send an event to the feature instance, wait for the response, collect the execution log, access the user response, and terminate the feature. Applications must start tasks quickly and efficiently and process their content. The structure must also queue events, schedule the execution of tasks and control the stopping of inactive task instances and the allocation of resources according to the state of the queue and the rate of arrival of events. Therefore, in a cloud environment, programs should carefully consider how to scale up and manage failures.

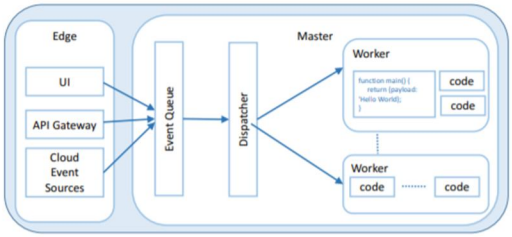


Fig. Serverless Platform Architecture

III. COST ANALYSIS

Amazon Web Services was the main cloud provider of serverless "lambda functions" in 2014, but from that point on, comparative offers from Google, Microsoft, IBM and others were presented. In fact, various miners plan to trade and aspire to success in the market. There are different differences and some comparisons between predictive models and actual costs, and cloud pricing is well known.

Using lambdas without a server doubles the cost. One is the cost of the exhibit and the other is the cost of the hidden. Visible costs include the demand and use of the CPU and RAM. Confidential costs include API requests and networking costs. There are also unexpected costs to maintain coding. An AWS Lambda job with 512 MB of memory costs $ 0.030024 compared to an on-demand server that requires the same information.

$ 0.0059. Therefore, serverless operation does not add a positive load to the workload, since the processor is fully used continuously. Serverless can be expensive if storage and network are the best candidates for the cost. Associated with a large number of APIs, API Gateway tends to represent a huge share of the cost without a server. The more APIs you request per request, the less you will benefit from converting to serverless.

Severless - Prices

	AWS	Azure	Google	IBM
Name of Services	Lambda	Functions (Consumption plan)	Cloud Functions	Cloud Functions
Function Invocations (per 1M)	$0.20	$0.20	$0.40	N/A
Duration/Memory (per 1M GB-secs)	$16.67	$16.00	$2.50	$17.00
Duration/CPU (per 1M GHz-secs)	N/A	N/A	$10.00	N/A
Network egress (per GB)	$.09	$.087	$0.12	$.09
Free Invocations (per month)	1M	1M	2M	N/A
Free Duration/Mem (per month)	400K	400K	400K	400K
Free Duration/CPU (per month)	N/A	N/A	200K	N/A
Free network egress (per month)	Part of overall EC2 free tier of 1 GB	Part of overall free tier of 5 GB	5 GB for Cloud Functions	None noted for Cloud Functions

As of Mar 28, 2018 RIGHTSCALE *Source: RightScale*

Fig. Price Comparison of Serverless platforms

IV. ADVANTAGES

There are many advantages with respect to the adaptation of serverless architecture. They are discussed accordingly as follows:

A. Reduction in Operational Cost

Serverless is the most basic automation approach. This allows you to hire everyone to manage the data center, the database, and even the code and logic that you work with. The predefined programs that many other people continue to use have economies of scale. A provider has thousands of very identical databases, so the cost of managing them can be high. Will be less. The first concerns the technological cost savings that result from the simple sharing of resources with other users (hardware, network, etc.).

B. Backend as a Service : Reduced Development Cost

IaaS and PaaS are based on the concept of changing server and operating system management. On the other hand, a serverless backend as a service is a product that markets all of the components of an application.

A good example is authentication servers. Many applications encode their own authentication features, which often include features such as registration, login, password management, and integration with other authentication providers. In general, this idea is very familiar to most systems, and tools like Auth0 have been developed to allow you to integrate a predefined authentication functionality into your application without having to write it yourself.

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.429
Volume 8 Issue VI June 2020- Available at www.ijraset.com

C. FaaS: Scaling Costs

It takes care of dismantling the components of microservices which are likely to have a very low charge demand, even if the operating costs of these finer solutions would not be achievable otherwise.

Great Democrats are such a cost advantage. When a business or a team member has to do something different, the operating costs associated with using FaaS to meet their IT needs are very low. In fact, if the overall workload is low enough (but not entirely large), a particular FaaS provider may not allow you to pay for the calculation at all.

D. Ease of Operational Management

On the serverless BaaS side of the enclosure, it is quite obvious that the management process is simpler than the other architectures. Reducing the number of components means reducing the work. However, there are different aspects of the FaaS side.

E. Reduced Complexity of Packaging and Deployment

Loading and starting FaaS functionalities is simple and clear compared to starting the whole server. Everything to do is a box and transfer all the code to a zip file. Puppet / Chef, shell start / stop commands, no choice to run one or more containers on the server. Just starting out, there is nothing to group.

Defining this process does not take much time, but for some teams, this advantage can be quite significant: a completely serverless solution requires zero system administration. .. While PaaS systems offer comparable implementation opportunities, when comparing PaaS and FaaS, the benefits of scaling are specific to FaaS.

V. DRAWBACKS

There are certain inherent drawbacks that the serverless architectures bring to the table.They are discussed accordingly as follows:

A. Problems with Multi Tenancy

Multi-tenancy refers to the situation where several instances of the software run on the same machine, possibly in the same hosting application, for different customers (or tenants). It is a strategy for achieving the economies of scale described above. Service providers work very hard to convince consumers that they are the only ones using their network, and good service providers generally do it well.

But a perfect solution, sometimes multi-tenant, has security problems (a customer can see the data of another customer), robustness (an error in the software of a customer causes the software of another customer to fail), And performance (a high load client slows down another client).

B. Lock-in by Vendor

It is also possible that the serverless functionality of one provider may be implemented differently by another provider. If you change providers, you will almost certainly need to upgrade your operational tools (deployment, testing, etc.) and modify your programming (for example, to adapt to a particular FaaS interface). Also, if there are variations in the capabilities of implementations from competing vendors, you may need to adjust your configuration or layout.

Even if you can quickly move one aspect of the environment, another architectural feature may have a greater impact. For example, suppose you use AWS Lambda to respond to events on the AWS Kinesis message bus. The differences between AWS Lambda, Google Cloud Functions and Microsoft Azure Functions can be relatively small, but the last two providers' implementations cannot connect directly to AWS Kinesis streams. In other words, you cannot move or carry the code from one solution to another without moving other pieces of your infrastructure.

C. Loss of Optimisation of Servers

With a full BaaS architecture, there is no way to optimize the server design for client efficiency. The front-end back-end model occurs at the server level and summarizes some of the fundamental aspects of the overall system. This allows customers to perform operations faster and save battery power for mobile applications. No template is required for a full BaaS.

All of this and the previous shortcomings occur in a complete BaaS architecture where all of the custom logic is in the client and the provider provides the only backend service. The two mitigations are to use FaaS or another type of lightweight server-side model to move specific logic to the server.

D. Monitoring and Observability

Monitoring is a difficult environment for FaaS because the containers are inherently short-lived. Many cloud providers offer monitoring assistance, but again, we've seen a lot of research from traditional monitoring providers. And what they can do, and what you can do, basically depends on the provider's master data.

In other cases, it might be perfect, but at least for AWS Lambda, it's very easy. What we really need is an open API in this space and the ability of third-party providers to support it more.

E. Debugging

One area of interest is debugging with FaaS. Here we see progress mainly in the functioning of local FaaS services following the above software changes. Microsoft provides excellent debugging tools for features that run locally but are triggered by remote events. Amazon offers similar functionality, but no development activity has yet taken place.

F. Do Yourself

AWS Lambda limits the number of concurrent executions of Lambda functions that can run at the same time. Suppose this limit is 1000. This means that you can run 1000 function instances at any time. If you have to go further, you can usually start getting extensions, wait, or slow down.

The problem here is that this limit applies to all AWS accounts. Some companies use the same AWS account for development and research.

This means that if someone performs a new type of load test somewhere in your organization and tries to execute 1,000 simultaneous Lambda functions, they are unintentionally executing the exit program.

Even if you use different AWS accounts for production and large environments, when one overloaded lambda storage (for example to manage a client's batch downloads) triggers another The lambda-based output API is triggered.

VI. THE FUTURE OF SERVERLESS

The serverless environment is still relatively recent. Serverless's most significant innovations would be reducing, eliminating, or at least strengthening, the implementation drawbacks.

A. Improvement of the Platform

Some of the disadvantages of serverless FaaS are currently due to the way the platform is implemented. Execution time, launch latency and inter-function limits are three notable. They can be resolved with an innovative approach or offer workarounds with potential additional costs. For example, launch latency can be mitigated by allowing customers to request that there are always two instances of the FaaS functionality with low latency. Customers who pay for this availability. Robust functionality of Microsoft Azure is part of this concept and has the ability to host device services.

B. Education

Many server-specific traps specific to the server are mitigated by awareness. Everyone who uses these sites should carefully consider what one or more program providers need to manage too many ecosystems.

Basic activities are another area of the school. Most organizations have fewer system administrators than before, so having a server makes migration easier. However, the system administrator is not only the one who installs Unix boxes and Chef scripts, but he is also at the forefront of services, networking and storage.

C. Increased Transparency and clear Expectations of Suppliers

Providers need to be more specific about what they can expect from the platform to take advantage of more hosting capabilities. Migration of the platform is difficult, but not impossible. Untrusted suppliers see customers doing business elsewhere.

D. New Technology and Security

Over time, serverless technologies will mature and people will become more aware. Developers are following new and improved security guidelines to avoid vulnerabilities.

E. Abstraction of the supplier's Implementation

The serverless architecture is primarily used to simplify the operational activities of serverless systems, but it also provides transparency about when and how to implement these applications. The tricky part of it all is designing an abstract FaaS computer interface with no knowledge of standardization, but that's exactly what the CNCF Serverless Cloud Events working group does.

F. State Management

The persistent lack of server state in FaaS is great for a decent range of applications, but for many others, whether it's a huge cache set or easy state access session. Commercial circuit breaker. A solution for high-speed applications has the potential to allow providers to keep instances of functionality longer between events and to enable standard caching approaches.

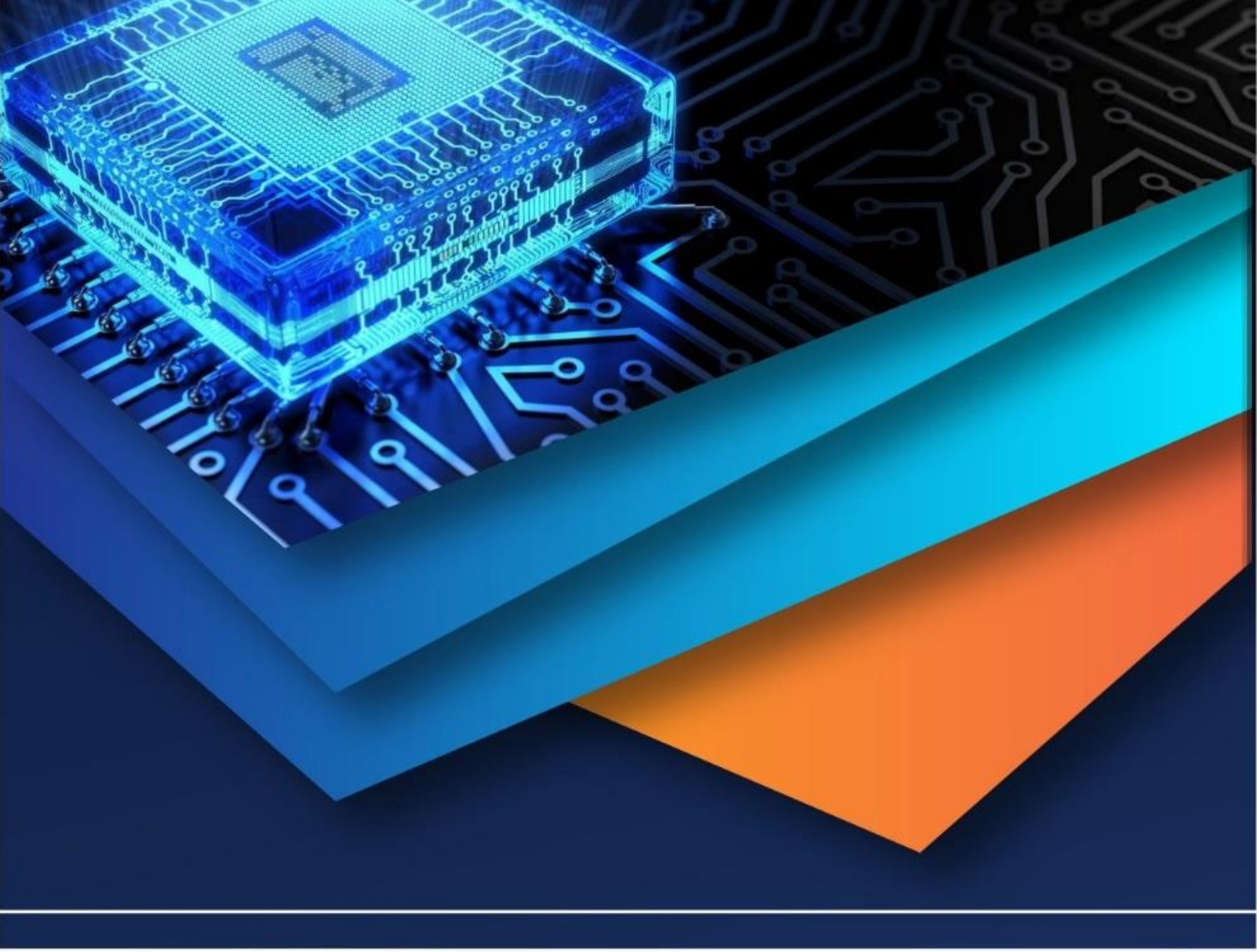
A better solution is to access out-of-process data with very low latency, such as querying the Redis database with very low network overhead. Given that Amazon already offers a Redis solution hosted via the Elasticache application, and also requires a relative colocation of EC2 instances (server) via placement groups, this does not seem to be a huge advance.

VII. CONCLUSION

In addition, today's serverless technology is essential for high-speed, non-inactive deployments, and helps with urgent tasks that can meet individual demands in a relatively short period of time. The economic implications of this demarcation in a serverless environment make it an essential tool for radically reducing cost facilitation and increasing the time required to promote a new future. It has low operating and production costs and reduces the complexity of the program through the use of microservices. But, like all positive things, suffering without a server has its pitfalls. Not suitable for long-term applications. Another major drawback is the foreclosure of suppliers. The program relies only on subcontractors. You do not need to fully monitor the request. You cannot permanently change scene or supplier without important developments Application upgrade. Additionally, APIs and platform costs may change due to the availability dependency of the platform. With comprehensive advice and group support, you will soon find that the demand for learning and adapting FaaS devices is quite high. Likewise, you need to divide the monolith into microservices. This is another tricky challenge and you can easily upgrade to a server without a server. That's why it's important to get help from a serverless resource expert. Serverless is well suited for systems such as IoT, virtual assistants and chatbots, image-rich applications and pipelines for agile and continuous delivery.

REFERENCES

[1] Aws lambda. URL https://aws.amazon.com/lambda/. Online; accessed December 1, 2016
[2] S3 Simple Storage Service. URL https://aws.amazon.com/s3/. Online; accessed December 1, 2016
[3] Building Serverless Apps with Webtask.io. URL https://auth0.com/blog/ building-serverless-apps-with-webtask/. Online; accessed December 1, 2016
[4] Aws re:invent 2014 — (mbl202) new launch: Getting started with aws lambda. URL https: //www.youtube.com/watch?v=UFj27laTWQA. Online; accessed December 1, 2016
[5] Bainomugisha, E., Carreton, A.L., Cutsem, T.v., Mostinckx, S., Meuter, W.d.: A survey on reactive programming. ACM Comput. Surv. 45(4), 52:1–52:34 (2013). DOI 10.1145/2501654. 2501666. URL http://doi.acm.org/10.1145/2501654.2501666
[6] Baldini, I., Castro, P., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Suter, P.: Cloud-native, event-based programming for mobile applications. In: Proceedings of the International Conference on Mobile Software Engineering and Systems, MOBILESoft '16, pp. 287–288. ACM, New York, NY, USA (2016). DOI 10.1145/2897073.2897713. URL http://doi.acm.org/10.1145/2897073.2897713
[7] Bienko, C.D., Greenstein, M., Holt, S.E., Phillips, R.T.: IBM Cloudant: Database as a Service Advanced Topics. IBM Redbooks (2015)
[8] Learn Chaincode. URL https://github.com/IBM-Blockchain/learnchaincode. Online; accessed 1 December 2016
[9] chalice: Python serverless microframework for aws. URL https://github.com/ awslabs/chalice. Online; accessed December 1, 2016
[10] Ethereum. http://ethdocs.org/en/latest/introduction/what-isethereum.html. URL http://ethdocs.org/en/latest/introduction/ what-is-ethereum.html. Online; accessed December 1, 2016
[11] Fernandez, O.: Serverless: Patterns of Modern Application Design Using Microservices (Amazon Web Services Edition). in preparation (2016). URL https://leanpub.com/ serverless
[12] OpenFog Consortium. URL http://www.openfogconsortium.org/. Online; accessed December 1, 2016
[13] Galactic Fog Gestalt Framework. URL http://www.galacticfog.com/. Online; accessed December 1, 2016
[14] Cloud functions. URL https://cloud.google.com/functions/. Online; accessed December 1, 2016
[15] Hendrickson, S., Sturdevant, S., Harter, T., Venkataramani, V., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H.: Serverless computation with openlambda. In: 8th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud 2016, Denver, CO, USA, June 20- 21, 2016. (2016). URL https://www.usenix.org/conference/hotcloud16/ workshop-program/presentation/hendrickson
[16] Openwhisk. URL https://github.com/openwhisk/openwhisk. Online; accessed December 1, 2016 20
[17] Cloud Foundry and Iron.io Deliver Serverless. URL https://www.iron.io/cloudfoundry-and-ironio-deliver-serverless/. Online; accessed December 1, 2016
[18] Introducing Lambda support on Iron.io. URL https://www.iron.io/introducingaws-lambda-support/. Online; accessed December 1, 2016
[19] Sharable, Open Source Workers for Scalable Processing. URL https://www.iron.io/ sharable-open-source-workers-for/. Online; accessed December 1, 2016
[20] Jira. URL https://www.atlassian.com/software/jira. Online; accessed December 5, 2016



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY