Reliable Cloud Software Development Architectures and Business Models Case Study: RIDEaaS and GAE Launcher

Orges Cico
Department of Computer Science
Norwegian University of Science and Technology
Trondheim, Norway
orges.cico@ntnu.no

Betim Cico
Department of Computer Engineering
EPOKA University
Tirana, Albania
bcico@epoka.edu.al

ABSTRACT

Development of cloud applications directly on the cloud infrastructure has become a common approach. Reliability concerns have also become more of a challenge during the last years. Our goal is to assert the potentials deriving from an integrated development environment, adopting software reliability concepts and fault tolerant techniques, as part of the cloud core services. As a methodology, we propose the implementation of a new cloud service Reliable Integrated Development Environment as a Service to become part of the existing core services. Our objectives are to 1) fulfill the need of having a development environment independent from personal desktop environments; 2) code development environment should be offered directly from the cloud service providers. With fault tolerant technique integration, we also suggest that the reliability of the cloud system should be handled from the cloud developers at software level when cloud quality assurance services fail to do so at their different levels. We also propose potential business models to become part of this core service for the major cloud providers. We introduce a new High Availability Coding model providing a roadmap for the future.

CCS CONCEPTS

• High Availability Coding • Fault Tolerant Techniques • Integrated Development Environment

KEYWORDS

Cloud systems, cloud software development, reliable integrated development environment, cloud SDK

ACM Reference Format:
Orges Çiço and Betim Çiço. 2019. Reliable Cloud Software Development Architectures and Business Models Case Study: RIDEaaS and GAE Launcher. In *Proceedings of 9th Balkan Conference on Informatics (BCI'19). September 26–28, 2019, Sofia, Bulgaria.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/ 3351556.3351586

BCI'19, September 26–28, 2019, Sofia, Bulgaria

ACM ISBN 978-1-4503-7193-3/19/09…$15.00
https://doi.org/10.1145/3351556.3351586

1 Introduction

Online code development has become more and more popular in the last five years. The advantages of utilizing online development environments have mainly been understood and discussed in [1, 2, 3]. Third party solutions have also been proposed for managing code development for different cloud providers. A review among existing major cloud providers (Amazon, Google, Azure) has been performed, as well as their approaches for online software development [4]. Some have proven to be more ahead of the others, but none of them have fully integrated a collaborative environment for large-scale projects. Either ad-hoc [5, 6, 7] or third-party solutions [8] have been adopted so far. However, both choices lack proper integration with cloud providers or do not provide a proper business model which would fit best the cloud tenants and end-users. Cloud reliability has also become a concern during the last decade [9]. We have previously proposed the migration of the current cloud Software Development Kit (SDK) on the cloud Platform as a Service (PaaS) infrastructure. In [10], we have further proposed the implementation of an Integrated Development Environment as a Service adopted at different cloud levels. We have proposed the adoption of the fault tolerant techniques within the scope of cloud code development, which can become part of the new Reliability as a Service [11]. We observed the need for the two previous proposals to be integrated into a standard service, Reliable Integrated Development Environment as a Service (RIDEaaS), being complementary to each other. We provide the integration through an experimental service running on Google App Engine called GAE Launcher and by migrating a development environment to the cloud with coding, deployment, debugging, logging functionalities. We ported the SDK from its current desktop version to the cloud platform, offering similar existing features as well as new ones. It currently works with python SDK, we plan to add more programming languages with the same approach. Our new framework allows new business model opportunities as well, which might either be a derivative of similar approaches already adopted on cloud platforms such as Pay as you go Coding (PaygoC) or On Demand Coding (ODC) for large scale collaborative projects and new outsourcing opportunities. Both proposed models have been clarified within [10] and would give cloud providers a quick economic boost in entering new open

source development communities and enterprise tenants. RIDEaaS arises new opportunities for increased confidence in developing highly reliable applications with robust code and the easiness to be applied throughout a cloud framework. The new service would make cloud providers enter new markets based on models proposed in [10], but also from coding optimization services that could be charged per use or allocation.

Our contribution is to prove the relevance of having cloud providers offer integrated development environment that would permit developers to code directly on their platforms with the intent to develop fault tolerant code tailored to the cloud infrastructure and services they are exploiting.

In section II, we describe part of the cloud service background. In section III, we report the state of the art fault tolerant techniques integrated at different cloud service and IDE environments. Section IV presents the RIDEaaS model and framework. Section V presents the actual implementation of the Google cloud of the RIDEaaS alongside the economic models. Finally, section VI discusses our proposal and conclusions.

2 Background

Most cloud systems are based on the following service layers of abstraction:

1) SaaS (Software as a Service)
2) PasS (Platform as a Service)
3) IaaS (Infrastructure as a Service)
4) MBaaS (Mobile Backend as a Service)

All the three layers are stacked up, providing the backbone for most cloud architectures today. Clients usually exploit the services at different levels of abstraction based on their immediate necessities. Figure 1 describes a possible representation of the stacked up services and their interdependency with cloud users. The figure tries to present an encapsulated representation so that it can be evident for the end user that services starting from core hardware augment themselves by reusing the other underlying services. This representation helps to understand the concept described in the paper by adding other service layers could involve some or all of the existing ones. We will later use this concept for our proposed service. Infrastructure as a Service (IaaS) offers what is commonly required as a basic functionality from the cloud system relates to its existing infrastructure in terms of hardware resources such as computation resources, data storing/backup, load balancing, scalability, security, etc. Platform as a Service (PaaS) offers the appropriate environment for developing and deploying applications. This is typically identified as services related to web servers and their execution environment for different programming and scripting languages, operating systems, databases, etc. Software as a Service (SaaS): Software distribution and their runtime environment are nowadays relying on distributed service providers, commonly known as cloud providers.

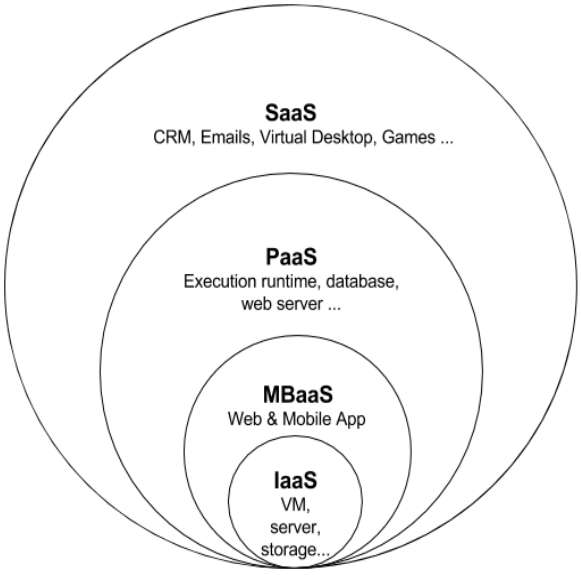

Figure 1: Elliptic view of the cloud services and their interdependency with cloud users

In this case, the provider is fulfilling all the management of the previously mentioned services IaaS and PaaS and the end user is only exploiting the different software functionalities, running on the cloud. Mobile Backend as a Service (MBaaS) cloud endpoints such as cloud storage, computing services, information/data retrieval. are accessed through well-defined application programming interfaces (APIs). This approach is fully integrated with services related to social media sharing and authentication; push up notifications commonly exploited for mobile applications.

2.1 Cloud SDK Tools

The cloud SDK commonly represents a set of tools for Cloud Platforms, which allow the end user to fully exploit the cloud infrastructure at a different service level (PaaS, IaaS). Such tools are usually part of CLI or GUI interfaces mostly common in the major cloud providers. The SDK-s also supports multiple programming languages and the primary objective is to allow not just development, but also the management of virtual clusters of computers. Each of which has similar properties to personal computers and provide primarily virtualization of basic computing components and I/O, so the systems can provide services starting from Web servers and continuing with CRM, database, computation, etc. Most of the services for every major cloud provider such as Amazon, Azure, and Google operate with on-demand or pay per use strategy (PAYG). The platforms are very flexible for every individual, business or governmental sector.

2.2 Cloud Supported Programming Languages and Technologies

The set of programming languages and environments supported proves the flexibility of current cloud platforms to support a large scale of technologies, Table I. The establishment of an integrated

cloud-based IDE environment on demand would not just reduce the scale of licensing, but also contribute into optimizing individual, business and enterprise resources when it comes to development of large-scale applications. Most developers will not be restricted from proprietary tools of third-party entities and shall rely mostly on the cloud infrastructure to keep their development going. This would not just reduce the operational cost but also help in increasing the productivity and collaborative approaches for distributed working environments. The possibility based on the similarities in programming languages and technologies could also contribute to integrating further different cloud providers. IDE environment has been widely influenced by polyglot, cross-platform and collaborative software development. RIDEaaS as some of the samples presented in the next section tries to address the long-term issue bringing the three characteristics together so that cloud development becomes as easy and efficient as the rest of the services already provided.

Table 1: Supported programming languages of major cloud providers

Cloud Provider	Denomination	PaaS Supported Programming Languages
Google	Google App Engine	Go, PHP, Java, Python, Node, .NET, Ruby
Amazon	AWS Elastic Beanstalk	Java, .NET, PHP, Node.js, Python, Ruby, Go
Microsoft	Azure Azure Cloud Services	Java, .NET, PHP, Node.js, Python, Ruby

3 State of the Art

3.1 Adopted fault tolerant and high reliability techniques in cloud services

Based on literature review the High Reliability (HR) concerns have been classified as follows based on a simplified view of Service Availability Forum (SAF):

- Virtualization and Middleware approaches. The main types of failures addressed in the proposed framework are at different levels.
- Layer based approach. Three different layers: a. Underlying Technologies - Addressing HR at Virtual Machine Level; b. Services - Providing fault tolerant mechanisms as services configurable from the cloud providers; c. Middleware - Handle mainly service operation, configuration, and decision making according to failure scenario.

3.1.1 Common Fault Tolerant techniques adopted at service level

Redundancy: The redundancy service can offer different levels of availability depending on the redundancy model, the redundancy strategy, and the redundancy scope. The redundancy model refers to the many different ways HA systems which can combine active and standby replicas of hosted applications. There exist four models [12]: 2N, N+M, Nway, and Nway active. The 2N ensures one standby replica for each active application. The scope of redundancy implementation could be at any layer such as Application, Virtual, and Physical Machine Layer.

Data replication: Data replication is used to maintain state consistency between replicas. The main problem associated with this service is the question of how to govern the tradeoff between consistency and resource usage [13]. In Cloud, the replication may be achieved either by copying the state of a system (checkpoint) or by replaying input to all replicas (lock-step based) [14].

Monitoring: Monitoring is a crucial service in an HA Cloud. Through this service, applications health is continuously observed to support other services. The primary goal of this service is to detect when a replica is down, but robust implementations can also follow the health indicators of an application (CPU and memory utilization, disk, and network I/O, time to respond requests) which will help to detect when a replica is malfunctioning. It can also be done at the virtual and physical machine level.

Failure detection: Failure detection is an important service contained in most HR solutions, which aims to identify systems faults (application, virtual or physical machine level) and provide needed information for services capable of treating problems to maintain service continuity. The authors list some mechanisms used to detect faults like ping, heartbeat and exceptions. From this perspective, failure detection can be classified in two categories according to detection mechanisms: reactive and proactive [15]. The first approach waits for KEEP ALIVE messages, but it identifies a failure after a period of time waiting without any KEEP ALIVE message. The second approach is more robust and is capable of identifying abnormal behaviors in the environment, checking the monitoring service and interpreting collected data to verify whether there are failures or not. Recovery Software system and cloud service recovery is part of a broader fault tolerant process containing the steps previously described. The latter usually encompasses error/fault detection, debugging, isolation and recovery. Different types of recovery have been proposed based on forward and backward recovery. In cloud services, these techniques heavily rely on redundancy at different levels of operation (Virtual or Physical Machine). Commonly backward recovery involves rolling back the system to a previous state stored as a failure-free checkpoint of the system. The most adopted approach is to Recover control Blocks (RcB). Forward recovery tries to continue the system execution by identifying a new state running on a parallel entity. N-version Programming (NVP) is the most commonly adopted approach. Exploiting the concept of backward and forward recovery where both attempts to return the system to a correct/error-free state the recovery techniques operating onto the Cloud have been classified into simple and smart. The simple approach involves rebooting completely the running application on the same/different node (VM) but eventually all data and states information are lost. While the smart recovery exploits fault-tolerant approaches adopted mostly by Backward recovery techniques

(Monitoring/Checkpointing). They commonly involve the creation of parallel backup replicas and checkpointing states so that they can switch their execution upon failure occurrence [16].

3.1.2 Other potential Fault Tolerant Techniques adopted at SaaS cloud service levels

Other potential techniques that could be adopted at a different service level are the following: SCOP - Self Configuring Optimal Programming, developed by Bondavalli, Di Giandomenico, and Xu [17]. Provides with a scheme in optimizing dependability and efficiency, throughout the exploitation of a flexible redundancy architecture, by adjusting at runtime. It exploits redundancy like NVP previously described, growing syndrome space for information gathering and result selection. Recovery Blocks (RcB) [18] scheme consists of several variants run from an executive and approved from an acceptance test. Sometimes real-time implementations add to the scheme a watchdog timer (WDT). From this technique, we can state that first, the RcB tries to ensure the AT the primary alternate through a try block. If the first one fails, the others are tried until a successor, a final failure is achieved. If a WDT is added, then the alternates are tried until the deadline for retrials is not expired. Various variants of the technique exist depending on the scenarios. The Distributed Recovery Blocks DRB combines distributed and parallel processing and recovery blocks [19]. The technique is applicable at both the software and hardware level. The technique exploits a pair of self-checking processing nodes or computing components structured as primary shadow pair, resident on different network nodes. The AT (Acceptance Test) ensures a result confirmation on the distributed nodes. Similar implementations and concept such as Exponential Backoff [20] are currently used from different cloud providers such as Google etc. However, adopting such approaches at the cloud service level and not just the end-user level would mitigate a lot of transient cloud issues. Thus, cloud systems operating SaaS adopting RtB techniques as proved later in the paperwork would ensure increased reliability of their services.

3.2 How the techniques can be integrated into the proposed framework across different cloud service

Most cloud services today exploit several programming languages in a distributed collaborative environment of virtual machine servers. Thus, this makes more natural the process of implementing the different fault tolerant techniques in a unified framework at the different service level as proved from the previous case study as shown in Table 2.

However, the same or more techniques can be exploited at a different service level. The framework implementation should also rely on the common programming languages supported by different service providers at PaaS level. Table 1 shows that most famous providers support almost the same programming languages. Thus, a generic framework development could be initiated with the prospect of expanding its programming language support.

Table 2: Fault tolerant technique applied at different cloud service level

Suggested Fault Tolerant Technique	SaaS	PaaS	IaaS
Design Diversity (NVP, RcB)		x	x
Data Diversity (RtB)	x	x	
Temp. Diversity combined with De sign Diversity (RcB)		x	x

3.3 Cloud based IDE environments

There have been few but essential initiatives related to moving development environments on web browsers. Some of them were specifically related to particular needs (such as embedded computing and software development [21] others more general [22]. One of the first was founded since 2010 by providing necessary features for code highlighting and more advanced related to cloud code deployment for the major providers such as AWS (Amazon Web Services), Microsoft Azure, Google App Engine. Only new environments have, however, tried to fully integrate the IDE with the cloud SDK by facilitating not just the development, testing and debugging but also the deployment process. Some of them worth mentioning in detail are Cloud 9 and Condevy [23, 24]. They are not the only ones operating in the market. Mainly for educational purposes, we can also mention others such as CloudForge, CodePlex, Compilr, jsFiddle, Eclipse Orion, CodeAnywehere, Coding, etc. Most of them are offering different integration levels with editors, documentation, compilation, versioning service such as Git, GitLab, etc.

1) Cloud 9: Founded in 2010 proves the concept of moving IDE oriented services to the web. Thus, code development and accessibility are made easy enough and relies on browser development. Acquired from the Amazon Cloud Services, it relies on the web EC2 Operating System or other web-oriented technology such as Aurora file system and the RDBMS for database management. Team collaboration is made easy by full integration of repositories such as Github, Bitbucket with the online shared workspaces [23].

2) Condevy: Condevy was initially launched in 2012 and has become since then a modern cloud-based development environment. Developers can work with shared, scalable, and distributed workspaces, which result in better compilation and deployment times. The environments are commonly set up as sandboxes, where the entire necessary configuration is prearranged so that the development process can focus primarily on the final application. This has become since yet the standard approach for Dockers exploitation, commonly done behind the scenes also from Condevy [24].

3.4 Shortcomings

Most of the presented solutions apart from Cloud 9 and Condevy have been adopted from educational institutions for students. Some of the major cloud providers have however become aware of the fact the benefits of services offering the development environment as a whole. The authors in [15] have correctly stated the issues arising from not adopting a development environment

for cloud applications from companies. Even though they claim that developers prefer the desktop environment, the inconvenience is apparent when it comes to locally configure the development environment, maintaining, updating, and deploying code in real time. Moreover, most monitoring and debugging make sense to be fully integrated with the development process as part of the core services offered from the cloud. Finally, the authors state also that new models are necessary to be adopted either with existing technology and programming languages or with new emerging ones. Following sections shall discuss how RIDEaaS can act as a new service complementary to the previous solutions addressed so far, with all the underlying economic benefits when adopting scalable business processes in the cloud [25]. Two business models are proposed based on resource and cost management for Small Medium Enterprises (SMEs) adopting cloud services and application development, as mentioned in [26].

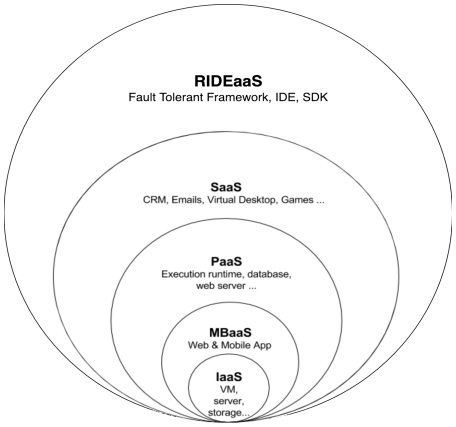

Figure 2: RIDEaaS Multilevel Service Layer

4 RIDEaaS Model and Framework

Most of the previously mentioned techniques have been adopted with an ad-hoc approach to different cloud architectures. However, since many of them address similar issues for different cloud vendors than a framework at a cloud service level could be the right approach for future applications that have High Availability/Reliability requirements. Figure 2 describes a possible cloud service offering Reliability as a Service (RIDEaaS). From the figure can be noticed the two possibilities of applying fault tolerant techniques at different cloud services, as well as providing the end user with application-level Framework that would ease the process of integrating high reliability in cloud application development.

4.1 RIDEaaS Platform

The techniques applied might be at different service layers. Part of the techniques such as Retry Blocks (Data Diversity) can be applied at PaaS layer while others might also be applied at different service layers in parallel (e.g., Design diversity and N-Version Programming). Some of the techniques are already available at IaaS layer such as storage redundancy, backup. The cloud service provider and the end user should have the freedom of adopting and choosing the service level where to apply the techniques, with minimum effort. Cloud developers should be supported by a framework, which they can exploit to build highly reliable cloud applications quickly. Such a framework is described in the next section. Based on the previous discussions we propose that the model should include several entities: Browser-based SDK fully integrated with the different service layers (PaaS, IaaS)

and their REST API-s or client libraries Browser-based IDE encapsulating the SDK functionalities within the platform Development tool that fully exploits the pay per use or pay as you go model Coding Synchronization through Versioning or Agile Environments (GitHub, Jira, etc.).

Developers can also use other offline IDE-s with their common environment and synchronization is made from collaborative repositories Integrating the development environment platform in the cloud and treating it as a usual service layer shall provide end cloud users not only with an online IDE, which can scale at demand, but also provide development teams with better upfront cost evaluation and help them focus on fast and efficient coding meeting project deadlines, rather than setting up beforehand collaborative environments. Community-based open source coding will be facilitated even further by bringing together on one common cloud platform and adopting new coding approaches such as coding contribute on demand (Pay as you go Coding and Code on Demand) or autonomous coding techniques relying on machine learning algorithms. The IDEaaS can be adopted at business rules and corporate specific requirements related to policies, privacy, and copyright protection, by intrinsically adopting public, private, or hybrid cloud solutions. Figure 3 represents our proposed platform. We can observe the cloud layers adopting the development services provided by RIDEaaS. A particular focus is put on the PaaS and IaaS, which are the primary entities exploited within the cloud infrastructure for development purposes. Optimizing their usage have been both virtualizations at VM or OS level (Dockers). However, the model also supports the possibility, when applicable, to be exploited by SaaS layer where different applications can use the API provided by IDEaaS in order to integrate their development or code testing environment. For the first time the model proposes that the cloud providers incorporate the development environment within their infrastructure; Not only making development and deployment more efficient but also bringing together cloud features into a common ground for fast and agile software management. Most cloud providers may adopt the IDEaaS service either as a third

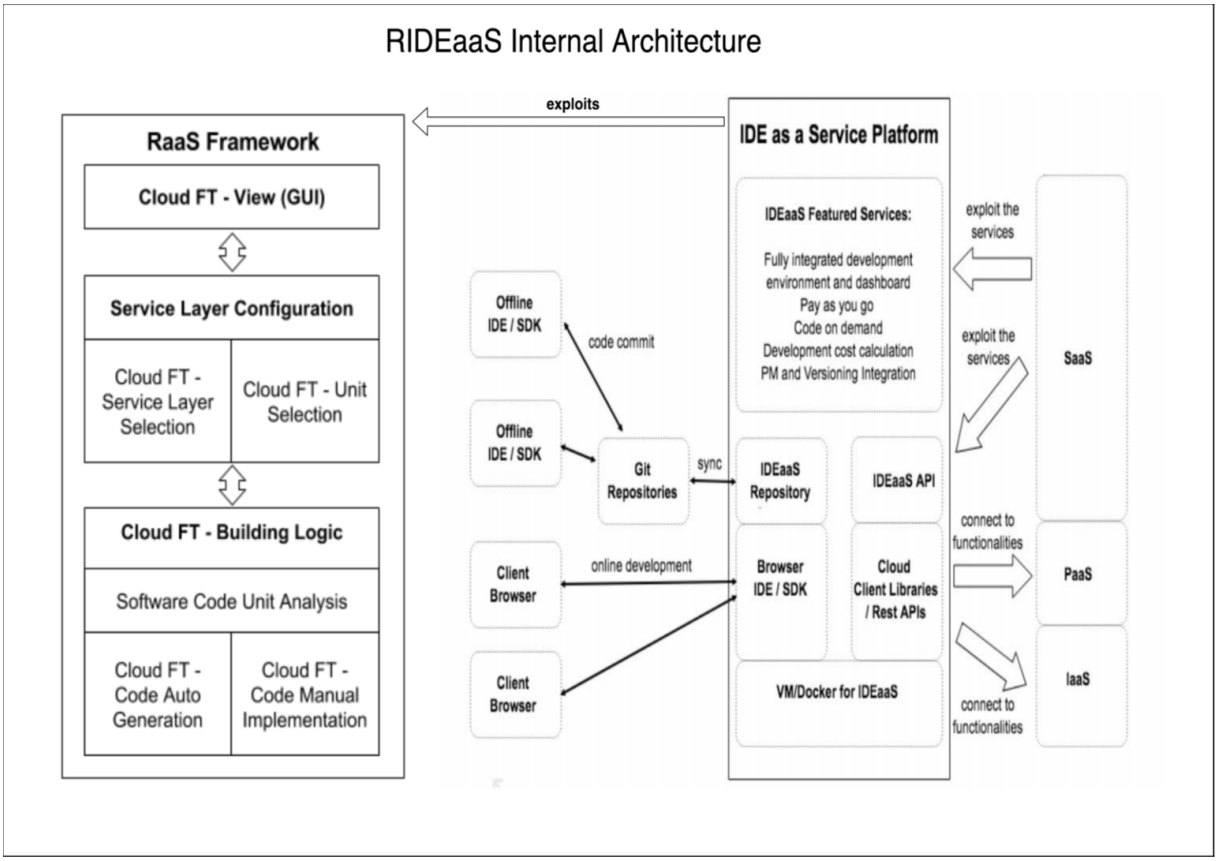

Figure 3: RIDEaaS - Platform

party cloud application at a software layer or offered as an integral part of their infrastructure/platform. The second model would make open source contribution and development more convenient and convince more cloud-based businesses to develop their products efficiently. Other providers adopting already some similar approaches such as Amazon or the ones that have not yet undertaken the cloud-based development steps such as Azure might still benefit from the model. Proof of the concept and model adoption is presented in the following section, where actual development and integration of the IDEaaS service has been performed for one of the major cloud providers GCP (Google Cloud Platform).

4.2 Reliability as a Service (RIDEaaS) Framework

In order, for the end user (cloud clients), to efficiently exploit the different fault tolerant techniques a framework solution is proposed especially for PaaS layer so that that coding techniques can be directly applied to the software components implemented from the developer. The latter can manually implement the code or in some cases, rely on code auto-generation, both part of the framework. It will initially imply development overhead, which will, especially in the production model of the cloud software system. The other two service layers might be maintained from the framework especially in terms of manually enabling or not of the different fault tolerant techniques along with the PaaS and IaaS layers. In order to automate the process most of the techniques should be fully integrated with proper interfaces (APIs) within the cloud applications. Figure 3 provides a possible overview of the framework as part of the RIDE internal architecture of services. From the framework we can observe that after first deployment the developer has GUI based option to select from the type of fault tolerant technique to be applied on different service layer for different functionalities translated into unit entities depending on the layer (at RaaS level this can be easily identified as coding unit, the other levels are harder to address and specify a proper unit entity). The first step of the framework is to analyze the unit current state of development, and then suggest for possible manual updates or even auto-generate code when common solutions can be automatically updated based on the previously mentioned techniques.

5 Case Study: RIDEaaS Based on GAELauncher SDK and IDEaaS

The section presents the actual migration of the Google cloud python SDK-s libraries from their desktop environment to the

Google App Engine (GAE). The deployment process, which was previously based on the desktop application, has been made possible to be completed from a GAE VM. Moreover, a simple code interpreter has also been developed, offering the possibility to implement and link the code getting deployed directly on the cloud. GAE Launcher also can add synchronize with local code which can be uploaded and part of a GitHub repository.

More features can be added in the future, and the development could become part of the cloud infrastructure by playing a key role and not just act as an application running on Google PaaS, which can later be part of the SaaS application pools. More functionality from gcloud tool could be part of a more sophisticated development environment fully integrated into the cloud.

5.1 Migrating GAE Desktop SDK to the Cloud

The migration based on the concept stated in [29] of the Google App Engine Launcher python SDK from desktop to the GAE environment has involved the modification of several SDK libraries and their adoption with client libraries [13]. In particular, the appcfg.py has been adapted to run on the GAE VM (Virtual Machine) environment. Common modifications relate to limitations of the cloud environment such as lack of file system, spawning of processes, etc. Other cloud services such as Google Cloud Storage and SQL have been used instead of local filesystems. Moreover, the application has to run as a single web service process. The major changes to the code are related to modifying the appcfg.main(argv) [19,20].

Development server would not bring added value since the virtual cloud environment provides enough resources to handle development and production scenarios. While application staging could be of better profit. Other major features have been integrated such as Browser-based python development environment with Django framework Github project synchronization. Further integration with cloud-based PM tools such as Asana and Jira are part of the features that would contribute into new cloud-based coding concepts such as: Pay as you go during project development Code on demand, increasing outsourcing possibilities of project functionalities Project cost estimation Optimization of collaborative and community coding for large scale projects relying on cloud platforms.

5.2 Adopted Architecture

The adopted architecture involves different entities in collaboration with the current cloud services. We present an overview of the integrated services in Figure 3. We can observe the integration of the features for efficient cloud development and deployment. The current architecture can be easily integrated into the current cloud console by featuring more infrastructure services such as Storage, Compute, StackDriver (logging, debugging), etc. This allows several users to exploit an integrated development platform where many features are part of the cloud but development oriented. Integration would mean adoption to the proposed platform, which can be part of a new core cloud service (RIDEaaS). To achieve so, we have further developed the IDE to fit with the cloud console debugging environment and implementation pipeline (development, testing, debugging, and deployment).

The reference in [27] represent the current state of development. Apart from the business concept discussed in [10], new concepts arise from RIDEaaS, such as High Availability Coding. The following section discusses this new opportunity and how it fits our previous proposals.

5.3 Business Models for New Cloud Based RIDE Services

Our previous proposal [10] IDEaaS included two significant economic model opportunities:

1) Pay as you go Coding (PaygoC)
2) On Demand Coding (ODC)

Both models shall help cloud software managers optimize their PM flow, outsourcing, and costs along with larger collaborating communities. While large enterprises shall never again suffer from collaborative development overheads for configuring their shared coding environments.

1) PaygoC - Pay as you go is not a new business model to the cloud but when it comes to IDEaaS it can be directly adopted for coding utilizing resource hours. Thus, making it easy to evaluate development costs without any infrastructure or licensing upfront costs. Portable operating systems (Dockers) might not be required since many of the environment setups are acquired as API services or libraries. Of course, the upfront configuration might be needed per project basis, but any further configurations on the online IDE will be shared among all the project developers. Customer pricing might rely on per use basis, typically charged by the hour, of the development environment and exploitation of other existing services from the cloud console.

2) Model 2 (ODC) - On Demand Coding permits outsourcing services to be facilitated and optimized whenever coding expertise is needed on demand. The significant opportunities that might be risen from the model could integrate employment platforms with cloud development environments.

Making it possible to develop large open source development cloud-based communities fully integrated with existing freelancing platforms. Cloud based outsourcing would definitively provide better security, development policies, project cost evaluation, and avoid over budgeting. Figure 4 describes the PaygoC and ODC business model canvas. We can observe from the proposed business model the different potentials arising from integrating several new actors in the cloud development in one common place. The benefits of the two models have been discussed in [10], where billions in profits could be derived from their exploitation from many cloud providers. RIDEaaS adds a new model High Availability Coding HAC, which fits well with the lean canvas model in Figure 4. The model suggests that for particular components, the developer can request high availability approaches to be manually developed from him or auto-generated from the cloud RaaS framework. Some of the key techniques adopted are the ones related to fault tolerant techniques proposed in the background section. The section has proven both the viability of the model as well as the possibility to become a key

factor in cloud computing services. Next section discusses further opportunities arising from the proposed model.

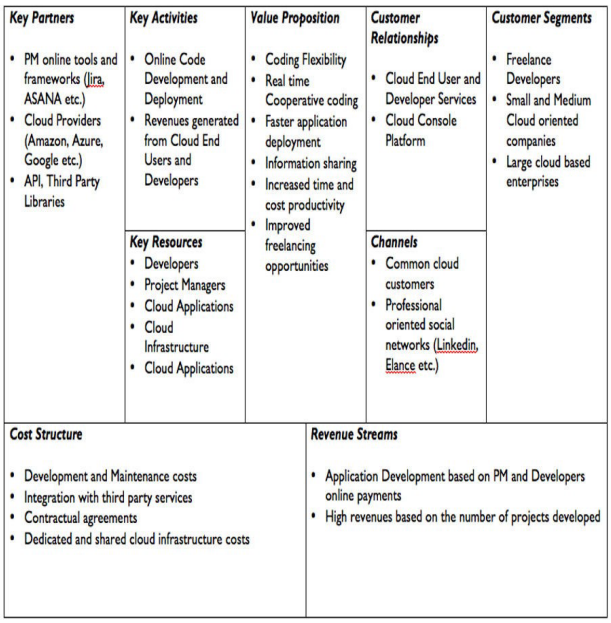

Figure: 4. PaygoC and ODC business model canvas

6 Discussions and Conclusions

In this work, we propose a new cloud service RIDEaaS, which will have a significant impact on how cloud development will be perceived in the near future. Most of the Cloud SDK-s today for popular programming languages are becoming hard to manage and somehow redundant with the growing needs of programs for more flexible and cooperative environments and having new business models adopted and new development services added to the cloud infrastructures shall provide increased growth of interest in exploiting cloud applications at the enterprise level. The new RIDEaaS service model suggests that the development environment could be offered as other services currently are within the cloud. These concepts have been discussed before either as IDEaaS. What RIDEaaS adds to the previous models is the full encapsulation in one service of the IDEaaS with RaaS, since they have a complementary nature. This leads to one new business model HAC, which will allow cloud end users and developers to determine the critical components in their software and apply fault tolerant techniques to them. This will impact the cloud performance and will allow developers a higher degree of control over the availability of their software. We propose in the future to fully develop the RIDEaaS and to adopt it with other concerns related to security with a higher degree of control from the end users. Artificial intelligence could also be a future contribution in helping code development and analyzing the coding architecture in order to help make essential choices for the developers.

REFERENCES

[1] Arie van Deursen et al. "Adinda: a knowledgeable, browser-based IDE". In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2. ACM. 2010, pp. 203–206.
[2] Andreas Zeller. "The future of programming environments: Integration, synergy, and assistance". In: Future of Software Engineering. IEEE Computer Society. 2007, pp. 316–325.
[3] Randall Frost. "Jazz and the eclipse way of collaboration". In: IEEE software 24.6 (2007).
[4] William Voorsluys, James Broberg, and Rajkumar Buyya. "Introduction to cloud computing". In: Cloud computing: Principles and paradigms (2011), pp. 1–41.
[5] Code Run: http://www.coderun.com/ide/. 2019.
[6] Jam Jenkins, Evelyn Brannock, and Sonal Dekhane. "JavaWIDE: innovation in an online IDE: tutorial presentation". In: Journal of Computing Sciences in Colleges 25.5 (2010), pp. 102–104.
[7] JAVAWIDE: http://www.javawide.org. 2019.
[8] BESPIN: https://mozillalabs.com/skywriter/2010/10/01/bespin-0-9a2-released-skywriter-update/. 2019.
[9] Christopher Dabrowski. "Reliability in grid computing systems". In: Concurrency and Computation: Practice and Experience 21.8 (2009), pp. 927–959.
[10] Orges Cico, Zamir Dika, and Betim Cico. "Integrated Development Environment as a Service (IDEaaS) - Models and Architecture part of the Google Cloud Core Services". In: International Journal of Computer Applications 182.6 (July 2018), pp. 44–50. ISSN: 0975- 8887. DOI: 10.5120/ijca2018917534. URL: http://www.ijcaonline. org / archives / volume182 / number6 / 29770 - 2018917534.
[11] Orges Cico, Zamir Dika, and Betim Cico. "High Reliability Approaches in Cloud Applications for Business - Reliability as a Service (RAAS) Model". In: International Journal on Information Technologies and Security 9.3 (2017), pp. 3–18.
[12] GPCDBMosetti, Carlo Poloni, and B Diviacco. "Optimization of wind turbine positioning inlarge windfarms by means of a genetic algorithm". In: Journal of Wind Engineering and Industrial Aerodynamics 51.1 (1994), pp. 105–116.
[13] Tao Chn, Rami Bahsoon, and Abdel-Rahman H Tawil. "Scalable service-oriented replcation with flexible consistency guarantee in the cloud". In: Information Sciences 264 (2014), pp. 349–370.
[14] Kyoungho An et al. "A cloud middleware for assuring performance and high availability of soft real-time applications". In: Journal of Systems Architecture 60.9 (2014), pp. 757–769.
[15] A. Imran et al. "Cloud-Niagara: A high availability and low overhead fault tolerance middleware for the cloud". In: 16th Int'l Conf. Computer and Information Technology. Mar. 2014, pp. 271–276. DOI: 10. 1109 /ICCITechn.2014.6997344.
[16] Tsanko Alexandrov and Aleksandar Dimov. "Software Availability in the Cloud". In: Proceedings of the 14th International Conference on Computer Systems and Technologies. CompSysTech '13. Ruse, Bulgaria: ACM, 2013, pp. 193–200.
[17] Andrea Bondavalli, Felicita Di Giandomenico, and Jie Xu. A cost-effective and flexible scheme for software fault tolerance. University of Newcastle upon Tyne, Computing Laboratory, 1992.
[18] Algirdas Avizienis and John PJ Kelly. "Fault tolerance by design diversity: Concepts and experiments". In: Computer 8 (1984), pp. 67–80.
[19] KH Kim. "The distributed recovery block scheme". In: Software Fault Tolerance 3(1995), pp. 189–210.
[20] Exponential Back-off: https://cloud.google.com/storage/docs/exponential-backoff. 2019.
[21] Jurgen Hausladen, Birgit Pohn, and Martin Horauer. "A cloud-based integrated development environment for embedded systems". In: Mechatronic and Embedded Systems and Applications (MESA), 2014 IEEE/ASME 10th International Conference on. IEEE. 2014, pp. 1–5.
[22] Ling Wu et al. "CEclipse: An online IDE for programing in the cloud". In: Services (SERVICES), 2011 IEEE World Congress on. IEEE. 2011, pp. 45–52.
[23] CLOUD9: https://aws.amazon.com/cloud9/. 2019.
[24] Condevy: https://codenvy.com/. 2019.
[25] Rainer Schmidt. "Scalable business process enactment in cloud environments". In: Enterprise, Business Process and Information Systems Modeling. Springer, 2012, pp. 1–15.
[26] Eric Kuada, Kwami Adanu, and Henning Olesen. "Cloud computing and information technology resource cost management for SMEs". In: EUROCON, 2013 IEEE. IEEE. 2013, pp. 258–266.
[27] GAE Launcher: http://gae-launcher.appspot.com. 2019.