WSO2 Stratos: An Application Stack to Support Cloud Computing

Afkham Azeez: WSO2, Flower Road, 59, Colombo, 03, Sri Lanka
E-Mail: azeez@wso2.com
Azeez is a Senior Architect and Senior Manager at WSO2. He is an elected Member of the Apache Software Foundation (ASF) and also a Project Management Committee (PMC) member and committer on a number of projects at Apache Software Foundation. He is also the author of Clustering implementation for Apache Axis2. Azeez holds a B.Sc. and a M.Sc. degrees in Computer Science from the University of Moratuwa, Sri Lanka.

Dr. Srinath Perera: WSO2, Flower Road, 59, Colombo, 03, Sri Lanka
E-Mail: srinath@wso2.com
Srinath is a Senior Architect at WSO2 where he focuses on systems architectures. He has been involved with the Apache Web Service project since 2002 and is a member of Apache Software Foundation, a member of the Apache Web Service project, and a member of Apache Web Service Project Management Committee. Srinath is also a committer of Apache Axis, Axis2, and Geronimo, and a co-founder of Apache Axis2. Srinath holds a B.Sc. in Computer Science and Engineering from University of Moratuwa, an M.Sc. in Computer Sciences and a Ph.D. in Computer Sciences from Indiana University.

Dr. Sanjiva Weerawarana: WSO2, Flower Road, 59, Colombo, 03, Sri Lanka
E-Mail: sanjiva@wso2.com
Sanjiva is Chairman & CEO of WSO2 which he founded after having spent nearly 8 years in IBM Research where he co-authored many Web Services specifications including WSDL, BPEL4WS, WS-Addressing, WS-ResourceFramework and WS-Eventing. In recognition for his company-wide technical leadership, Sanjiva was elected to the IBM Academy of Technology in 2003. Sanjiva has been involved with open source for many years. Sanjiva was the original creator of Apache SOAP and has been part of Apache Axis, Apache Axis2 and most Apache Web Services projects. He is a member of the Apache Software Foundation. Prior to joining IBM, Sanjiva spent 3 years at Purdue University as visiting faculty, where he received his Ph.D. in Computer Science in 1994.

Paul Fremantle: WSO2, Flower Road, 59, Colombo, 03, Sri Lanka
E-Mail: paul@wso2.com
Paul Fremantle is Co-Founder and CTO of WSO2, where he oversees and manages WSO2’s overall product strategy. Prior to founding WSO2, Paul was a Senior Technical Staff Member at IBM where he had a variety of roles for nearly 10 years. At IBM, Paul created the Web Services Gateway, and led the team that developed and shipped it as part of the WebSphere Application Server. Paul also worked in IBM Global Services for 3 years, providing technical and business consultancy around the Internet and e-business. Before joining IBM, he was a consultant at ZS Associates, providing analytical sales forecasting consultancy to the Pharmaceutical market. Paul has a MA in Mathematics and Philosophy and a M.Sc. in Computation, both from Balliol College, Oxford University.

Selvaratnam Uthaiyashankar: WSO2, Flower Road, 59, Colombo, 03, Sri Lanka
E-Mail: shanker@wso2.com
Shankar is a Senior Architect and Senior Manager at WSO2 where he focuses on Axis2/C, WSF projects, Cloud platform and interoperability. He is a Member of the Apache Software Foundation (ASF), a member of Apache Web Service Project Management Committee (PMC), and a committer of Apache Axis2 and Apache Stonehenge. He holds a B.Sc. in Computer Science from the University of Moratuwa, Sri Lanka.

Samisa Abesinghe: WSO2, Flower Road, 59, Colombo, 03, Sri Lanka
E-Mail: samisa@wso2.com
Samisa is VP of Engineering at WSO2 and has been with the company from nearly its inception. Prior to taking over all of engineering, Samisa headed the WSO2 WSF/C and WSO2 WSF/PHP projects. He has been a committer of the Apache Web Services project, since May 2004, and initiated the Apache Axis2/C project and is a Member of the Apache Software Foundation. He has also published two books: RESTFul PHP Web Services, and PHP Team Development. Samisa holds a B.Sc. in Engineering from the University of Moratuwa, Sri Lanka, as well as an M.Sc. from the University of Colombo School of Computing, Sri Lanka.

Keywords: C.2.4 [Computer Communication Networks]: Distributed Systems—Client/server; D.2.11 [Software Engineering]: Software Architectures—Service-oriented architecture (SOA)

Schlagworte: keywords in german
Abstract
Cloud computing has heralded many advantages including self-service, elasticity, pay as you go, improved accessibility to computation resources, and ease of deployment and deployment automation. However, to extend those advantages to a larger audience, higher-level abstractions such as Software as a Service (SaaS) or Platform as a Service (PaaS) are needed. This paper presents Stratos, which is a Platform-as-a-Service based around the principles and design of Service Oriented Architecture (SOA). Furthermore, we introduce the concept of Cloud Native attributes, which we argue are properties essential to extend the benefits of the underlying Cloud to PaaS and SaaS users. We present Stratos as a Cloud Native PaaS offering, discuss the architecture that enables Cloud Native attributes, and discuss scenarios where Stratos can be useful to the end user.

Zusammenfassung
abstract in german
1 Introduction

Cloud Computing [1], the idea that computing power, storage, etc. can be rented on demand, used and released without worrying about where they are physically located, has shown much promise and already has many real-world applications. Although there is no universal consensus on what the Cloud is, “Cloud” is often categorized into three forms:

- Software as a Service (SaaS): applications that are available for use without having to buy, deploy and manage.
- Platform as a Service (PaaS): middleware that offers a computing platform which leverages an underlying IaaS and is the basis for building SaaS applications.
- Infrastructure as a Service (IaaS): low-level computing resources such as CPU, memory, storage and networking.

Proponents of Cloud often cite self-service, elasticity, pay as you go, improved accessibility to computing resources, and ease of deployment automation as its key benefits (e.g. Carr [8]). Those benefits have far reaching consequences. Let us look at a few.

Cloud makes large-scale computing resources accessible to average computing users, which were previously only accessible to large organizations, thus democratizing computations. For example, a small research group can now run a reasonably large analysis in the Cloud, or a small business can store its data in geographically distributed replicas, thus achieving high reliability.

Unlike before, Cloud virtualizes hardware and lets users provision or de-provision them programmatically. As a result, Cloud has increased the reach of automation. This enables organizations to do changes or create new solutions faster, thus enabling them to react fast and adapt themselves to changes in their environment, which is a crucial competitive advantage.

One important aspect of this is elasticity. For instance, capacity planning is done for the peak load, not for the average load where the latter is often only a fraction of the former. Consequently, most organizations have a significant portion of their hardware idling throughout the year waiting for the peak load. However, Cloud enables organizations to autoscale their systems—that is, to expand when there is high load and contract when there is low load.

However, most importantly, Cloud enables organizations to outsource their non-competitive functions where maintaining and operating infrastructure is a significant portion of the IT budget. For example, according to Saat et al. [2], IT labor represents about 70% of IT costs. The ideal solution to this problem comes in the form of Software as a Service (SaaS) where a part of the system is outsourced to an external Service. For an example, Salesforce provides Customer Relationship Management (CRM) as a Service, Google Apps provide office software suite as a Service, and Dropbox provides storage as a Service.

However, such packaged solutions are available only for well-defined application categories like CRM, Email etc. In order for organizations to achieve competitive differentiation in an IT-dominated world, it is crucial that they write and manage their own business-critical systems. Platform as a Service (PaaS) provides a middle ground between these two interests where PaaS infrastructures let users build their custom applications and host them in the Cloud.

Although several PaaS offerings (e.g. Google App Engine and Microsoft Azure) are available they do not employ many of the lessons learnt from Service Oriented Architecture (SOA) [4] or SOA platforms. SOA has become the defacto standard for building complex, distributed and integrated enterprise systems and has achieved wide market adoption. Therefore, providing a SOA platform as a Service within the Cloud will enable users to outsource most enterprise applications, thereby reducing costs. More importantly, SOA is a design pattern for building large-scale applications and composite applications that can support evolutionary lifecycles, where SOA enables parts of the applications to be updated independently of each other, under a governance model. This design pattern is very important within the Cloud and therefore a SOA-based PaaS is an important design point for Enterprise applications in the Cloud.

This paper focuses on PaaS architectures for SOA. It makes two contributions. First, we propose a model to evaluate PaaS architectures based on what we call Cloud Native properties. Second, we present WSO2 Stratos, which is a Cloud-based platform for hosting SOA applications and artefacts as a Service (PaaS).

While providing such higher levels of abstractions, those SOA PaaS offerings should extend the benefits of the Cloud to application owners. We have identified six properties that are necessary for a cloud platform that extends the benefits of the cloud to the end user. We call them cloud-native properties and they are distributed/dynamically wired (run in many machines), Elastic, multi-tenant, support self-service, Granularly billed and metered, and Incrementally Deployed and Tested.

Stratos provides most aspects of SOA like Web Service Hosting, Enterprise Service Bus, Service Registry, and Identity and authorization as a Service. Stratos’ programming model uses the same SOA programming model, which means that users can deploy in Stratos, the same artifact they have in their local machines.

Furthermore, the SOA programming model has evolved over years, and developers have grown accustomed to it. Therefore, we have taken the best effort to retain the programming model used in non-Cloud
setups within the Cloud setup as well, which reduces the burden on end users. Furthermore, Stratos hosts those applications in a multi-tenanted environment and supports auto-scaling together with other Cloud-native properties out of the box.

The rest of the paper is organized as follows. The next section discusses the cloud-native properties and PaaS architectures and Section 3 describes few motivating usecases for Stratos. Section 4 describes the architecture of Stratos and how cloud-native properties are realized. Section 5 compares Stratos with other PaaS offerings. Finally, Section 6 concludes the paper.

2 Cloud-Native Properties and SOA Cloud Framework

Let us look at each of cloud-native properties in detail.

1. Distributed/Dynamically Wired: end user application can run on multiple machines and can discover and dynamically wire its parts at runtime

2. Elastic (Uses the Cloud efficiently): the applications develop using the platform can scale based on load incurred on the application. This includes monitoring the load, allocating more computation units when the load is increasing, deallocating computation units when the load is decreasing, and managing that process to minimize the cost.

3. Multi-Tenant (Only costs when you use it): Multi-Tenancy enables a single server to support multiple tenants (users) while providing the user experience that each has its own server. Chong et al. [13] discusses Multi-Tenancy in detail.

4. Self-Service (in the hands of users): decentralized creation and management of tenants and automated Governance across tenants. This enables pay-as-you-go, because, this model enables users to scale up without waiting for a manual process.

5. Granularly Billed and Metered (pay for just what you use): Users only have to pay for what they use, not a fix rental.

6. Incrementally Deployed and Tested (seamless live upgrades): supports users to change, test, and update their applications within the PaaS infrastructure. Furthermore, the platform should enable users to run multiple versions of the same application side by side, and this will simplify the rollout cycle of applications by a great extent.

Service Oriented Architecture (SOA) is based on Services that are working together in a loosely coupled manner. A middleware platform for SOA includes support to develop, run, manage, and consume Services, as well as support for Service mediation, Service discovery, and add-on functionalities like security, transactions etc. As shown in Figure 1, those functionalities are provided by few servers. For example, Web Service Application Server (WSAS) provides support for Web Service development, hosting, and management, Enterprise Service Bus (ESB) provides Service mediation, and Registry provides data storage and governance. Each server often provides a web-based console, which enables users to create, configure, and manage SOA artifacts.

The idea of a Cloud-based SOA platform (Stratos) and its Cloud-native properties are best described through a usecase. Let us assume that a user has a system that consists of three Web Services, and he needs to mediate messages received by those Web Services through an Enterprise Service Bus and wants to compose those Web Services together as a workflow through a Workflow Engine.

In a conventional setup, the architecture would include several servers—a WSAS, ESB, and a Workflow Engine—that runs a Web Service, Service mediation logic, and a workflow, respectively. User would setup servers, configure them to talk to each other, and deploy Services etc. However, in contrast, WSO2 Stratos provides a one platform to support the entire SOA, where users can log into a Web console and create required SOA Services (servers).

For instance, in this usecase, the user logs into the Web console and requests WSAS, ESB, and BAM as a Service, where Stratos creates a virtual instance of each type. When the user clicks on a Service (e.g. ESB), Stratos presents a UI similar to the UI of the ESB standalone server, where he can add ESB configurations the same way as the standalone product. Consequently, a user can deploy his Web Services and a workflow into Stratos just the same way he would deploy those artifacts in the standalone servers. However, unlike with standalone deployments, the Stratos-based deployment carries several crucial benefits.

If the user’s system is getting a very high load that cannot be handled through a single machine, he has to manually set up new servers and configure clustering, thus setting up a large-scale deployment. However, WSO2 Stratos scales up and down the servers based on application load, while allocating and releasing resources as needed. The user is spared the burden of managing a large-scale deployment. Furthermore, in the Stratos implementation, each Stratos node supports multiple users (tenants) while providing isolation and maximizing resource usage. Therefore, multiple users may share many resources, resulting inefficient resource sharing. Also, with the Stratos architecture, SOA artifacts incur only a very small cost when not in use. In
addition, users can expand or change their systems without any help from the hosting party, which gives them the benefit of a self-service model. Users are only billed for what they use, based on the storage, bandwidth, and the number of requests received by Services. Moreover, users can run multiple versions of SOA artifacts side by side and can easily revert to an older version if needed. This allows users to incrementally change their systems.

To summarize, Stratos provides a one-stop place to deploy and run any SOA artifact, while keeping the standalone programming model and user experience intact for the most part. The Cloud-native properties make sure that the benefits of the Cloud such as self-service, elasticity, auto-scaling, and pay-as-you-go are extend to the users of the PaaS platforms like Stratos.

3 Motivating Use Cases

To understand Stratos and how it can make a difference to the end user, lets us look at few uscases of Stratos.

In matter of few clicks, Stratos enables users to create their own complete SOA platform comprised of Services like Web Service container, ESB, Workflow Engine, Registry, and BAM. Users may have only a specific Service (e.g., Registry), pick and choose few Services, or expand current Services later. Each Service is integrated with the other, liberating the users from having to configure and integrate them. Importantly, Stratos supports normal SOA artifacts, thus, keeping the SOA programming model intact. For an example, with Stratos, a user can deploy his existing Web Service archive without any changes, and even the User Interfaces in the Stratos Web console have the same functionality as well as the look and feel of the normal standalone version. Therefore, Stratos provides seamless migration with a flat learning curve.

Stratos supports auto-scaling for applications by monitoring the load and automatically allocating and deallocating resources. This enables optimal use of computational resources and removes the need for complex clustering configuration and management. Furthermore, auto-scaling enables the end-user to start small and to scale up gradually while only paying for what they use. Since resources are allocated on demand through multi-tenancy, the cost of having an application deployed in Stratos is minimal. A rarely used application will actually use very little resources, and the owner will be only billed accordingly. Therefore, it is an ideal setup for startups and small & medium businesses with varying computing demands. Also, through self-service, Stratos allows users to change their setup within minutes, enabling them to react to changes faster. This is a valuable competitive advantage.

Stratos can be deployed in both public and private Clouds. In addition to well-known Cloud benefits, multi-tenancy also provides a natural setup to support different parts (e.g., departments) within an organization. It provides partitioning while enabling high-level of resource sharing across different parts of the organization.

Stratos gives organizations that already use SOA artifacts, an opportunity to outsource their SOA infrastructure and to focus on their key functions. Although the same is true for most other Cloud offerings, Stratos enables outsourcing at the SOA level, which is a much higher-level of abstraction and hence can provide more cost savings. By securely pooling organizational functions with others, Stratos enables cost saving through economics of scale.

Furthermore, Stratos includes an inbuilt Business Activity Monitor (BAM), which monitors applications and lets users drill down to details. Often BAM is an after-thought in most SOA deployments. However, within Stratos, once enabled it automatically integrates and monitors your applications. Organizations can use this to drive their decisions and to be notified about changes. Through BAM, Stratos enables access to billing and metering information collected from end-user applications, which enables application developers to charge their users based on the usage. Stratos can be an ideal environment to sell SOA-based applications, which can give rise to a culture of creating and selling Web Services, making the vision of Services market a reality.

Finally, Stratos is a great tool for learning and for users who want to play with SOA. It lets users to skip download, install, configure path and directly play with SOA. It makes Web Service hosting more accessible. For example, a user may choose to implement the backend of his web site as a Web Service and host the Web Service in Stratos. Furthermore, Stratos provides an ideal environment to collaboratively build systems with geographically distributed teams, by sharing Web Services and other SOA artifacts.

4 Stratos Architecture

WSO2 Carbon is a component-based framework for building SOA servers, and it supports an end-to-end SOA platform. WSO2 Stratos provides the functionality of Carbon “as a Service”, and it is built using the Carbon platform. Carbon builds on OSGI and provides services like Web Service hosting, security, data storage, and UIs as core Services. Carbon has built each concept like Web Service hosting, mediation, Service orchestrations, logging, and Service registries etc., as carbon components. End-users can define their own servers (product) by composing a subset of features together. For an example, a user can either extend an existing server (e.g., add Business Activity Monitoring Support to the ESB) or define his own product by mixing and matching different features. Fremantle et al. [6] provides a detailed
discussion on the Carbon Platform and its implementation.

Stratos architecture has three aspects: multi-tenancy support, how Stratos nodes are arranged in a scalable and flexible topology, and how it supports elasticity.

4.1 Multi-tenancy

Stratos is comprised of server nodes where each can support multiple tenants within the same node while providing isolation between tenants through multi-tenancy. This enables Stratos to avoid assigning a physical server for each tenant. Starting a VM instance for each tenant is also possible, but that runs a different OS instance for each tenant whereas with multi-tenancy, isolation is achieved at the middleware level. Hence, multi-tenancy provides greater sharing. Azeez et al. [7] discusses this and presents Stratos multi-tenancy architecture in detail. Stratos supports multi-tenancy in three parts: storage, security, and execution.

Security: Stratos has extended the security and permission model of WSO2 carbon to support multiple tenants.

Storage: The Registry component enables users to store data products, artefacts and configuration items in a hierarchical tree, and it is available across the platform within every product. The Registry supports multi-tenancy by introducing a new tenant ID row to resource tables of the underlying database. Registry API provides isolation by checking all the calls and authorizing them based on multi-tenancy security models. This gives each tenant the user experience that he has his own registry instance although many users are in fact sharing a single instance.

Execution: Each Carbon component uses Axis2 as the core, and multi-tenanting Axis2 has extended the multi-tenancy support to most components. By design, Axis2 keeps the state and execution logic separate where all the states of an execution are contained within a context hierarchy. We have enabled multi-tenant executions by creating a different context hierarchy for each tenant and using the correct context when an execution is aimed at a specific tenant. Since the rest of Axis2 is stateless, the execution code does not need any changes. (see [7])

4.2 Scalability Architecture

Since tenants are completely isolated from each other, Stratos can use distribution techniques to scale up the architecture in the following manner. Information about each tenant is stored in the registry. Therefore, we have scaled up Stratos in two parts: scaling up the registry and scaling up executions.

To scale up, Stratos statically partitions tenants across a group of registries. However, the tenant stor-
cific cluster group that handles the target tenant of the message. The cluster-group is fronted by a service-aware load balancer, which redirects the request to a specific cluster that supports the specific type of Service(server) for the message. If the target tenant is not active, the cluster-group load balancer assigns the tenant to a cluster before forwarding the message and instructs the cluster to load the tenant. After loading, each node in the cluster supports the loaded tenant. If all clusters are full, Stratos will create a new cluster and assign the tenant. After that, the cluster-group load balancer forwards the request to a clustered load balancer, which sends it to a node to process the request. The cluster load balancer supports sticky sessions, so that requests from the same client are sent to the same back-end node, allowing him to maintain a session.

4.3 Auto-Scaling

Each cluster is elastic, or in other words auto-scales. That is, the cluster load balancer monitors the load of each node and distributes the load across nodes, and when all nodes are overloaded, it adds new nodes. Conversely, if the load has gone down, it shuts some nodes down. Furthermore, the cluster-group load balancer keeps track of the number of requests received by each tenant and when a tenant is inactive for an extended period, it instructs the corresponding cluster to unload the tenant. However, holding a tenant only occupies memory, and therefore, we only unload a tenant sparingly.

To summarize, Stratos statically allocates storage to tenants, but allocates computation nodes dynamically as needed. Hence, Stratos can truly support a “pay for what you use” model.

Applications in Stratos can use WS-Discovery to discover other services and wire themselves. WS-Discovery implementation in Stratos uses a well-known address-based schema. Furthermore, underline group communication keeps cluster nodes up-to-date about other nodes in the same cluster.

Stratos lets users run multiple versions of the same Web Service side-by-side. It does so by deploying multiple Web Services within Axis2 with their versions appended to the name, and then redirecting Web Service requests to the correct Web Service version. Axis2 includes an extension point called “message dispatcher”, which enables users to extend the logic that redirects requests to Web Services. We support versions using a new Axis2 Dispatcher that routes service requests to the current active version.

5 Related Work

As described earlier, Azeez et al. [7] discusses multi-tenancy support for Stratos in detail, and that compares and contrasts Stratos multi-tenancy support with ear-
provides worker nodes, which can be used for batch-like applications.

Currently Stratos assigns tenants to the next available resource and does not optimize tenant assignments. However, optimizing tenant assignment will be a useful addition. For example, Fehling et al. [16] discusses such an optimization technique that uses simulated annealing.

6 Summary

This paper presents WSO2 Stratos, which is an SOA-based PaaS architecture that enables end-users to host their applications and SOA artifacts in the Cloud. We identify Cloud Native properties: Distributed/Dynamically Wired, Elastic, Multi-Tenant, Self-Service, Granularly Billed and Metered, and Incrementally Deployed and Tested, and argue that a Cloud platform that supports Cloud Native properties will extend the benefits of the Cloud fully to its end-users.

We present the architecture of Stratos and discuss in detail how that architecture enables Cloud Native properties. While keeping the SOA programming model intact, Stratos lets users deploy SOA artifact-like Services, Mediation logic, Workflows etc., in a virtual server provided by Stratos. Stratos monitors the load received by those applications, and automatically scales them up and down by allocating and deallocating resources. Furthermore, as we discussed in Section 4, Stratos allocates resources to those artifacts as late as possible, and therefore, a deployed application that only receives a minimal load will incur very small cost, thus enabling granular billing and metering as well as pay for what you use.

Finally, Section 3 discusses few usecases for Stratos and their potential impact. In summary, it will enable elastic, pay-as-you-go hosting of SOA artifacts, which could potentially impact the way SOA is deployed and used. Stratos is available publicly at http://cloud.wso2.com/ and is free within storage and computation allocation limits. Paid accounts, which allow higher levels of resource usage will be made available soon. The entire system is 100% open source and can be downloaded and run on your own infrastructure from http://wso2.org/.

There is a lot of future work to be done. For an example, some of the key areas are Quality of Service (QOS) and supporting Service Level Agreements (SLA). A key enhancement we are working on is to support agreed-upon latency and throughput based on different packages by adding resources and prioritizing requests at load balancers. Other important aspects are exploring fault tolerance scenarios, data backup and recovery, and monitoring and managing the resulting infrastructure. Furthermore, we are working on Carbon Studio, which is an eclipse-based single Integrated Development Environment (IDE) that simplify application development for Stratos.

References

Figure 1: SOA Platform Architecture

Figure 2: Stratos Architecture