International Workshop
on
"Micro Services and its Application in Analytics and Cloud"

Technically Sponsored By

Infosys Campus Connect

02nd-03rd Feb. 2019

(Duration: 2 Days)

WORKSHOP REPORT

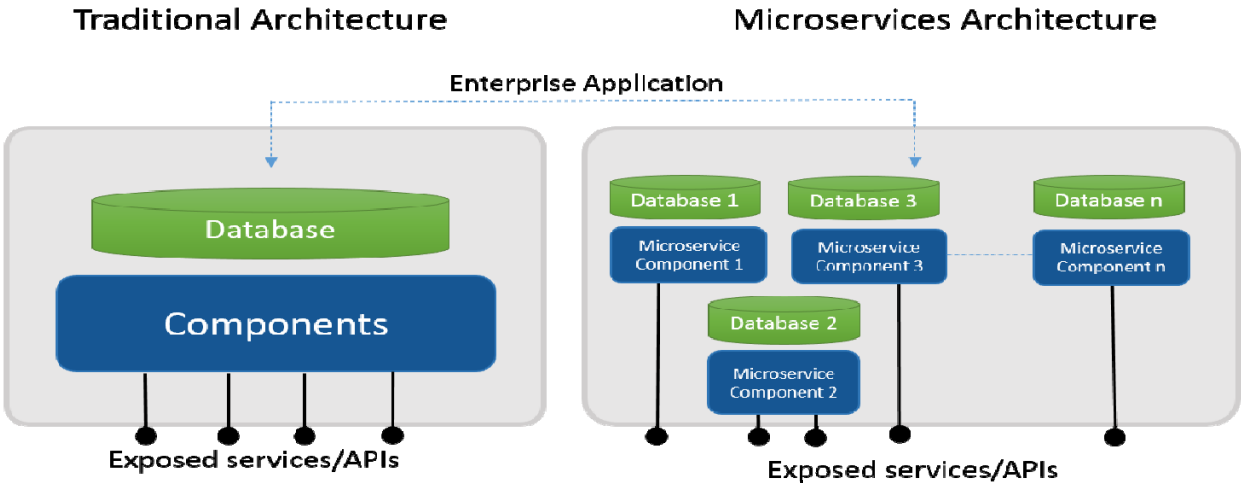
Perspective: About the Workshop

Objective:

The objective of this workshop is to provide training on Micro Services and its Application in Analytics and Cloud.

An Introduction to Microservices

Microservices are small, self-contained services that can evolve independently and deploy separately to support continuous integration and continuous delivery. A microservice architecture promotes developing and deploying an application or a set of features composed of independent, autonomous, modular, self-contained units. Each unit is assigned to a team that owns it for improving it. Adoption of microservices, teams can rapidly ship newer versions of applications or features without disrupting rest of the solution.



Why Microservices?

Enables Agile Processes: Moving away from a waterfall-based mindset that views software projects as large, multi-year capital expenditures. Microservices helps enterprises to fully adopt agile development and deployment methods with a strong technology platform that facilitates cloud-native approaches and microservices architecture setup.

Leverages Best-fit Technology for Each Component: Developers are choosing best-of-breed languages, frameworks, and tools to write parts of applications. One large application might be composed of microservices written in Node.js, Ruby on Rails, Python, R, and Java. Each microservice can be written in a language that is best suited for the task. Teams that develop microservices can make technology decisions that are right for the job. They can experiment with modern technologies, libraries, languages, and frameworks, yielding faster innovation cycles.

Modularity and Code Re-usability: Organizations today invest in reusable building blocks that are composable. Each microservice acts like a Lego block that can be plugged into an application stack. By investing in a set of core microservices, organizations can assemble them to build applications catering to a variety of use cases.

Elastic Infrastructure: With multiple cloud infrastructure providers is available in the market, enterprises today can dynamically provision, configure and orchestrate a few hundred virtual servers. But, instead of launching multiple instances of the application server, it is possible to scale-out a specific microservice on-demand. Microservices simplifies load balancing because when the load shifts to other parts of the application, an earlier microservice will be scaled-in while scaling-out a different microservice.

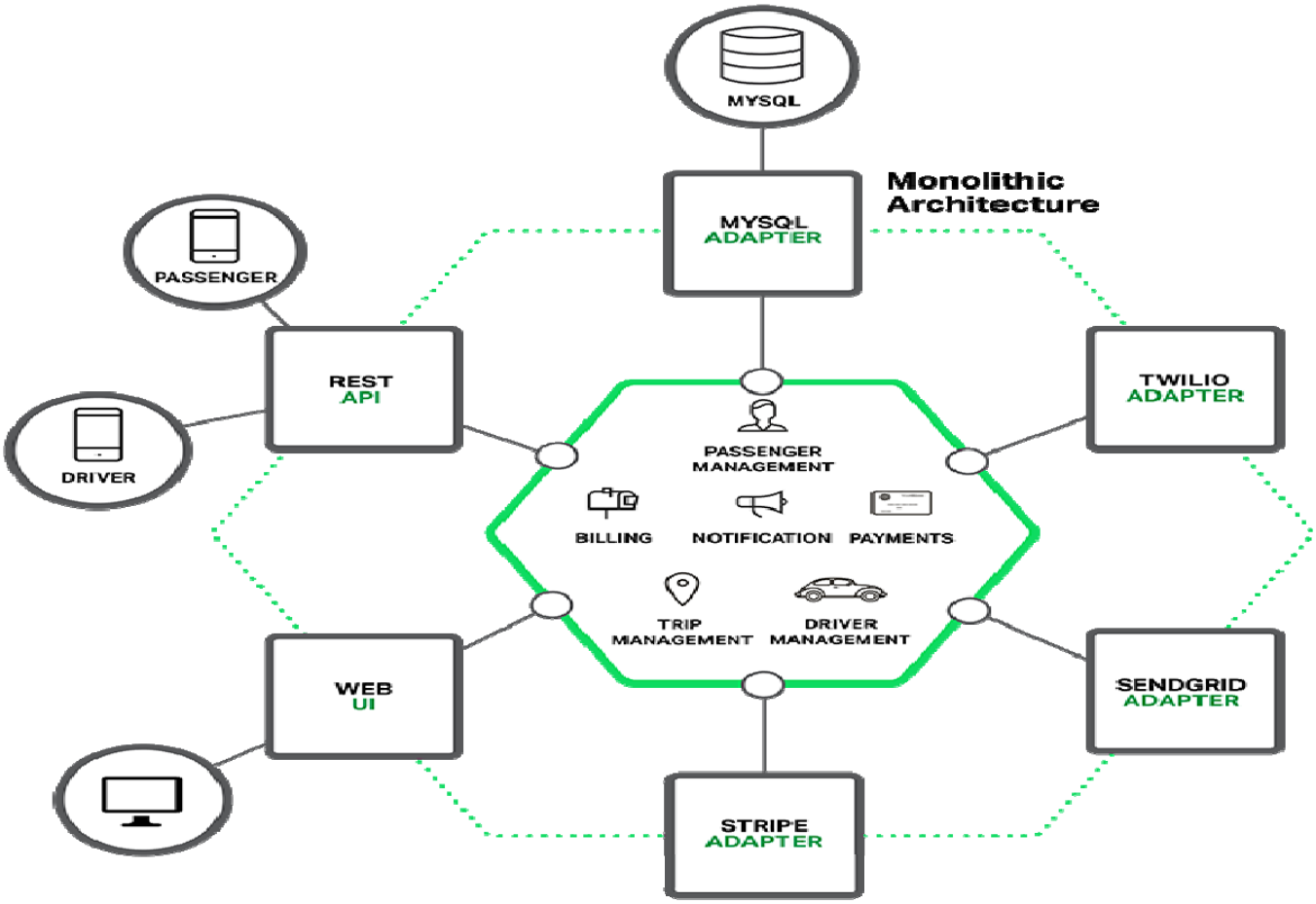
Ease of CICD (Continuous Integration and Continuous Delivery): Microservices helps small, autonomous, co-located teams with test driven development, continuous integration and continuous delivery that make the successful launch of each feature or application.

Decentralized Data and Governance: While microservices encourages its developers to save time by always using re-usable code libraries established by others, while also giving them the freedom to flirt with alternative solutions when needed. In addition, a solution with traditional architectures uses a single logical database across different applications. In a microservice, each service built for specific application or feature usually manages its unique database.

Building Monolithic Applications

A monolithic application is built as a single unit. Enterprise Applications are built in three parts: a database (consisting of many tables usually in a relational database management system), a client-side user interface (consisting of HTML pages and/or JavaScript running in a browser), and a server-side application. This server-side application will handle HTTP requests, execute some domain-specific logic, retrieve and update data from the database, and populate the HTML views to be sent to the browser. It is a monolith – a single logical executable. To make any alterations to the system, a developer must build and deploy an updated version of the server-side application.

This new application would have a modular hexagonal architecture, like in the following diagram:



At the core of the application is the business logic, which is implemented by modules that define services, domain objects, and events. Surrounding the core are adapters that interface with the external world. Examples of adapters include database access components, messaging components that produce and consume messages, and web components that either expose APIs or implement a UI.

Despite having a logically modular architecture, the application is packaged and deployed as a monolith. The actual format depends on the application's language and framework. For example, many Java applications are packaged as WAR files and deployed on application servers such as Tomcat or Jetty. Other Java applications are packaged as self-contained executable JARs. Similarly, Rails and Node.js applications are packaged as a directory hierarchy.

Applications written in this style are extremely common. They are simple to develop since our IDEs and other tools are focused on building a single application. These kinds of applications are also simple to test. You can implement end-to-end testing by simply launching the application and testing the UI with Selenium. Monolithic applications are also simple to deploy. You just have to copy the packaged application to a server. You can also scale the application by running multiple copies behind a load balancer. In the early stages of the project it works well.

The Benefits of Microservices

The Microservice architecture pattern has a number of important benefits. First, it tackles the problem of complexity. It decomposes what would otherwise be a monstrous monolithic application into a set of services. While the total amount of functionality is unchanged, the application has been broken up into manageable chunks or services. Each service has a well-defined boundary in the form of an RPC- or message-driven API. The Microservice architecture pattern enforces a level of modularity that in practice is extremely difficult to achieve with a monolithic code base. Consequently, individual services are much faster to develop, and much easier to understand and maintain.

Second, this architecture enables each service to be developed independently by a team that is focused on that service. The developers are free to choose whatever technologies make sense, provided that the service honors the API contract. Of course, most organizations would want to avoid complete anarchy and limit technology options. However, this freedom means that developers are no longer obligated to use the possibly obsolete technologies that existed at the start of a new project. When writing a new service, they have the option of using current technology. Moreover, since services are relatively small it becomes feasible to rewrite an old service using current technology.

Third, the Microservice architecture pattern enables each microservice to be deployed independently. Developers never need to coordinate the deployment of changes that are local to their service. These kinds of changes can be deployed as soon as they have been tested. The UI team can, for example, perform A|B testing and rapidly iterate on UI changes. The Microservice architecture pattern makes continuous deployment possible.

Finally, the Microservice architecture pattern enables each service to be scaled independently. You can deploy just the number of instances of each service that satisfy its capacity and availability constraints. Moreover, you can use the hardware that best matches a service's resource requirements. For example, you can deploy a CPU-intensive image processing service on EC2 Compute Optimized instances and deploy an in-memory database service on EC2 Memory-optimized instances.

Microservices Business Benefits

Reduces Time to Market: As each microservice is built and aligned around a business function to reduce the complexity of the application change-management process. It expedites updating new technologies, libraries, languages, and frameworks, yielding faster development cycles, including rollback. Each service is individually changed, tested, and deployed without affecting other services, enabling faster time to market.

Superior Application Quality: By using "divide-and-conquer" approach of microservices architecture, teams can perform both functional and performance testing of each unit easier than before as components can be tested in isolation and combined with a full or virtualized set of microservices. It also minimizes test automation and quality-assurance overhead and facilitate concurrent, A/B release testing on subsystems. Thus, microservices approach results in overall improvement in application quality.

Zero-Downtime Deployment: Microservices enables versions and releases planning process and backward compatibility requirements with master and multiple slave containers. It allows you to deploy the latest version of your microservice without interrupting the operation of the other micro-services.

Enterprises must modernize its applications on regular basis to keep-up the pace of change revolving around user-experiences, competitive advantage and quality delivery with highest possible performance. This results in frequent application upgrades with new features and bug fixes.

In the wake of this continuous evolution of adapting changes, traditional architecture has been slowly disappearing giving rise to microservices architecture. Giants like Amazon, eBay, Netflix, Twitter and many more are already reaping benefits, having hosted on microservices architecture.

The Drawbacks of Microservices

As Fred Brooks wrote almost 30 years ago, there are no silver bullets. Like every other technology, the Microservice architecture has drawbacks. One drawback is the name itself. The term *microservice* places excessive emphasis on service size. In fact, there are some developers who advocate for building extremely fine-grained 10-100 LOC services. While small services are preferable, it's important to remember that they are a means to an end and not the primary goal. The goal of microservices is to sufficiently decompose the application in order to facilitate agile application development and deployment.

Another major drawback of microservices is the complexity that arises from the fact that a microservices application is a distributed system. Developers need to choose and implement an inter-process communication mechanism based on either messaging or RPC. Moreover, they must also write code to handle partial failure since the destination of a request might be slow or unavailable. While none of this is rocket science, it's much more complex than in a monolithic application where modules invoke one another via language-level method/procedure calls.

Another challenge with microservices is the partitioned database architecture. Business transactions that update multiple business entities are fairly common. These kinds of transactions are trivial to implement in a monolithic application because there is a single database. In a microservices-based application, however, you need to update multiple databases owned by different services. Using distributed transactions is usually not an option, and not only because of the CAP theorem. They simply are not supported by many of today's highly scalable NoSQL databases and messaging brokers. You end up having to use an eventual consistency based approach, which is more challenging for developers.

Testing a microservices application is also much more complex. For example, with a modern framework such as Spring Boot it is trivial to write a test class that starts up a monolithic web application and tests its REST API. In contrast, a similar test class for a service would need to launch that service and any services that it depends upon (or at least configure stubs for those services). Once again, this is not rocket science but it's important to not underestimate the complexity of doing this.

Another major challenge with the Microservice architecture pattern is implementing changes that span multiple services. For example, let's imagine that you are implementing a story that requires changes to services A, B, and C, where A depends upon B and B depends upon C. In a monolithic application you could simply change

the corresponding modules, integrate the changes, and deploy them in one go. In contrast, in a Microservice architecture pattern you need to carefully plan and coordinate the rollout of changes to each of the services. For example, you would need to update service C, followed by service B, and then finally service A. Fortunately, most changes typically impact only one service and multi-service changes that require coordination are relatively rare.

Deploying a microservices-based application is also much more complex. A monolithic application is simply deployed on a set of identical servers behind a traditional load balancer. Each application instance is configured with the locations (host and ports) of infrastructure services such as the database and a message broker. In contrast, a microservice application typically consists of a large number of services. For example, Hailo has 160 different services and Netflix has over 600 according to Adrian Cockcroft. Each service will have multiple runtime instances. That's many more moving parts that need to be configured, deployed, scaled, and monitored. In addition, you will also need to implement a service discovery mechanism (discussed in a later post) that enables a service to discover the locations (hosts and ports) of any other services it needs to communicate with. Traditional trouble ticket-based and manual approaches to operations cannot scale to this level of complexity. Consequently, successfully deploying a microservices application requires greater control of deployment methods by developers, and a high level of automation.

Deploying analytics microservices in the cloud

Microservices are an architectural approach to creating cloud applications, where each application is built as a set of services. Each service runs in its own processes and communicates through application programming interfaces (API).

Services are built around specific business logic, written in any language and they are independently scalable, upgradeable and deployable. When an application is broken up into its component services, changes only affect specific services. Likewise, each service can independently scale in response to demand without consuming unnecessary resources.

One approach to automation is to use an off-the-shelf PaaS such as Cloud Foundry. A PaaS provides developers with an easy way to deploy and manage their microservices. It insulates them from concerns such as procuring and configuring IT resources. At the same time, the systems and network professionals who configure the PaaS can ensure compliance with best practices and with company policies.

Here's some history on how we've moved to using the cloud as a platform: International Data Corporation (IDC) has outlined a generational shift in IT patterns called the First, Second and Third platforms.

- The First Platform has custom hardware and software; it uses a centralized software architecture and scales vertically to serve thousands of apps and millions of users.
- The Second Platform is based on enterprise hardware and software. Client-server software architecture and vertical scaling are used to serve tens of thousands of apps and hundreds of millions of users.
- The Third Platform employs distributed software over horizontally scaling commodity hardware, the distributed nature of which enables linear scaling to serve millions of apps and billions of users.

The First Platform is made up of mainframe, mini computers with terminals as the user interface. The Second Platform saw the introduction of PCs, LANs, client-server architecture, and the Internet, with PCs as the main user interface devices. The Third Platform, cloud native, is the domain of mobile, cloud, big data and social. The majority of user interactions there take place through mobile devices.

Though there are many differences between the Second and Third platforms, the fundamental shift from enterprise to low-cost commodity hardware ushers in a level of cost advantage that makes large-scale cloud infrastructure possible. However, the shift from reliable enterprise hardware to a commodity alternative pushes the resiliency requirements from the underlying hardware up to the enterprise application.

Historically, enterprise applications were built as monolithic units in three main tiers: a client-side user interface, a database, and a server-side application.

The server-side application would handle HTTP requests, execute business logic, store, retrieve and update data from the database, and populate the client-side interface. The application was a single, logical executable. Any changes to the system involved building and deploying a new version of the server-side application. As more applications were deployed to the cloud, changes to any part of the application would cause the whole application to be rebuilt and redeployed. More resources were required to scale the entire application than the necessary portion of the application that needed scaling.

Cloud-native applications must be designed so that they can tolerate service failures. When a service fails, the application must respond as gracefully as possible. Thus, microservices require a sophisticated monitoring and logging setup for each individual service. It’s important to quickly detect failures so you can restore the service or substitute another, similar service.

A microservices-based architecture promotes an “API First” approach, decoupling APIs from their implementations for more agile development. It also works well for the continuous delivery software development process.

Service-oriented architecture (SOA) may sometimes feel like an outmoded IT industry paradigm from the first decade of the 2000s, but it could not be more relevant to the cloud computing arena of the mid-2010s and beyond.

Nevertheless, the interoperability pressures that spawned the SOA mania haven’t gone away. From the start, the point of SOA was to help users reduce their lock in to specific vendor platforms that offered core application and middleware functions on inflexible stacks. It’s all about maximizing users’ ability to share and reuse distributed services over a loosely coupled, multiprotocol, interoperability fabric made up of dumb pipes. Not just that, but SOA is also a paradigm that enables all these services on whatever platform to be accessed through fine-grained standard interfaces and to evolve independently while maintaining stable, service invocation interfaces.

The new SOA

None of of those interoperability points is less relevant to IT professionals today than it was 10 years ago. SOA has evolved into *microservices architecture*, a term with increasing frequency in recent months in industry circles. The flexible deployment capabilities implied in that microservices definition go to the heart of SOA, which has long been practiced in the most complex cloud services environments. Enabling distributed microservices to function as a distributed application may require a complex middleware fabric of reliable messaging, transactional rollback and long-running orchestration capabilities. These shared services need to span diverse microservices interaction patterns, such as hierarchical, parent-child, peer-to-peer or various blends thereof.

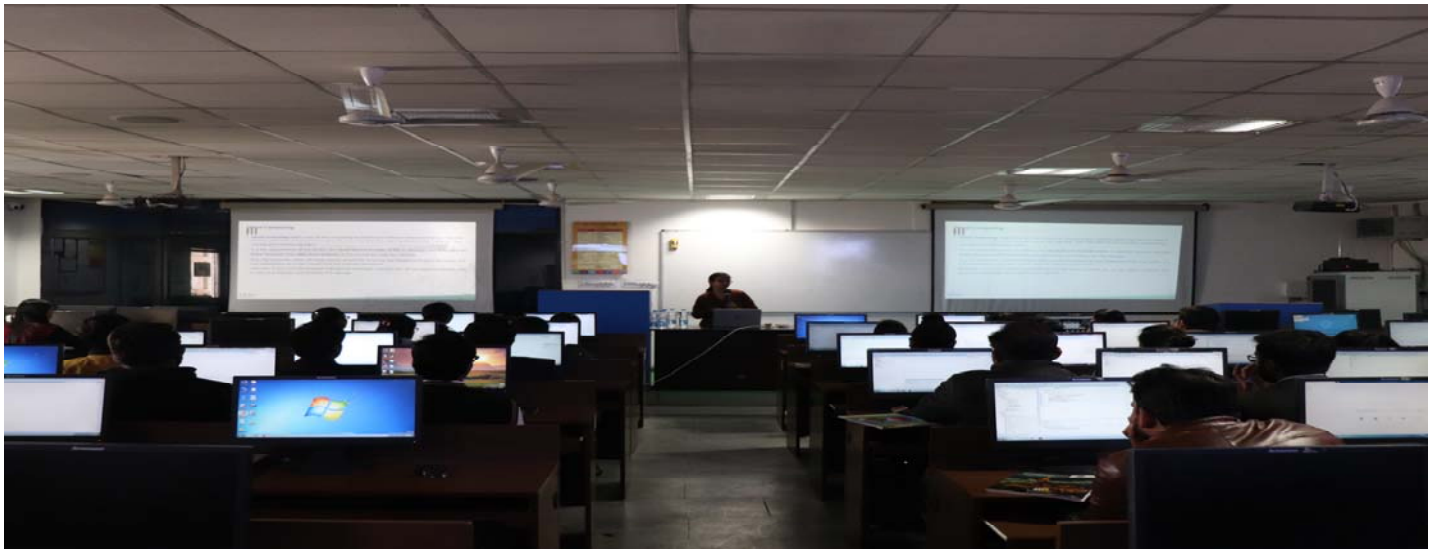
In the intervening years, big data analytics and the Internet of Things moved to the forefront of everybody's IT strategy. To the extent that any of those technologies was on the SOA community's collective radar in the mid-2000s, it was in big data analytics predecessors such as very large databases (VLDBs) and such precursors of the Internet of Things as machine-to-machine computing.

The microservices ahead

If you look at the distributed computing cloudscape of 2016 and beyond, clearly edge-oriented, big data analytics microservices are fast on their way to ubiquity. These microservices are taking the form of analytics—especially machine learning, deep learning and natural-language processing—that leverage the rich pallet of sensors on every Internet of Things endpoint. These embedded analytics microservices will be ubiquitous in every smart device, material or other artifact that comes to market. And data scientists will be building the embedded analytics microservices that enable all these Internet of Things–enabled artifacts to handle most of this processing locally and more rapidly and flexibly than any cloud service.

Actually, the *micro* part is a bit superfluous in this new context of the Internet of Things and big data analytics. The edge-oriented analytics services may not be any more micro. In other words, granular in functional scope than the properly factored services called for in established SOA practices. Nevertheless, the endpoints themselves where these services execute smart sensors, smartphones and so on will indeed be more micro or rather, *nano* in their resource constraints than the server clusters assumed in classic SOA.

Another way of looking at microservices architecture is simply that it's SOA in the new era of Internet of Things–centric, big data analytics–infused fog computing.



Technology: Microservices Using Java Eclipse Framework

Trainers: 1. Mr. Kulvaibhav Kaushik, Technology Lead, Infosys Limited

2. Ms. Ranu Sharma, Technological Analyst, Infosys Limited

Department: CS/IT, 6th & 8th Semester B.Tech. Students

Number of Students: 81

Number of Faculties: 05

Tools Used: Eclipse, Tomcat Server, Spring Framework

Venue: Industry Academia Interface Lab (IAI Lab)

Faculty Participants from SKIT: Dr. Shubhra Saxena, Mr. M.K.Beniwal, Mr. Pankaj Dadheech, Ms. Anjana Sangwan, Ms. Neha Mathur.

- **How the workshop will progress to the conference (ICETCE-2019)**

  The workshop is beneficial and have progress to the conference as workshop includes the "Micro Services and its Application in Analytics and Cloud" which directly maps with our International Conference *on* ***"Emerging Technologies in Computer Engineering: ICETCE-2019" organized on 01st-02nd Feb. 2019*** Sponsored By Department of Science & Technology, Rajasthan, Springer, IETE and Technically Sponsored by IBM, INFOSYS Campus Connect & Natural Group of Theme "Microservices in Big Data Analytics".

Hands On Session:

WORKSHOP SCHEDULE

Microservices and application in Analytics and Cloud

(ICETCE-2019) International Workshop, SKIT Jaipur

2nd - 3rd February, 2019

From	To	Duration	Theme	Workshop Topic	Mode	Anchor
Day 1: 2nd February, 2019						
9:00 AM	10:00 AM	1:00	Inagural	Registration		SKIT Team
				Inaugral Address	Lecture	
				Objectives and overview of workshop		Infosys Team
10:00 AM	10:15 AM	0:15		Break		
10:15 AM	12:00 Noon	1:45	Microservices	Introduction to Microservices	Lecture	Infosys Team
				Market trends		
				Comparison with traditional services and SOA		
				Architecture and Design		
				Tools for Microserices		
12:00 Noon	1:00 PM	1:00		Lunch Break		
1:00 PM	3:00 PM	2:00	Application Security	Microservice applications		Infosys Team
				Introduction to Analytics	Lecture	Infosys Team
				Microservice application in analytics		
Day 2: 3rd February, 2019						
9:00 AM	9:10 AM	0:10	Application Security	Context Setting for the day	Lecture	Infosys Team
9:10 AM	10:45 AM	1:35		Introduction to cloud		
				Microservice application in cloud		
10:45 AM	11:00 AM	0:15		Break		
11:00 AM	12:00 Noon	1:00	Project	Installation and setup	Practical	Infosys Team
				Assignment and Project		
12:00 Noon	1:00 PM	1:00		Lunch Break		
1:00 PM	2:30 PM	1:30	Project	Assignment and Project	Practical	Infosys Team
2:30 PM	3:00 PM	0:30	Feedback and Closure	Closure, Feedback and Felicitation		SKIT/ Infosys Team

- **Summary**

Building complex applications is inherently difficult. A Monolithic architecture only makes sense for simple, lightweight applications. You will end up in a world of pain if you use it for complex applications. The Microservice architecture pattern is the better choice for complex, evolving applications despite the drawbacks and implementation challenges.

- **Workshop's Outcomes (R&D, Placement)**

The workshop had a lot of knowledge of recent trends in Java those are very important for the students of pre final year and final year. This workshop will definitely help the students in their recruitment as well as in academics.

- **Road Ahead**

The workshop will help students in future for their minor and major projects of the 7th & 8thsemester students.

We are expecting these knowledgeable workshops in future also.

Welcome

Inaugural Address

Thank You

INTERNATIONAL WORKSHOP ON

"Micro Services and its Application in Analytics and Cloud"

(02nd Feb. 2019 to 03rd Feb. 2019)

TECHNICALLY SPONSORED BY: INFOSYS LTD.

ATTENDENCE SHEET

S.No	Name of Student	02/02/2019		03/02/2019	
		Pre lunch	Post lunch	Pre lunch	Post lunch
1	Ankita	PRESENT	PRESENT	PRESENT	PRESENT
2	Anshika Gupta	PRESENT	PRESENT	PRESENT	PRESENT
3	Deepanshu Samdani	PRESENT	PRESENT	ABSENT	PRESENT
4	Karan Raj Singh	PRESENT	PRESENT	PRESENT	PRESENT
5	Puneet Saxena	PRESENT	PRESENT	PRESENT	PRESENT
6	Raj Agarwal	PRESENT	PRESENT	PRESENT	PRESENT
7	Ritika Somani	ABSENT	PRESENT	PRESENT	PRESENT
8	Shipra Agarwal	PRESENT	PRESENT	PRESENT	PRESENT
9	Shradha S	PRESENT	PRESENT	PRESENT	PRESENT
10	Siddharth Sharma	PRESENT	PRESENT	PRESENT	PRESENT
11	Simran Khandelwal	PRESENT	PRESENT	PRESENT	PRESENT
12	Vaibhav Parashar	PRESENT	PRESENT	PRESENT	PRESENT
13	Vasishtha Vedant Sharma	PRESENT	PRESENT	PRESENT	PRESENT
14	Riitwick Krishnatri	PRESENT	PRESENT	PRESENT	PRESENT
15	Rohit Sharma	PRESENT	PRESENT	PRESENT	PRESENT
16	Ajay Sain	PRESENT	PRESENT	PRESENT	PRESENT
17	Prateek Surana	PRESENT	PRESENT	ABSENT	PRESENT
18	Anmol Babbar	PRESENT	PRESENT	PRESENT	PRESENT
19	Aayushi Choudhary	PRESENT	PRESENT	PRESENT	PRESENT
20	Adesh Gautam	PRESENT	PRESENT	PRESENT	PRESENT
21	Aditya Umesh Acharya	PRESENT	PRESENT	PRESENT	PRESENT
22	Akanksha Agarwal	PRESENT	PRESENT	PRESENT	PRESENT
23	Ankitpkg Goyal	PRESENT	PRESENT	PRESENT	PRESENT
24	Ankitrg Goyal	PRESENT	PRESENT	PRESENT	PRESENT
25	Ankit Kumar	PRESENT	PRESENT	PRESENT	PRESENT
26	Anshul Kumar Garg	PRESENT	PRESENT	PRESENT	PRESENT
27	Apoorva Gupta	PRESENT	PRESENT	PRESENT	PRESENT
28	Archit Bohra	PRESENT	PRESENT	PRESENT	PRESENT
29	Archit Mathur	PRESENT	ABSENT	PRESENT	PRESENT
30	Ashutosh Gurjar	PRESENT	PRESENT	PRESENT	PRESENT
31	Ayushi Gupta	PRESENT	PRESENT	PRESENT	PRESENT
32	Bhavya Kantiwal	PRESENT	PRESENT	PRESENT	PRESENT
33	Bhumi Vijay Shah	PRESENT	PRESENT	PRESENT	PRESENT
34	Bhupendra Kumawat	PRESENT	PRESENT	PRESENT	PRESENT
35	Deepanshu Sain	PRESENT	PRESENT	PRESENT	PRESENT
36	Harshita Jain	PRESENT	PRESENT	PRESENT	PRESENT
37	Himanshu Garg	PRESENT	PRESENT	PRESENT	ABSENT
38	Himanshu Satija	PRESENT	PRESENT	PRESENT	PRESENT
39	Ishanya Ashutosh Keshodhan	PRESENT	PRESENT	PRESENT	PRESENT
40	Keshav Mundra	PRESENT	PRESENT	PRESENT	PRESENT
41	Khushi Agrawal	PRESENT	PRESENT	PRESENT	PRESENT
42	Krishna Bhardwaj	PRESENT	PRESENT	PRESENT	PRESENT
43	Shubham Gupta	PRESENT	PRESENT	PRESENT	PRESENT
44	Kuldeep Singh	PRESENT	PRESENT	PRESENT	PRESENT
45	Manish Kumar Bansal	PRESENT	PRESENT	PRESENT	PRESENT

INTERNATIONAL WORKSHOP ON

"Micro Services and its Application in Analytics and Cloud"

(02nd Feb. 2019 to 03rd Feb. 2019)

TECHNICALLY SPONSORED BY: INFOSYS LTD.

ATTENDENCE SHEET

S.No	Name of Student	02/02/2019		03/02/2019	
		Pre lunch	Post lunch	Pre lunch	Post lunch
46	Manoj Jeswani	PRESENT	PRESENT	PRESENT	PRESENT
47	Manoj Sharma	ABSENT	PRESENT	PRESENT	PRESENT
48	Prakhar Khandelwal	PRESENT	PRESENT	PRESENT	PRESENT
49	Pramod Kumar	PRESENT	PRESENT	PRESENT	PRESENT
50	Pranjul Gupta	PRESENT	PRESENT	PRESENT	PRESENT
51	Punit Agarwal	PRESENT	PRESENT	PRESENT	PRESENT
52	Punit Kumar	PRESENT	PRESENT	PRESENT	PRESENT
53	Rahul Saini	PRESENT	PRESENT	PRESENT	PRESENT
54	Raj Kumar Kachhawa	PRESENT	PRESENT	PRESENT	PRESENT
55	Rishav Chetan	PRESENT	PRESENT	PRESENT	PRESENT
56	Sagar Goyal	PRESENT	PRESENT	PRESENT	PRESENT
57	Saharsh Anand Sharma	PRESENT	ABSENT	PRESENT	PRESENT
58	Saurabh Khandelwal	PRESENT	PRESENT	PRESENT	PRESENT
59	Shreya Gupta	PRESENT	PRESENT	PRESENT	PRESENT
60	Simran Ajwani	PRESENT	PRESENT	PRESENT	PRESENT
61	Tushar Setia	PRESENT	PRESENT	PRESENT	PRESENT
62	Upendra Upadhyay	PRESENT	PRESENT	PRESENT	PRESENT
63	Vaishali Jain	PRESENT	PRESENT	PRESENT	PRESENT
64	Vikas Motiani	PRESENT	PRESENT	PRESENT	PRESENT
65	Vinod Sharma	PRESENT	PRESENT	PRESENT	PRESENT
66	Yashika Singh	PRESENT	PRESENT	PRESENT	PRESENT
67	Anchal Arora	PRESENT	PRESENT	PRESENT	PRESENT
68	Rishi Garg	PRESENT	PRESENT	PRESENT	PRESENT
69	Atul Goyal	PRESENT	PRESENT	PRESENT	PRESENT
70	Ayushi Goyanka	PRESENT	PRESENT	PRESENT	PRESENT
71	Devansh Sharma	PRESENT	PRESENT	PRESENT	PRESENT
72	Divya Jain	PRESENT	PRESENT	PRESENT	PRESENT
73	Manmath Vasistha	PRESENT	PRESENT	PRESENT	PRESENT
74	Muskan Dosi	PRESENT	PRESENT	PRESENT	PRESENT
75	Piyush Jain	PRESENT	PRESENT	PRESENT	PRESENT
76	Shivam Pandey	PRESENT	ABSENT	PRESENT	PRESENT
77	Udita Garg	PRESENT	PRESENT	PRESENT	PRESENT
78	Vartika Khandelwal	PRESENT	PRESENT	PRESENT	PRESENT
79	Inesh Kumar	PRESENT	PRESENT	PRESENT	ABSENT
80	Pawan Jangid	PRESENT	PRESENT	PRESENT	PRESENT
81	Abhijeet Yadav	PRESENT	PRESENT	PRESENT	PRESENT
82	Dr. Shubhra Saxena	PRESENT	PRESENT	PRESENT	PRESENT
83	Mr. M.K.Beniwal	PRESENT	PRESENT	PRESENT	PRESENT
84	Mr. Pankaj Dadheech	PRESENT	PRESENT	PRESENT	PRESENT
85	Ms. Anjana Sangwan	PRESENT	PRESENT	PRESENT	PRESENT
86	Ms. Neha Mathur	PRESENT	PRESENT	PRESENT	PRESENT