(https://www.agiratech.com/)

Enterprise Application (https://www.agiratech.com/category/enterprise-application), Technical (https://www.agiratech.com/category/technical), Web Development (https://www.agiratech.com/category/web-development)

Distributed Transactions Patterns In Microservice Architecture

Say Hello!

By Bharanidharan Arumugam — September 25, 2019 — 2373 Views

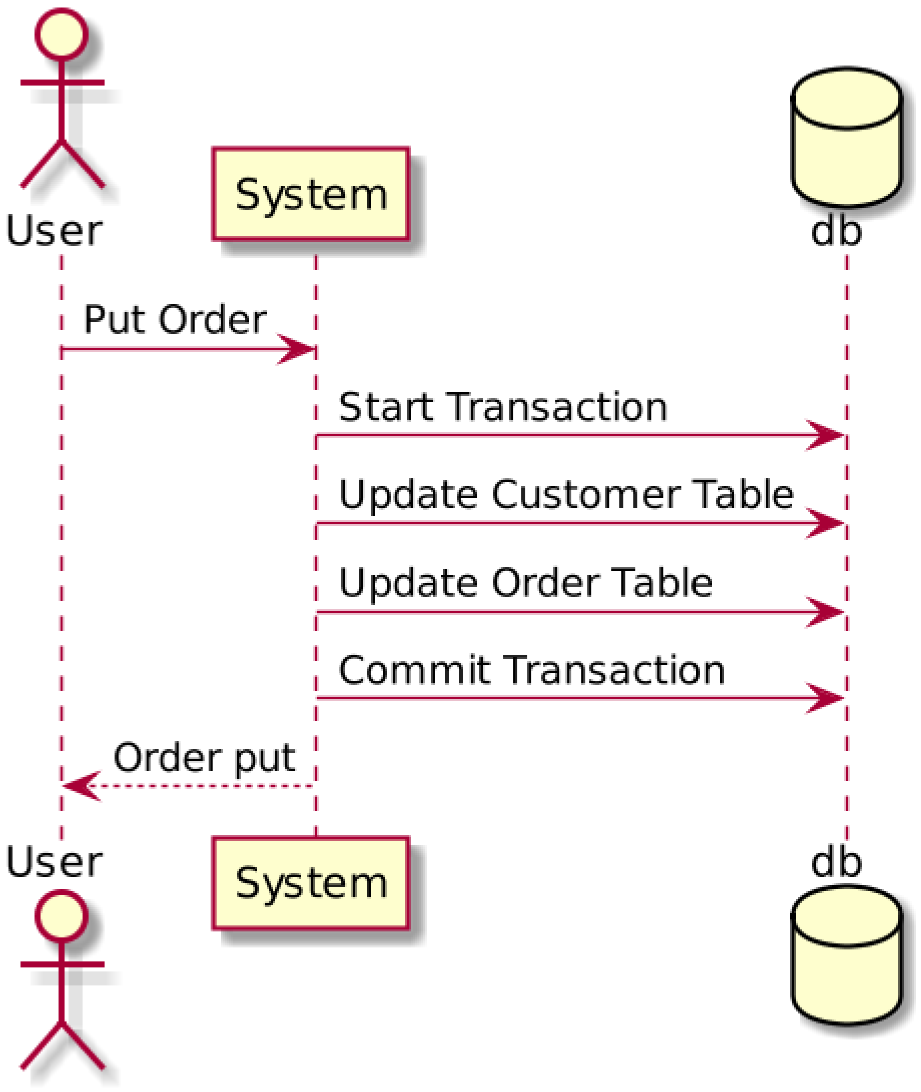
SCROLL

Microservices architecture is very popular in recent days to build big systems/apps. However, in this architecture, the most common problem is how to manage distributed transactions across multiple microservices. Here is a brief article on my real-time experience through our client's project, the actual problem and the possible solutions(patterns) that could solve it.

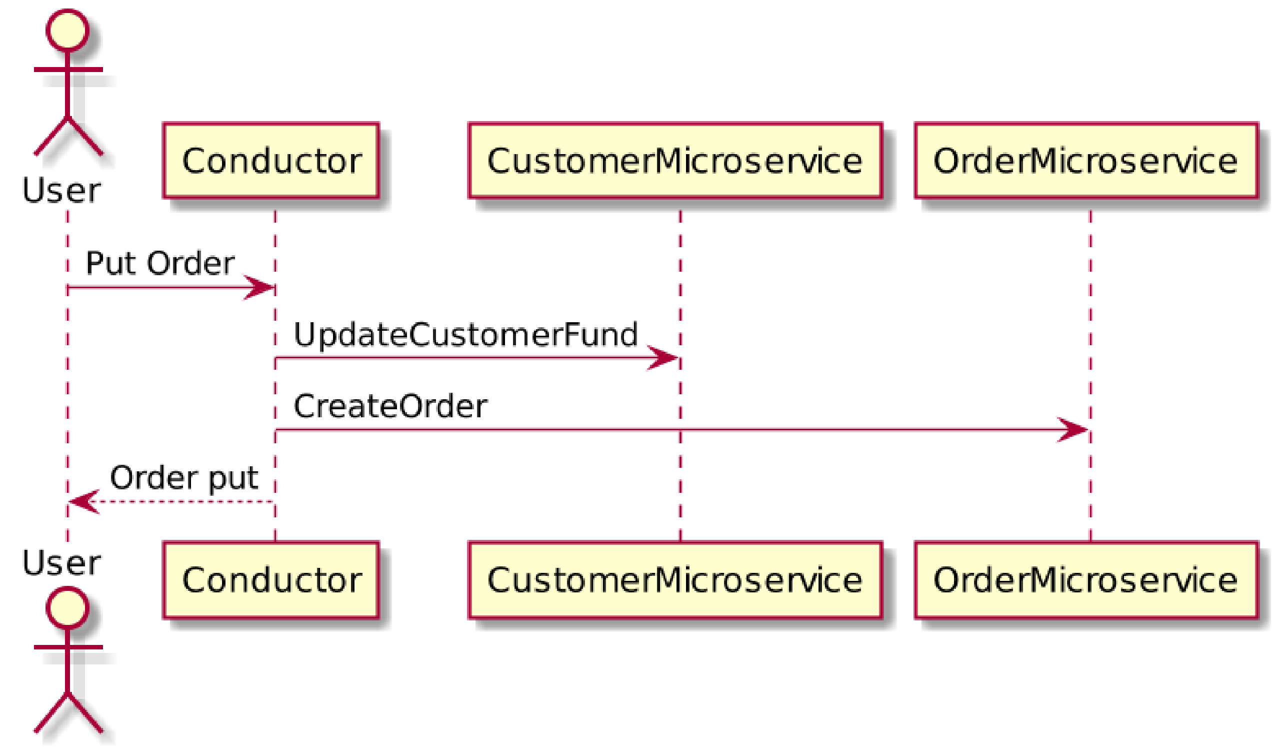
What is a distributed transaction?

When a monolithic application decomposes into microservice services, we have to break all transactions into distributed transaction. This means that all transactions in the monolithic system are now distributed into multiple services. Here is a customer order example with a monolithic system:

Privacy - Terms



(https://www.agiratech.com/wp-content/uploads/2019/09/1-monolithic.png) In the above example, if a user sends an Order action to a monolithic system, the system will create multiple database transactions that work over multiple database tables. If any step fails, the DB transaction can rollback. This is known as the ACID principle(Atomicity, Consistency, Isolation, Durability), which is guaranteed by the database system.
When decomposing this into microservice, we created 2 services i.e Customer Microservice and OrderMicroservice. Both of these have separate databases. Here is a customer order example with microservices:



Say Hello!

(https://www.agiratech.com/wp-content/uploads/2019/09/2-microservice-distributed-transaction-problem.png)

When an Order request comes from the user, both microservices will start to apply changes into their database. Because the transaction is now across multiple databases, it is now considered as a distributed transaction.

What are the Major Problems in Microservices?

In a monolithic system, we have a database system to ensure the ACID principle. But in microservice, we need to clarify the following key problems in ACIDity.

1. Keep transactions atomic – In a database system, atomicity means that database operations are such that either all queries get complete, or nothing occurs. The microservice-based system does not have a global transaction coordinator by default. In the above example, if the CreateOrder method fails, how do we roll back the changes that we applied by the Customer Microservice?
2. Keep transactions isolate – Before completing all the distributed transaction in microservice, if the user sends another request, should the object return old data or the updated one?. In the above example, Update Customer wallet

Privacy - Terms

succeeds. But it is still waiting for a response from CreateOrder service. Should requests for the current customer's fund return the updated amount or not?

Ways to Solve the Major Microservices Problems

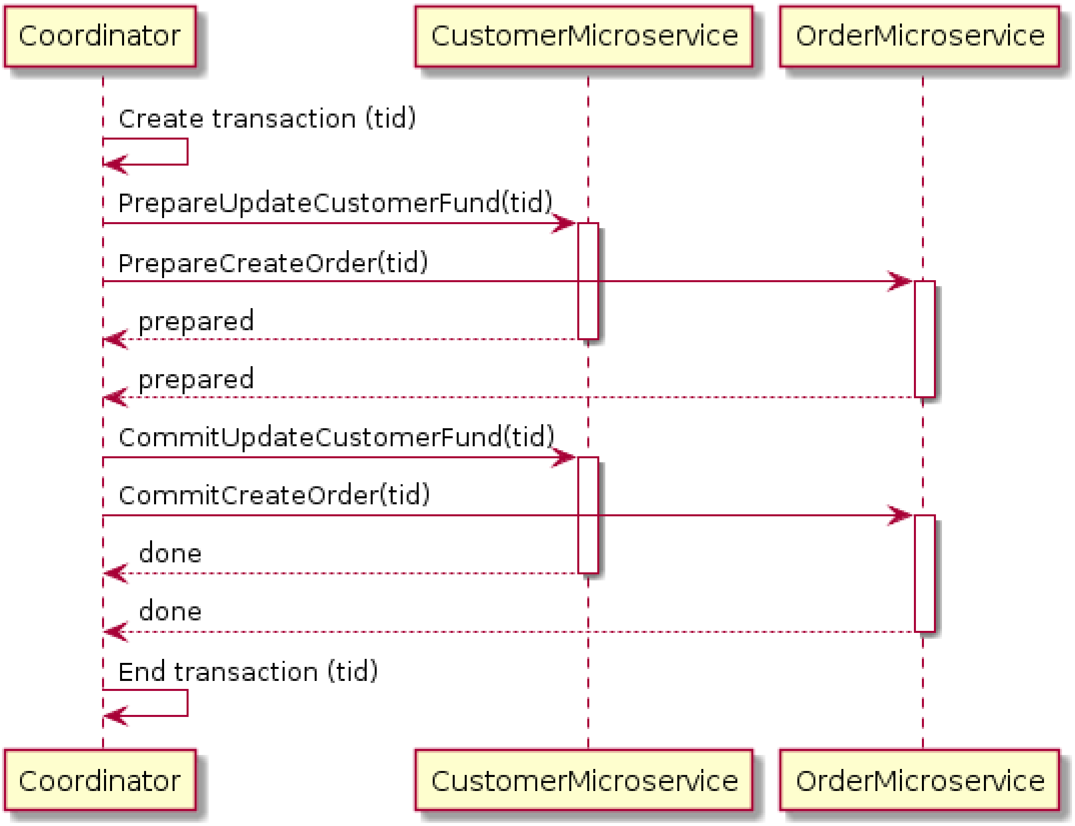
The following two patterns can resolve the above problem:

1. Two-phase commit pattern
2. Saga(3PC) pattern

Say Hello!

Two-phase commit(2PC) pattern:

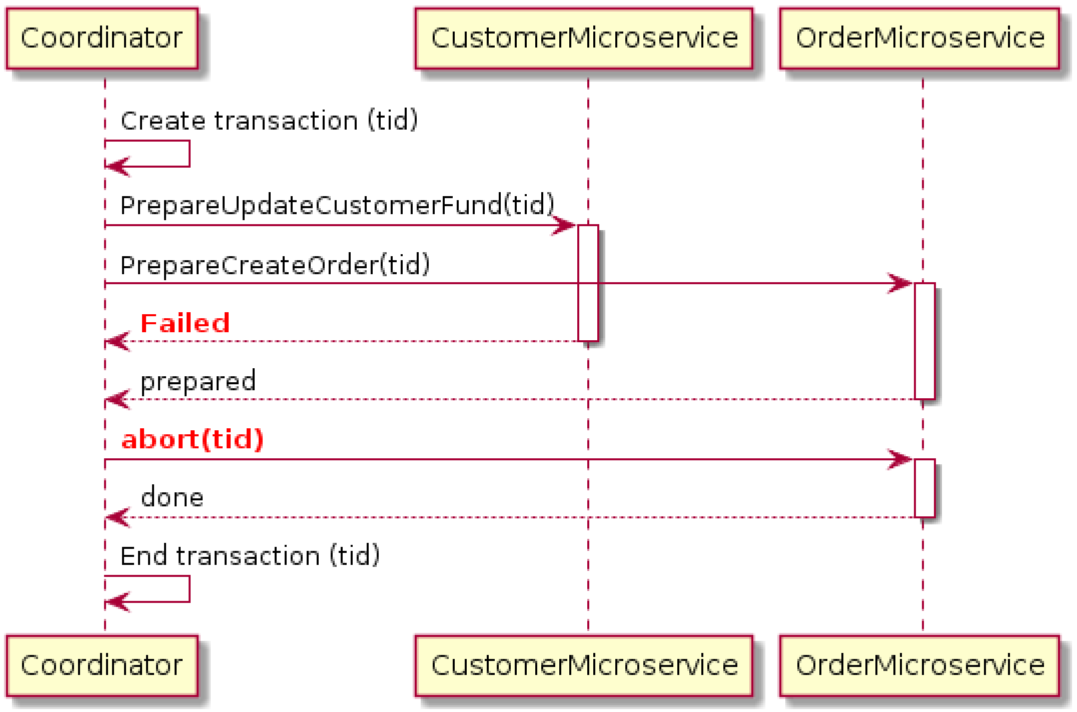
2PC has two phases: Prepare and Commit phase. In the prepare phase, all the microservices will ask you to prepare for some data change that could be done atomically. Once all microservices are prepared for data change, the commit phase will ask all the microservices to make the actual changes in DB. Normally, 2pc needs one global coordinator which takes care of Prepare and Commit phases across multiple services. Here is a 2PC implementation for the customer order example:



(https://www.agiratech.com/wp-content/uploads/2019/09/3-2pc-success.png)

In the above example, when a user sends an order request, the Coordinator will first create a global transaction with all information. Then, it will ask the Customer microservice to prepare for updating a customer fund with the created transaction i.e the Customer microservice will then check if the customer has enough funds to proceed with the transaction. Once the Customer microservice is ready to perform the change, it will lock down the DB records from further changes and tell the Coordinator that it is prepared. The same thing happens for the order in the Order microservice. Once the Coordinator has confirmed all microservices are ready to apply their DB changes, then it will ask them to apply their changes by requesting a commit with the transaction. Once both phases are completed, records will be unlocked.

In the above process, if we get any failure at any point of time, the Coordinator will abort the transactions and start the rollback process. Here is a diagram of a 2PC rollback for the customer order example:



(https://www.agiratech.com/wp-content/uploads/2019/09/4-2pc-failure.png) In the above sequence diagram, the Customer microservice fails to prepare for some reason, but the Order microservice prepares to create the order. The Coordinator will ask to abort all the prepared transactions i.e roll back changes and unlock the records.

Advantages of 2PC:

Using 2PC, we achieved all ACID principles.

1. More atomic(A)
2. Data consistency(C)
3. Read-write Isolation(I)
4. Durability(D)

Disadvantages of 2PC:

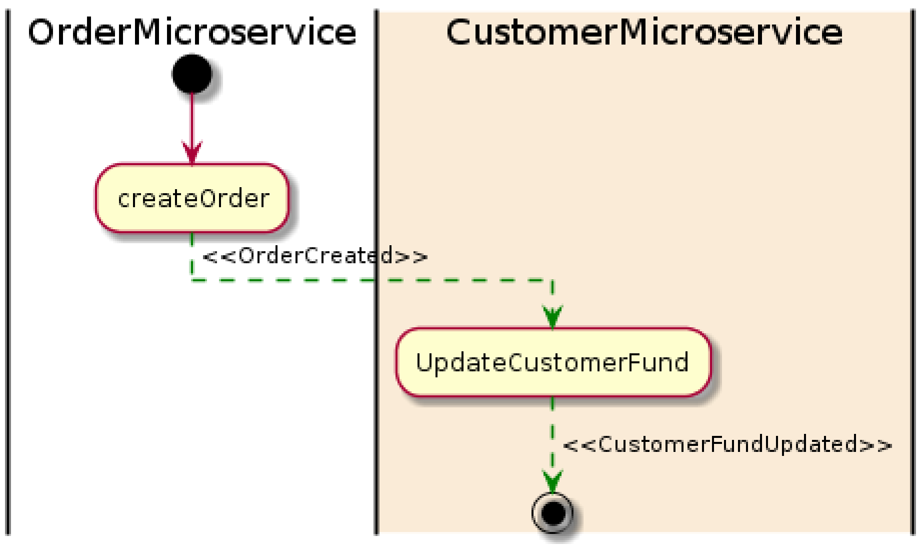
1. Request is synchronous (blocking)
2. Lock records/object until transaction completes
3. Possibility of Deadlock between transactions

The above drawback may lead to system performance bottleneck too.

SAGA(3PC) pattern:

The Saga pattern is another widely used pattern for distributed transactions. The Saga pattern is asynchronous and more reactive. In this pattern, all the distributed transaction is completely asynchronous i.e we have to use AMQP such as RabbitMq, Kafka, etc. So the microservices communicate with each other through an event. If you want to know more about the event bus, please refer to our blog on **rabbitMQ. (https://www.agiratech.com/blog/message-queues-golang-rabbitmq/)**

Here is a diagram of the Saga pattern for the customer order example:



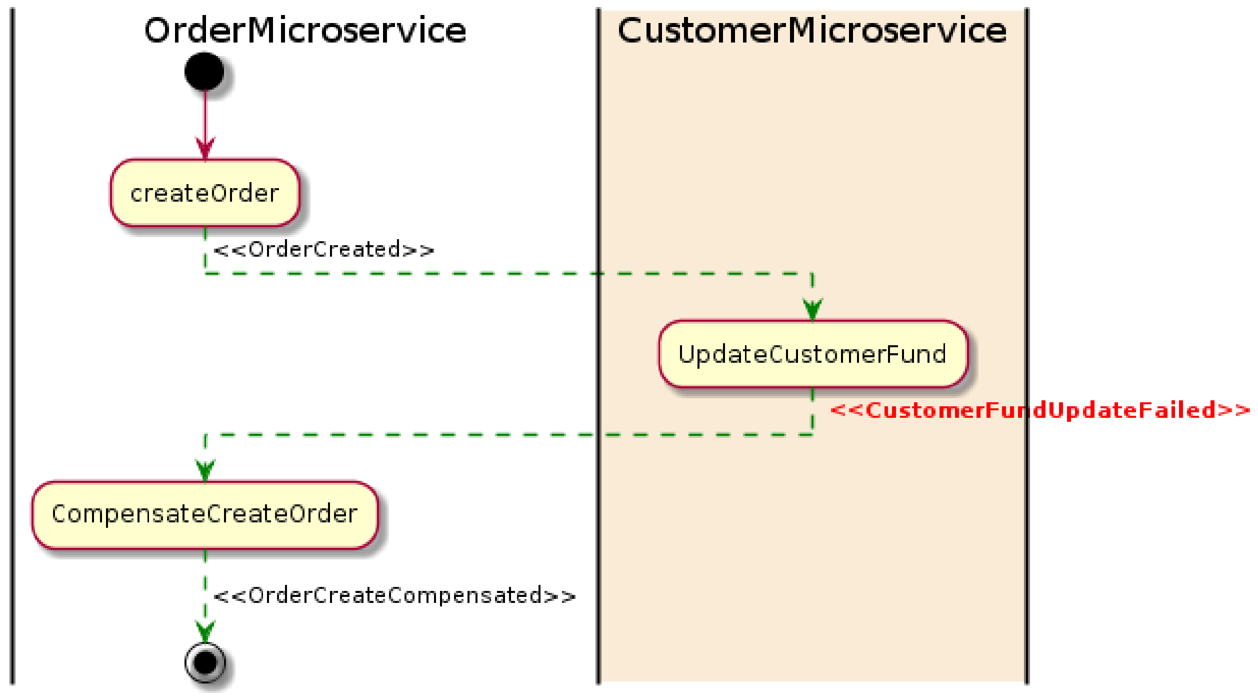
(https://www.agiratech.com/wp-content/uploads/2019/09/5-saga-success.png)

In the example above, the Order Microservice receives a request to place an order. Firstly, it starts to create an order and then emits an OrderCreated event. The CustomerMicroservice listens for this (OrderCreated)event and updates a customer fund once the event is received. If the amount is successfully deducted, then it will emit a CustomerFundUpdated event, which is the end of the transaction.

If any microservice fails to complete its transaction, the other microservices will begin to run compensation transactions event i.e to roll back the changes. So we call this pattern as 3 phase commit. Here is a diagram of the Saga pattern for a compensation transaction:

Say Hello!

Privacy - Terms



Say Hello!

(https://www.agiratech.com/wp-content/uploads/2019/09/6-saga-failure.png) In the example above, the Update Customer Fund is failed for some reason and it then emitted compensated event i.e CustomerFundUpdateFailed event. The Order Microservice listens this the event and starts to revert the order which was created earlier.

Advantages of the Saga pattern

1. Data reliability and Long-lived transactions
2. Request are asynchronous
3. No lock for DB object

Disadvantages of the Saga pattern

1. Difficult to debug/test especially if we have multiple microservices
2. it does not have read isolation i.e user can see the order created but in next second, the order is removed due to failure transaction

Conclusion

The Saga pattern is the most preferable way of solving distributed transaction problems for a microservice-based architecture. As a team if we have to adopt the Saga pattern, it requires a change in mindset for both development and testing. Sometimes It could be a challenge for a team that is not familiar with this pattern. Therefore, it is very important to choose the proper way to implement it for a project.

Read more interesting blogs related to Microservices such as **Monolithic vs Microlithic architecture (https://www.agiratech.com/blog/monolithic-architecture-vs-microservices-architecture-a-glance/)**, **Message queues in golang RabbitMQ (https://www.agiratech.com/blog/message-queues-golang-rabbitmq/)**, **Building service oriented architecture using rails and kafka (https://www.agiratech.com/blog/building-service-oriented-architecture-using-rails-and-kafka/)**

Say Hello!

TURN YOUR VISION TO MAGNIFICENT REALITY

WITH

Our Web and Mobile Solutions

REQUEST QUOTE (HTTPS://WWW.AGIRATECH.COM/CONTACT-US/)

Advantage Of Microservice Architecture (Https://Www.Agiratech.Com/Tag/Advantage-Of-Microservice-Architecture)

And Culture (Https://Www.Agiratech.Com/Tag/And-Culture)

Challenges In Microservice Architecture (Https://Www.Agiratech.Com/Tag/Challenges-In-Microservice-Architecture)

Disadvantages Of Microservice Architecture (Https://Www.Agiratech.Com/Tag/Disadvantages-Of-Microservice-Architecture)

Microservice Architecture Adapts Following Concepts (Https://Www.Agiratech.Com/Tag/Microservice-Architecture-Adapts-Following-Concepts)

Microservice Architecture Java (Https://Www.Agiratech.Com/Tag/Microservice-Architecture-Java)

Microservice Architecture: Aligning Principles (Https://Www.Agiratech.Com/Tag/Microservice-Architecture-Aligning-Principles)

Microservices Architecture C# (Https://Www.Agiratech.Com/Tag/Microservices-Architecture-C)

Microservices Spring Boot (Https://Www.Agiratech.Com/Tag/Microservices-Spring-Boot)

Microservices Vs Api (Https://Www.Agiratech.Com/Tag/Microservices-Vs-Api)

Microservices Wiki (Https://Www.Agiratech.Com/Tag/Microservices-Wiki)

Monolithic Architecture (Https://Www.Agiratech.Com/Tag/Monolithic-Architecture)

Monolithic Vs Microservices Architecture (Https://Www.Agiratech.Com/Tag/Monolithic-Vs-Microservices-Architecture)

Practices (Https://Www.Agiratech.Com/Tag/Practices)

What Is Microservice Architecture? (Https://Www.Agiratech.Com/Tag/What-Is-Microservice-Architecture)

Say Hello!

0 Likes

Bharanidharan Arumugam

Technical Architect | Tech Lead | Mentor | - An enthusiastic & vibrant Full stack developer has 7 plus years of experience in Web development arena. Owns legitimate knowledge in Ruby, Ruby On Rails, AngularJs, NodeJs, MEAN Stack, CMS Services. Apart all, A Modest human who strictly says "No To Harming Humans".

Privacy - Terms

PREVIOUS POST

(https://www.agiratech.com/ways-to-stand-out-as-a-leader)

(https://www.agiratech.com/

NEXT POST

(https://www.agiratech.com/8-startup-tips-for-succeeding-business)

Related Blogs

Modern Best Practices in the
DRUPAL CMS DEVELOPMENT & INTEGRATION
Agira
CMS
www.agiratech.com

Web Development (https://www.agiratech.com/category/web-development)

(https://www.agiratech.com/modern-best-practices-in-the-cms-development-and-integration)

Say Hello!

Web Development (https://www.agiratech.com/category/web-development)

(https://www.agiratech.com/how-do-large-

Privacy - Terms

companies-hire-the-best-web-developers)

Web Development (https://www.agiratech.com/category/web-development)

(https://www.agiratech.com/java-vs-net-who-will-reign-in-the-future)

Enterprise Application (https://www.agiratech.com/category/enterprise-application)

(https://www.agiratech.com/tips-to-increase-online-purchases-on-your-retail-sites)

10 Tips To Increase Online Purchases On Your Retail Sites (Https://Www.Agiratech.Com/Tips-To-Increase-Online-Purchases-On-Your-Retail-Sites)

Say Hello!

Privacy - Terms

Enterprise Application (https://www.agiratech.com/category/enterprise-application)

(https://www.agiratech.com/ecommerce-vs-mcommerce-what-are-the-differences)

Enterprise Application (https://www.agiratech.com/category/enterprise-application)

(https://www.agiratech.com/ecommerce-platforms-for-seo)

Top 5 ECommerce Platforms For

Say Hello!

Agira Technologies

We empower your business through technology and enterprise-grade innovative solutions for businesses.

Privacy - Terms

(https://www.facebook.com/agiratechnologies/)

(https://twitter.com/agiratech)

(https://www.linkedin.com/company/agira-technologies-pvt-ltd)

(https://www.youtube.com/channel/UCwhIlf3kUYpD1JTiKTIDIRw)

(https://www.instagram.com/agiratechnologies/)

Say Hello!

Services

Web Development (https://www.agiratech.com/web-development)

Mobile Development (https://www.agiratech.com/mobile-development)

User Experience (https://www.agiratech.com/ui-ux-design)

Legacy Modernization (https://www.agiratech.com/legacy-modernization)

Rescue Project (https://www.agiratech.com/rescue-project)

DevOps Consulting (https://www.agiratech.com/devops-consulting)

Privacy - Terms

Integration Services (https://www.agiratech.com/integration-services)

IT Staffing (https://www.agiratech.com/it-staffing)

Blockchain (https://www.agiratech.com/blockchain)

Learning Management System (https://www.agiratech.com/learning-management-system)

Geographic Information System (https://www.agiratech.com/gis)

Say Hello!

Company

Contact Us (https://www.agiratech.com/contact-us)

Careers (https://www.agiratech.com/careers)

Case Studies (https://www.agiratech.com/case-studies)

Get Connect

Phone: +1 732 992 8488

Email: sales@agiratech.com

Agira Technologies Inc, 1604 US Highway 130, North Brunswick, New Jersey 08902.

Privacy - Terms

Agira Technologies Pvt Ltd, #36/ 2B, Ground Floor, Mount Poonamalle High Rd, St.Thomas Mount, Tamil Nadu 600016.

Iteron AG, Picassoplatz 4, CH-4052 Basel.

Say Hello!

Privacy - Terms