**Microservices Architecture**

Microservices Architecture has become a norm off late due to its simplicity in the Maintenance and faster time to deliver the new features minimizing the number of issues.

With Microservices based Architecture, Monoliths are converted into the smaller Applications each providing a particular feature of the overall functionality and they interact with each other through communication protocols such as HTTP and TCP.

Benefits associated with the Microservices based architecture outweigh the extra overhead that is associated with it.

Pros of Microservices

1. Single Responsibility Model
2. Independent Feature Growth with its own Release Cycle
3. Easy to Scale Individual Applications
4. Reduces Time to Market and Speeds up your CI/CD pipeline, Agility
5. Better Failure Tolerance
6. Easy to Maintain
7. Platform and Language Agnostic

Cons

1. Need more collaboration
2. Operational Overhead
3. Need to define well set Devops/Automation Process
4. Security issues
5. Data Replication/Synchronization along with Operational Overhead
6. Overall Operational Cost

Things to Consider while converting Monolithic to Microservices Based Architecture

1. Database Perspective
2. Application Perspective
3. Infrastructure Perspective
4. Network Perspective
5. Deployment/Automation Perspective
6. User Perspective
7. Security Perspective

Some of the considerations while converting Monolithic to Microservices are below

1. Vertical Slicing vs Horizontal Slicing
2. Deployment Mechanism
3. Communication Mechanism
4. Data Synchronization Needs/Data Migration to Data Warehouse
5. Nothing Shared Architecture Model

Things to consider in the Microservices based Architecture

* High Availability
* Scalability
* Latency
* SSL Termination
* Rate Limiting
* Stateless/Stateful
* Security
* DevOps/Deployment
* Service Discovery/Registry
* Distributed Tracing
* Monitoring
* Design Patterns

Real time Experience with Microservices Based Architecture and Roles and responsibilities performed in the previous projects

1. **High Availability** is obtained by deploying the applications in multiple Data Centers or Regions/AZ’s and having a Load balancer in front to distribute the load. Few options of Load Balancers include HA Proxy and F5 Controller or ALB from the AWS. I have heavily used HA Proxy and fine tuned and monitored the HA Proxy for the Load Balancing. GSLB is used for as the Geo based Routing to route requests to the nearest Region and HAProxy is used to route requests within the region.
2. **Scalability** is all about how does our system perform under heavy load/unusual load and is it capable of handling those loads without any interruption to the service. Autoscaling options are available and Capacity planning is performed to get the baseline of the number of requests that a node/server could handle and prorate it accordingly for the peak loads. One of the mechanisms for Scalability considerations were how to Auto scale. We considered CPU/Memory and Number of Requests Threads that are being processed to determine the Autoscaling needs.
3. **Latency** is all about how fast your request is being processed. Few things we considered for this approach was Caching using Redis/Highly Performant codes with squeezing as much as we can from the DB Query perspective by creating indexes/hints, Caching the look up values and Precompiling the Queries/Creating Materialized Views if necessary/CDN using Edge locations/Reducing Network hops etc.
4. **SSL Termination** – SSL is needed to encrypt your data at transit for security reasons and one of main considerations is where to perform the SSL termination. Normally SSL Termination happens at the Load Balancer level but I have seen applications using SSL termination being done close to the application for extra security. In my previous projects, we have been doing SSL termination at the Load Balancer level.
5. **Rate Limiting –** This feature ensures that our application is not overloaded by a single Client misuse or to ensure we enforce some limits on the resource usage. When it comes to rate limiting there are multiple limits to be considered. One is the **Global Limit** whereby we ensure that the application/Microservice does not receive more than the certain number of RPS to drain the underlying resources or have limitations based on the Hardware limitations. Rate Limiting is also done for each Client accessing the Microservices to ensure that a single Client doesn’t overuse the System Resources. In my previous projects, we have been using both the Global/Client Limits.
6. **Stateless/Stateful** nature of the Microservices. Most of the Microservices are built on top of Http Protocol using Rest Based API’s and hence are by default Stateless. We need to ensure that our Microservices are always Stateless and each request is independent of the other from the perspective of Request distribution.
7. **Security –** All the Microservices that are developed are secured using various means. One of the Standard mechanisms includeOAuth 2.0 WITH JWT Token. JWT Token includes information about Authentication and Authorization and OAUTH 2.0 provides a standard mechanism to authenticate and provide JWT tokens to be used clients to access the Microservices. We had a separate team dealing with the OAuth Security services which were exposed as a Services for the Clients and Microservices to consume for Authentication and Authorization purpose.
8. **DevOps/Deployment –** As part of theMicroservices based approach one of the main things is about how do we go about making the release a seamless process and how to automate to avoid the overall release process for the faster and independent delivery of the Services. Jenkins Pipelines with the Automatic Webhooks to detect the Code changes are built. Efficient CI/CD pipelines are constructed to ensure that release is smoother and faster. In my previous project our release to Dev and Stage are automatic with the Jenkins Pipe line which follows a pure CI/CD – Continuous Integration and Continuous Delivery. But for Production we had a separate Jenkins job which gets triggered manually to perform the deployment. We followed different deployment strategy from Blue/Green to Canary to Rolling updates based on the needs.
9. **Service Discovery and Service Registry** is one of necessary things to consider in the Microservices based architecture/applications when there are so many Microservices are involved running in numerous instances/nodes. We had a custom-built application for the Service Discovery/Service Registry though there are many commercial/ OSS solutions are available. Few of the options

include Netflix OSS like Eureka.

1. **Distributed Tracing** ensures/aids the team in faster trouble shooting and understanding the request propagation across multiple Microservices to track a given request. Concept of Correlation Id/TraceId are heavily used. Spring Cloud Sleuth is one of the Softwares which helps in Distributed Tracing by propagating the information of TraceId/SpanId and RequestId across multiple services. We had used Spring Cloud Sleuth for our internal applications with in the Business Unit and the Concept of Correlation id to be propagated across business units
2. **Monitoring –** Various tools were heavily used when it comes to monitoring theapplications. Log Analytics such as Splunk/ELK is used to created Real Time Dashboards as well as Historical Dashboards for monitoring/alerting purpose. Application Process Monitoring tools are also used heavily to monitor the applications from the CPU/Memory/Thread Utilization and other aspects of the application. AppDynamics have been heavily used. These tools are used to come up with the SLI/SLO/SLA’s and used to create/identify various metrics.
3. **Design Patterns –** When it comes todesign patterns there are various different patterns available from the micorservices perspective. But the idea behind this is to avoid reinventing the wheels. But following are the few things that needs to considered and when it comes to inter service communications and identify the existing patterns accordingly.
   1. **Distributed Transaction** – How does a Business need be follow the ACID property if it spans multiple Microservices
   2. **Connection Mechanism -** Failure of the Microservices and Retry logic. Connection Pooling and other connection oriented/related task. With Interservice communication, with various Timeouts for the Http Calls like Socket Timeout, Connection Timeout, Request Time out and any failures should we need to retry the Http Calls are something that we leverage the existing design patterns
   3. **SSL/Security**
   4. **Aggregating** the Responses
   5. **Service Mesh** to identify the Calls with a Dashboard to view the Complete Depencies. Istio as Sidecar pattern is being used.
   6. **Logging/Tracing Capabilities.**

My roles and responsibilities in the Microservices based Architecture include performing all of the above along with the considerations for Operation Excellence, Cost Optimization, Capacity Planning, Deployment considerations.