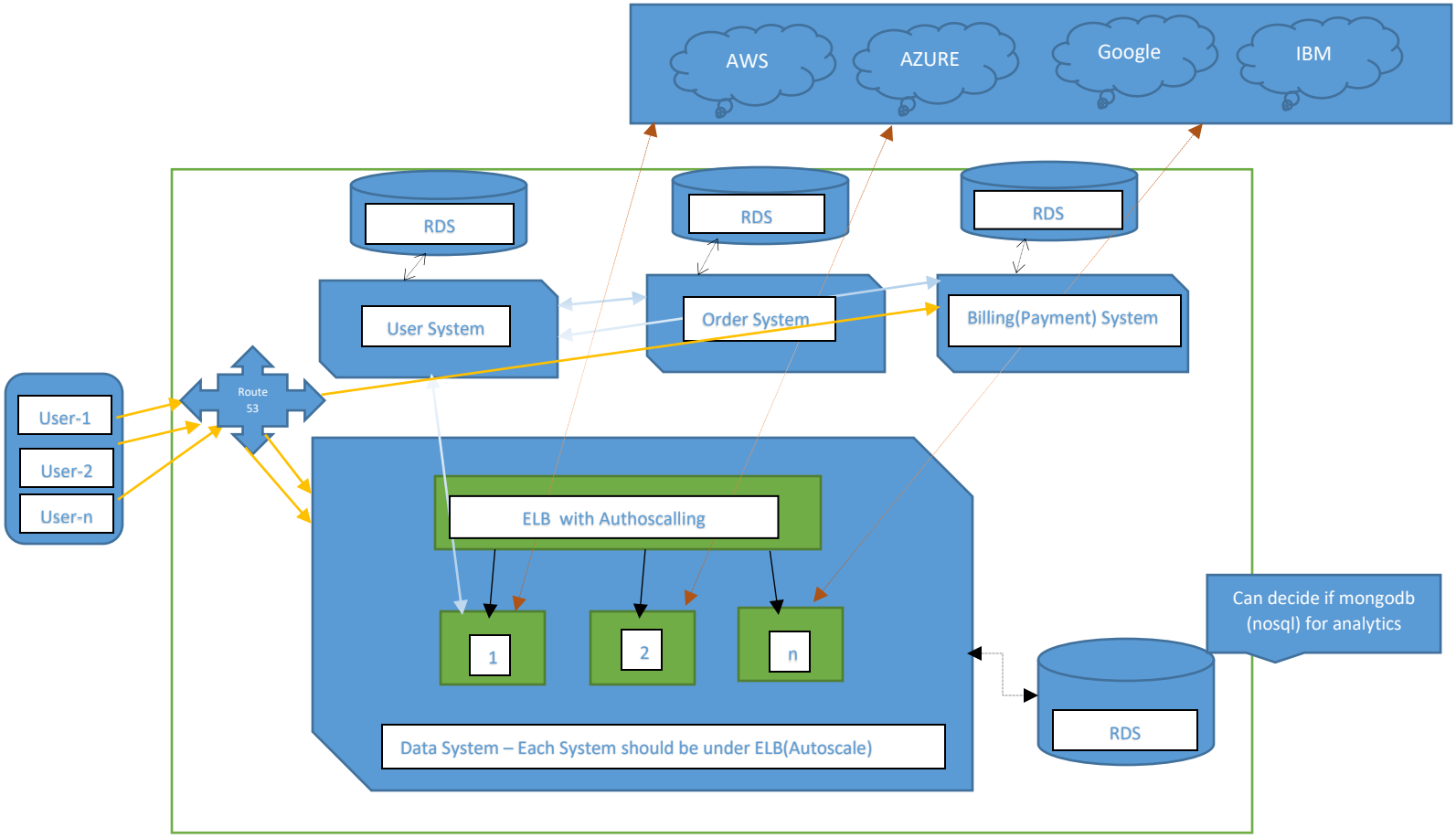
A] Get data immediately from MLass (cloud) service (based on business requirement)

1. Users request System to get the data from api (/user/data) api
2. Each user validated by USER SYSTEM – which contains all users data
    a. Users Is validated through JWT Token
    b. Each user is validated through user system only
    c. Once validated, user data provided to the requested system, with user registered CLOUD system.
        i. If not registered then given appropriate message as per that.
    d. If not validated then given message as per that (501 – Not authenticated)
3. On successful validation
    a. Based on user CLOUD system, get the data
        i. Assume we are using mLass system apis so now it will be very faster, so get data immediately.
    b. Get related data and give back to end user

B] Get data after some time based on Business Requirement

As discussed if User is given first time the data for processing Or if it is time taking process any how then we can go by below way (based on callback url with process id):

1. Follow above 1 and 2 steps
2. On successful validation
    a. Call /user/order service
    b. Get on process id (one table in database user_processes, created new entry and get newly generated id as PROCESS_ID
        i. This PROCESS_ID is unique for each process based on which user (system – mobile or angular) can get the data for that process based on PROCESS_ID
        ii. One api to get all process info based on PROCESS_ID
        iii. The user_processes table also contains some fields like percentage OR status field
        iv. This status is (0-under_process, 1-Success, 2-Failure)
            1. Which will be given back to user when he requests
            2. If success (1) then he can call the above /user/data) api
3. Also, here we can use SQS to process multiple requests without missing any :
    a. Put each request in SQS
    b. SQS will be integrated with SNS
    c. **SNS will call Node.js api (EC2 with ELB and Autoscalling) OR LAMBDA**
    d. Node.js will process those requests



Note : Here, assume each system is under ELB and integrated Autoscalling group, so can achieve performance without problems and also can manage cost (as autoscalling with min:1 to max: as per business requirement)

About Each Service (in detail) :: (SOA architecture): - also used for developer understanding:

1. User Service
    a. User for authentication and authorization
    b. Other user related setting also in this system
    c. Get called by each system
2. Order Service **(if required, can decide based on discussion in more details about business requirement)**
    a. Call user service for auth, payment service to process order payment
    b. For user's new order service, internally call cloud service to process data
    c. User can put order and if it takes time can create the process id and give it back to user
    d. Based on process id , create one api to get the status of the process
    e. Contain other different apis like order status/history/details…etc.
3. Billing(Payment) service: **(if required, can decide based on discussion in more details about business requirement)**
    a. Call user service for auth, order service to get order details
    b. Used to store user payment , based on the cloud service used
    c. Also can integrate here diff. payment gateways as per requirement
    d. Store user payment/update payment history
    e. Contain other different apis like process payment/history/order payment/user payment…etc.
4. Tenant(data) service:
    a. Used mainly and heavily to get data of the user
    b. Get data based on User and Cloud service
    c. Can call user service (for auth). / cloud service / order service / payment service (to check payment)

Note : Here, assume each system is under ELB and integrated Autoscalling group, so can achieve performance without problems and also can manage cost (as autoscalling with min:1 to max: as per business requirement)

Deployment Process : - also used for developer understanding:

- Create different branch for each service
- User CI/CD (Jenkins) for each branch
  - Give git/svn url and Jenkins and configure the pre and post hooks
- After post/copy the code on EC2 instance start the application using PM2
  - With PM2 use - i option (for clustering)
- User different environments like dev/qa/prod
- If user lambda then user Serverless module
  - Which has also different config file based on dev/qa/prod
- Use environment variables for each config variable
  - Also use encryption to set the environment variable /decrypt on using them
  - Can also use S3 to store the config variables and on deployment first get all config variables/values from S3 and load to Environment
  - **So for adding/changing any config variable no need to give new build**

LLD: Low level (code based architecture) – also used for developer understanding:

- Create different services as below
  - User Creation : /user -- POST
  - User authentication: /authentication – POST
  - Order process : /user/order – POST
  - Order details : /order/id – GET
  - Orders List : /orders – GET – **User Id from Token**
  - Payment Process : /payment – POST
  - Payment details : /payment –POST (with ordere id)
  - **USER DATA : /data – GET**
- For USER DATA : /data API , Or Order Process Service (OR payment service)
  - User Factory Design Pattern
    - Get Object of related Cloud Service by passing CLOUD Name (i.e. AWS/AZURE/GOOGLE/IBM)
      - Create different CLOUD classes in config folder
      - Create Abstarct Class to be extended by each CLOUD SERVICE Class
        - For common method like get data
        - Or process order
        - Cloud login/authentication etc.
      - Create different interfaces as per required cloud service methods and signatures.
  - Each to enhance/maintain as different classes for each service
  - Also easy to integrate new service in future, just need to add that Cloud service in db and based on that create new class in config folder

- **Here, each service has separate database and developer must not call the other database internally, but instead of that class other service, related to that database**
    - i.e. if required user data then do not connect and call user data from db
        - but call /user/details api and get data from this way only
    - if required, order data in payment or data service then do not call order database directly,
        - but call /order/details api and get data from this way only

Each API Process/Flow document : (for developer):

- Follow MVC structure
- Express.js as Framework
    - Server.js is the main file of system
- MySql as database (use pool for db connections)
- Create all routes in route folder
- Use sequelize(mongoose) as ORM
    - Create all models in models folder
- Routes and Models are added from server.js (main file)
- Use some middleware for CORS and creating REQUEST_ID for each request uniquely, store in req.
- Request process (MVC):
    - Call Controller function from route
        - Controller only used for request validation and response
        - Use JOI module for request validation
        - Use helper functions (common) for response
        - Use LOGGER for each function entry and before exit, with REQUEST_ID
            - Easy to identify where actually request stopped
            - In each response give REQUEST_ID also
    - Call BO (business operations) file from controller
        - Do all business logics here only
        - Call third party (AWS/GOOGLE..) or any from here only
        - For any third party CREATE Files to communicate in SERVICE folder
        - From Service file – create FACTORY Object for social using LIB folder files
        - Can call other BO and DAO but can not call any controller (can not go reverse)
    - Call DAO (Data Access Objects) from BO for db operations
        - In DAO write/use all queries and db interactions
        - Can call any other DAO but can not go back (can not call any bo or controller)
    - DAO -> BO -> Controller give response back using callback
    - User promise and async/await, wherever required
    - Use promise.all wherever required
        - call USER.Authenticate
        - **then call order.detail, billing.info, /data api parallel if no dependeny else call in sequence**

        - **so, can call parallel and based on response take final decision.**
    - Use everywhere comment clearly for business logic and all above functions.
- Each developer use same branch for development (Git/Svn)
    - Each feature has different branch
    - On each commit it will be pushed to development branch using CI (Jenkins)
    - On QA can give build manually for testing at frequently as per requirement or when feature completed and on merging development with qa branch
    - On production also give build manually
    - Each environment has different databases
    - Database changes should be managed separately by each developer in same document (committed as queries –DDL and DMLs)