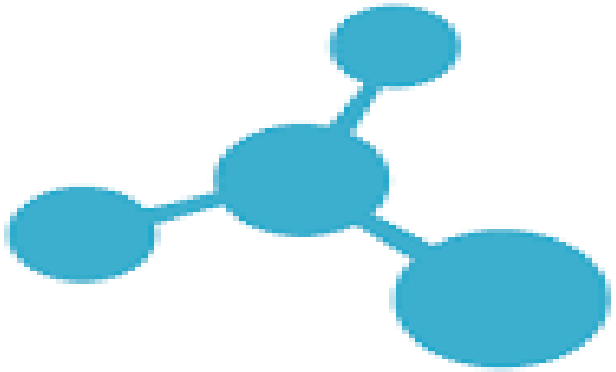
Welcome!



Moleculer MicroServices Framework

About Me : Prakash Badhe : prakash.badhe@vishwasoft.in

- Web Technologies Consultant-Trainer for 20+ years
- Passion for technologies and frameworks
- Skilled in technologies and tools
- Worked on server side Technologies and java script libraries
- Proficient in Microservices,Html5,Ajax,XML standards
- Working with Moleculer, NODE JS, Angular, Ember JavaScript frameworks and libraries

About you..

- **Skill-Sets Required : JavaScript,Nodejs web applications, REST API development.**
- **Your Job Role:**
- **Your objectives from this training:**
- **Your experience with MicroServices:**
- **Your experience with NodeJS, Moleculer or any other MicroServices framework:**
- **Your experience with other technologies:**
- **Experience in web applications development:**

Agenda

1. Java Script and NodeJS review
2. Introduction to MicroServices
3. Moleculer Framework Introduction
4. Create and build MicroServices with Moleculer Framework
5. Moleculer Framework Infrastructure services for MicroServices deployments
6. Discovery of services
7. Middleware caching
8. Serializers and transporters

Setup

1. Windows 7 or Windows 10 64 bit
2. NodeJS ver12.13.x
3. Adobe PDF Reader
4. Microsoft VS Code Editor
5. Chrome browser
6. Firefox Mozilla browser
7. Firefox add on : RESTED
8. MongoDB NoSQL Database server
9. MongoDB Compass
10. CURL
11. Live Internet Connection

We Proceed…Interactively

- The course material is shared with gitHub link.
- https://github.com/pbadhe34/Moleculer-Microservices
- **This contains instructions ,documents and sample programs.**
- Download this repo as zip and extract on your hard drive.
- We will follow this session practically with theory hand in hand.
- I demonstrate by showing my screen and actions with commands OR UI actions and speaking with my action .
- You follow the same on your system and verify the results as shown on my screen.
- In case of any queries/issues send me text on chat box or speak on your Mic.
- You practice on the case study exercises once considerable part is covered with demo and theory.

Java Script

- **JavaScript is the major client-side scripting language used in the Web applications in html pages.**
- **The JavaScript code is interpreted at runtime by the JavaScript engine in the browser and executed line by line.**

NODE JS

- Platform for JavaScript execution.
- Fastest JavaScript engine for execution.
- Platform for server side JavaScript web applications.
- Has the same underlying engine as Google Chrome browser has : V8 Engine for JavaScript.

Who is Using Node JS platform.?

- **PayTM (Managed by : One97)**
- **Netflix**
- **Wallmart**
- **CAPITAL ONE** : BANKING BUSINESS
- **Uber**
- **Paypal**
- **Linkedin**
- **Medium**
- **EBAY**
- **NASA**

NODE JS Features

- Built-in Package Manager : NPM and NPX to manage the dependencies.
- Asynchronous and Event Driven execution
- Non Blocking IO
- Single Threaded but Highly Scalable
- No Buffering
- Very fast execution with highest performance
- Open source
- Thousands of customized packages and tools run with NODEJS
- Built-in Networking capabilities to design web server
- Hundreds of tools designed to run on NODE JS

NODE JS applications

- Web applications : E-commerce, Social Media, B2C etc.
- Web services
- REST Services
- Build automation tools like webPack, gulp, grunt
- Payment Gateways
- Transaction Processing Middleware
- MicroServices
- IOT
- Real Time Chat
- Collaboration/Messaging applications

Start withy Node JS

- From command line
- NodeJs Version 12.13.x : node -v
- NPM ver. 6.12.0 : npm –v
- Run the node application : node server.js

Service Oriented Applications

- Applications as services.
- Applications developed with different programming languages and running on different platforms able to communicate each other and exchange information/data.
- Applications as Services collaborate together as separate modules and share data.
- SOAP web services
- REST services
- Shared data as xml, json, text etc.

Monolith applications

Monolithic application is a single-tiered software application where the user interface and data access code are combined into a single program running on a single platform as single process.

A monolithic application is self-contained and independent from other computing applications.

Monolithic application is designed without modularity.

Monolithic application is not scalable and difficult to maintain and develop with a team.

Monolith Advantages

- Simple to develop
- Simple to deploy
- Simple to troubleshoot
- Simple to scale with multiple copies.
- Suitable for development by a single resource

Monolith disadvantages

- Difficult to work across team.
- Difficult to scale dynamically.
- Difficult to upgrade to new requirements
- Difficult for Continuous deployments
- Difficult to increase the scope
- Run time performance issues
- Crashes the entire application due to small bugs.
- Porting/migration issues

Break the Monolith..

- Divide the single monolith application code into smallest possible isolated modules.
- The multiple modules can work as service applications and collaborate together to form the application backbone.

….Into MicroServices

- MicroService is an architecture pattern for service applications to collaborate together .
- Suitable to develop across team of developers.
- Easier to develop with smaller modules and smaller teams as well.
- Highly testable because of loosely coupling with other services/modules
- Easier to scale
- Easier to troubleshoot
- Flexible in code maintenance.
- Independently deployable units.

Drawbacks of MicroServices

- Manage the complexity of creating a distributed system.
- Manage the inter-service communication mechanism and deal with failures.
- Implementing requests spanning across multiple services is more difficult.
- Testing the services interactions is tedious.
- Troubleshooting at High level with integration issues .
- Increased resource consumption as memory , CPU since each service is a separate isolated process.

Decompose monolith application into MicroServices

- Strategies for decomposition
- Decompose by business capability and define independent services corresponding to business capabilities.
- Decompose by domain-driven design paradigm with isolated model design.
- Decompose by verb/action or use case and define separate services that are responsible for particular actions.
- Decompose by nouns/resources by defining separate a service that is responsible for all operations on particular entities/resources of a given type.e.g. Account, Users etc**.**
- Every service should have only a small set of responsibilities.

Data consistency with MicroServices

- To ensure loose coupling, each service has its own database.
- Maintaining data consistency between services is a challenge since the 2 phase-commit/distributed transactions is not an option for many services.
- The services can use shared database.
- In certain Enterprise applications, the replication across the database instances is done in background to maintain consistency

Infrastructure for MicroServices Execution

- **The environment for MicroServices deployment and execution.**
- **Collaborating platform for MicroServices**
- **Load balancing : dispatch the requests to available nodes by calculating the service load or in round robin manner.**
- Dynamic service discovery of services independent of the location/ip address by implementing service registry like domain name service (DNS) for web applications.
- API Gateway for Request-Response routing and security authentication.
- The fault tolerance features like Circuit Breaker, Bulkhead, Retry, Timeout, Fallback etc.
- Middleware for integration, Cache etc.
- Transporters for node to node communication in a cluster.
- Data Serializers and de-Serializers with custom schemas.

MicroService Deployment Platforms

- NetFlix
- Spring Cloud
- Google Cloud, AWS, Azure
- Moleculer Framework based on Node JS

Moleculer Framework

- Moleculer is a fast and powerful MicroServices framework for Node.js applications.
- It helps to build efficient, reliable & scalable services.
- Moleculer provides many features for building and managing the MicroServices.
- Asynchronous operations with promise-based solution.
- The request-reply paradigm
- Event driven architecture with balancing
- Supports plug-in/middleware system
- Built-in caching solution (Memory, MemoryLRU, Redis)

Moleculer Framework Features

- Loggers and Tracers
- API gateway
- Dynamic Service Discovery with service registry
- Load balancing
- Fault Tolerance with Circuit Breaks
- Serializers
- Master-less architecture, all nodes are equal

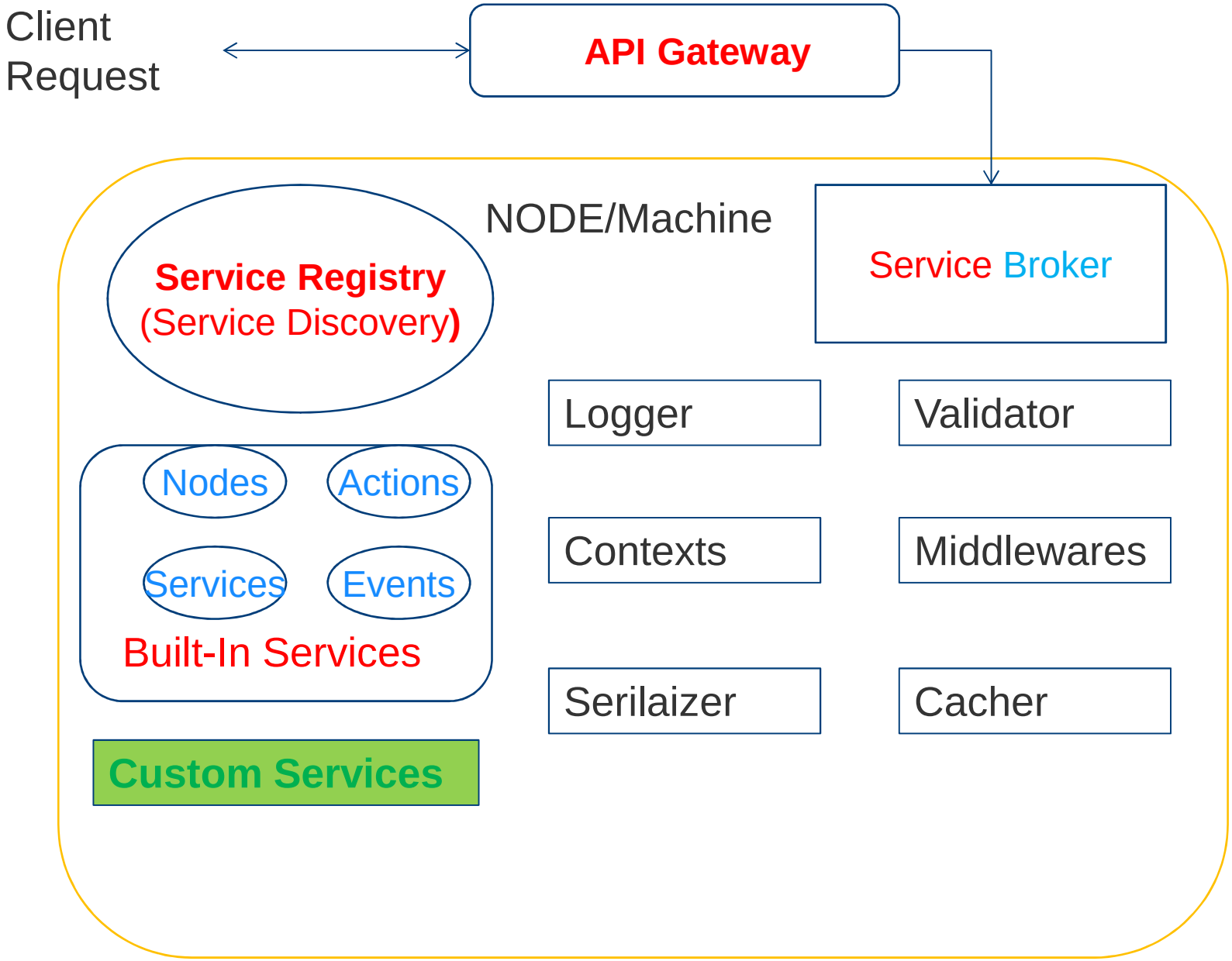
Start with Moleculer

- **From command line**
- Install the Moleculer CLI globally
- **>npm i moleculer-cli –g**
- Create a new Moleculer service project
- **>moleculer init project demo**
- **Press ENTER to all questions (accept default answers)**
- Once the sample project code is generated**.**
- **Run ‘npm install’ for dependencies**
- **And ‘npm run dev’ to start the node application.**

Moleculer Packages

- **moleculer-repl:**REPL module for Moleculer framework
- **Moleculer:** Moleculer framework
- moleculer-cli: Command line tool for Moleculer
- **moleculer-web:** API Gateway service for Moleculer framework
- **moleculer-db:** Moleculer service to store entities in database.
- **moleculer-db-adapter-mongo:** MongoDB native adapter for Moleculer DB service
- **moleculer-db-adapter-sequelize:** SQL adapter (Postgres, MySQL, SQLite & MSSQL) for Moleculer DB service
- **moleculer-template-project:** Common project template for Moleculer-based projects

Moleculer Services Architecture



API Gateway

- The moleculer-web supports the API gateway service for Moleculer framework.
- It is used to publish the services as RESTful APIs.
- The API gateway is the entry point through which the services are invoked.
- The API gateway coordinates with Service Broker to instantiate or identify the service instance on the Node machine.
- The API gateway supports centralized authentication, authorization to clients requesting the services.

Service Broker

- The **ServiceBroker** is the main component of Moleculer. Framework.
- It handles services, calls actions, emits events and communicates with remote nodes.
- The **ServiceBroker** instance is automatically created by the Moleculer framework in the beginning for every node.
- The broker provides load balancing to dispatch the requests to available nodes based on load balancing strategies.
- Supports the logging and tracking of Services.
- Use the Transporter to communicate with other nodes.
- The broker assigns the unique node-ID to the current node.

The default Services..

- The broker manages the default built-in services on every node.
- Nodes : Manages the no of nodes in the cluster: http://localhost:3000/api/~node
- To get the list of nodes : http://localhost:3000/api/~node/list
- The Node health status
- http://localhost:3000/api/~node/health
- Get the list of options
- http://localhost:3000/api/~node/options

The default services continued..

- Get the service metrics
- http://localhost:3000/api/~node/metrics
- Actions: List of action and mapping with the URLs
- http://localhost:3000/api/api/list-aliases
- Services : List of services on the node : http://localhost:3000/api/~node/services
- Events: Get the list of events
- http://localhost:3000/api/~node/events

The Moleculer Service-Runner

- The Moleculer Runner is a script defined by the CLI that helps to running the Moleculer projects.
- The script by default takes the **moleculer.config.js** file in the root of the project directory as configuration file with broker options.
- The **moleculer-runner** is called from the NPM script and it automatically loads the configuration file, create the broker and load the services.
- The **moleculer.config.js** is the default centralized configuration file for your node and all the services running on that node.
- Alternatively, you can declare your configuration as environment variables and manually start the broker and other services through the broker.

The Custom Service Operations and Mappings with Actions

- The DbMixin defined in db.mixin with moleculer-db package as DbService generates and registers the default actions with mappings as sepcified for eveyy domain model configured in the service definition.
- The actions(REST API) generated by default for the domain object

- **list**
- **find**
- **count**
- **create**
- **insert**
- **update**
- **remove**

This is the power of framework for MicroServices!

Additional Actions for Custom Services

- Additional custom actions can be defined with rest mapping, parameters and handlers to process the request and return the response.

```
updateBalance: {
   rest: "PUT /:id/account/add",
   params: {
            id: "string",
            balance: "number|integer|positive"
            },
   async updateHandler(ctx) {
   const doc = await this.adapter.updateById(ctx.params.id, { $inc: { balance:
   ctx.params.value } });
  const json = await this.transformDocuments(ctx, ctx.params, doc);
  await this.entityChanged("updated", json, ctx);
   return json;     }
},
```

Service Life Cycle Hooks

- Additional Life Cycle behavior can be configured by customizing the hooks in methods configuration with defauklt method names.
- **async seedDB**() –To generate initial data records.
- **async afterConnected**(): Handle the database connection established event.
- In action definitions the action hooks can be added to customize.
- **hooks: {**
- **before: {**
- **create(ctx) {**
- **ctx.params.quantity = 0;**
- **}**
- **}**
- The **route** has before & after call hooks to customize the actions.

The request-reply pattern for Message Exchange

- *Request–response is a message exchange pattern in which a requestor sends a request message to a replier system which receives and processes the request, ultimately returning a message in response.*
- *This provides two applications to have a two-way conversation with one another over a channel.*
- *This pattern is especially common in client–server architectures.*
- *The response processing is done in two ways mainly*
  - *Synchronous : The callers calls the api and waits for the response, This blocks the caller application till the response returns.*
  - *Asynchronous : The called service starts processing the response, but never bocks the caller service.*

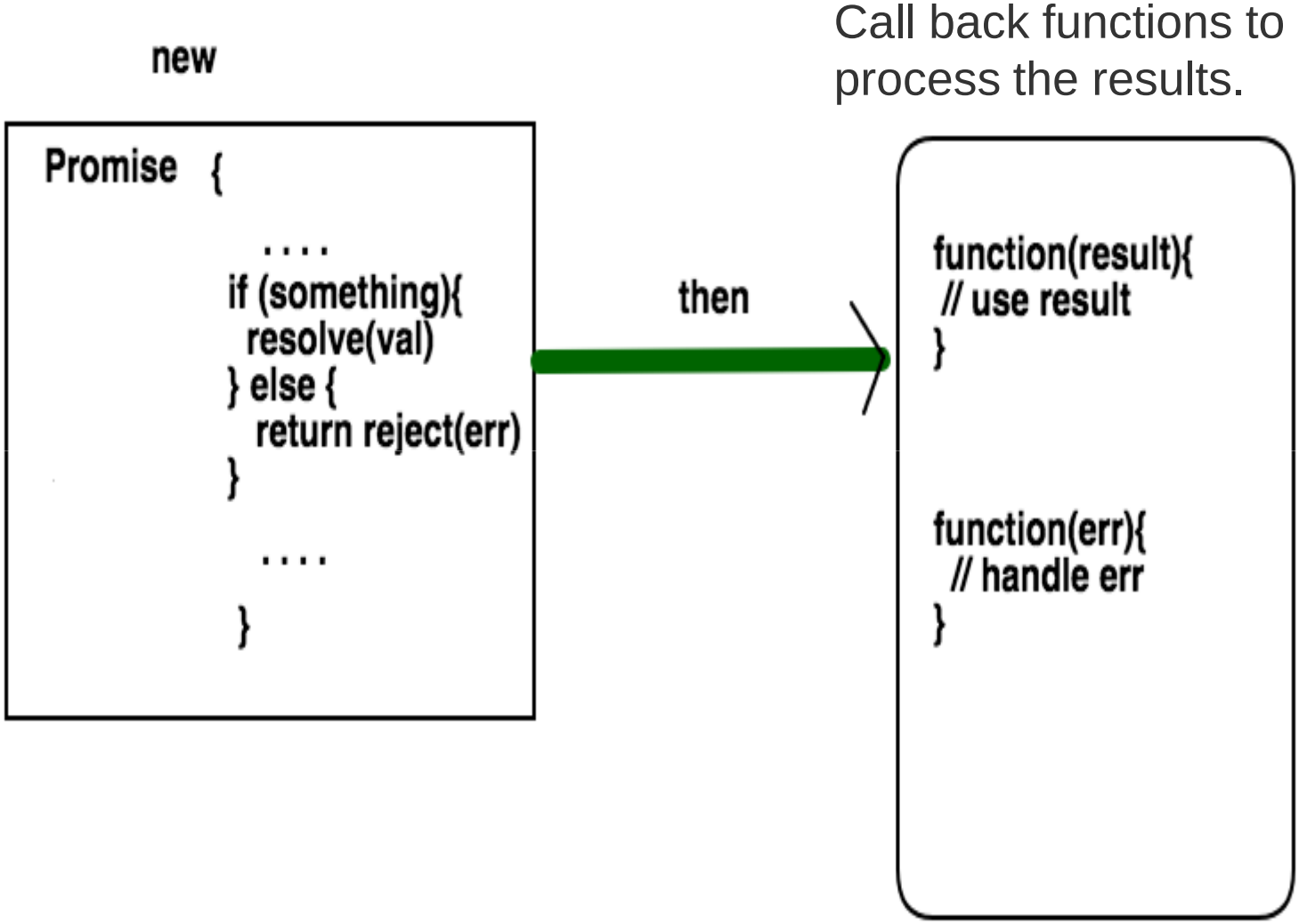
Event Driven Architecture

- Works n Asynchronous mode.
- The caller passes the handler - callback function which gets invoked by the called platform when the response has to be notified.
- The callback is never called by application, it is called by platform.
- The callback function is called in response to events raised.
- **How the caller keeps the track of return values from the asynchronies callback function..?.**

Promises to be kept…

- The key aspect of a **promise** is the return value from asynchronous invocations.
- A Promise is a value returned by an asynchronous function to indicate the completion of the processing carried out by the asynchronous function.
- In general based on the result of processing done by asynchronous function, the promise has three states
  - Error/Abort: The asynchronous function has rejected the promise.
  - Progress : The asynchronous function has processed certain percentage of job and still needs some more time to compete it
  - Success : The asynchronous function has successfully processed the job and there is return value in the promise.

The state of Promise



Promise Me..

- A **Promise** is a proxy for a value not known when the promise is created.
- It allows to associate handlers to an asynchronous action's eventual success value or failure reason.
- This lets asynchronous methods return values like synchronous methods: instead of the final value, the asynchronous method returns a *promise* for the value at some point in the future.
- **That is the asynchronous method has promised to return a value**.

Use of Promises

- **Most of remote invocations such as calling external services which don't have any control on the return value to the caller by the called service is designed to returns the promise to the caller in asynchronous mode.**
- The promise that performs async operations should call any one of the two methods **resolve** or **reject.**
- The called program should decide of when and where to call these functions. If the operation is successful, pass that data to the code that uses that promise, otherwise pass error

Result of Promise

- The code which uses a promise should call **then** function on that promise.
- The function takes two anonymous functions as parameters.
- The first function executes if the promise is **resolved(success)** and the second function executes if promise is **rejected**.(error)
- What happens if you try to access the value from promise before it is resolved or rejected. Then promise will be in the pending state**.**

Dynamic service discovery

- How to invoke the service : by sending request like this
- http:://192.168.1.18:/api/account/update
- This is hard coded approach.
- For dynamic nodes the address varies, in case of multiple instances of services in the network.
- Moleculer framework has a built-in module responsible for node discovery and periodic heartbeat verification.
- The discovery is dynamic meaning that a node don’t need to know anything about other nodes during start time.
- When it starts, announces it’s presence to all the other nodes so that each one can build its own local service registry.
- In case of a node crash (or stop) other nodes will detect it and remove the affected services from their registry.
- **This way the requests will be routed to live nodes as guided from the service registry..**

Discovery Modes

- This dynamic discovery is very similar to **Domain Name Service** provided for Internet web sites address mappings.
- This dynamic discovery of services is configured with the service registry configurations defined in the service js file.
- **Local :** Local discovery (default option) uses the transporter module to exchange node info and heartbeat packets with other nodes in the network.
- **Redis**: Redis-based discovery uses a dedicated connection with the external Redis server to exchange discovery and heartbeat packets.
- This approach reduces the load over the *transporter* module, it's used exclusively for the exchange of the request, response, event packets.
- **Etcd3:** Very similar to Redis-based discovery. It stores heartbeat and discovery packets at etcd3 server. etcd3's lease option will remove heartbeat info of nodes that have crashed or disconnected from the network.

Load balancing

- Moleculer supports built-in load balancing strategies.
- If a service is running on multiple node instances, ServiceRegistry uses these strategies to select a single node from the available ones.
- To configure strategy, set **strategy** broker options under **registry** property. In the config file.
- **registry:**
  strategy: "RoundRobin"
  }
- **Other strategies are "Random", "CpuUsage", "Latency", "Shard“ etc.**
- **Customized Strategy** class inherited from **BaseStrategy** for custom strategies of load balancing can be added.

Middlewares

- To customize the life cycle of services at global level..i,e. common options to all the services middleware are defined with customized life cycle hooks in config file.
- It’s same as plug-ins in other frameworks.
- The middleware is an Object with hooks & wrapper functions.
- It allows to wrap action handlers, event handlers, broker methods and hook lifecycle events.

Networking with Transporter

- To communicate with other nodes (Service Brokers) need to configure a transporter.
- Most of the supported transporters connect to a central message broker that provide a reliable way of exchanging messages among remote nodes.
- These message brokers mainly support publish/subscribe messaging pattern.

Transporter

- Transporter is an important module if you are running services on multiple nodes.
- Transporter communicates with other nodes.
- It transfers events, calls requests and processes responses …etc.
- If multiple instances of a service are running on different nodes then the requests will be load-balanced among them.
- The communication logic is independent of transporter.
- It means that you can switch between transporters without changing any line of code.
- The built-in transporters in Moleculer framework: TCP, NATS,Redis, Kafka,MQTT,AMQP, etc.
- Custom Transporters can be added.

Serializers in Moleculer

- Transporter needs a serializer module which serializes & de-serializes the transferred packets.
- The default serializer is the JSONSerializer
- The other built-in Serializers are “Avro", "ProtoBuf", "MsgPack", "Notepack", "Thrift“ etc.
- Custom serializer module can be created and **added in config file.**
- Recommended approach is to copy the source of JSONSerializer and implement the serialize and deserialize methods.

Caching in Moleculer

- Moleculer has a built-in caching solution to cache responses of service actions.
- To enable it, set a cacher type in broker option and set the cache: true in action definition what you want to cache.
- The cacher implementation is configured at global level in config file as Memory or MemoryLRU.
- The Cacher improves the performance of services.

Thank You!!

Happy Microing..