SHIPPING COMPANY ERP EXTENSION

FRAMEWORK

HIGH LEVEL ARCHITECTURE

|  |  |
| --- | --- |
| **Author** | Nikola Stjelja |
| **Email** | nstjelja@gmail.com |
| **Date of last change** | 28.12.2014 |
| **Version** | Final |
| **Client** | gzTeam |

Table of Contents

[OVERVIEW 3](#_Toc407530953)

[GENERAL ARCHITECTURAL PRINCIPLES 4](#_Toc407530954)

[LINUX BASED INTEGRATION POINTS 4](#_Toc407530955)

[NODE.JS FOR INTEGRATION COMPONENTS 4](#_Toc407530956)

[COMPONENT DECOUPLING 5](#_Toc407530957)

[OFF THE SHELF OPEN SOURCE COMPONENT INTEGRATION 5](#_Toc407530958)

[.NET DEVELOPMENT OF USER CENTRIC APPLICATION SYSTEMS 6](#_Toc407530959)

[COMPONENTIZATION OF APPLICATION DEPLYOMENTS 6](#_Toc407530960)

[HIGH LEVEL ARCHITECTURE MODEL 7](#_Toc407530961)

[BRUSH STROKE OVERVIEW 8](#_Toc407530962)

[CORE ERP FRAMEWORK 9](#_Toc407530963)

[CORE ERP WEB SERVICES 10](#_Toc407530964)

[LOGGING SYSTEM 11](#_Toc407530965)

[INTERNAL DATA INTEGRATION SYSTEM 12](#_Toc407530966)

[INTEGRATION APPLICATION ADMINISTRATION SYSTEM 13](#_Toc407530967)

[SILO APPLICATIONs (BOUNDED COnTEXT) 14](#_Toc407530968)

[SPECIFIC ARCHITECTURE PATTERNS 16](#_Toc407530969)

[DOCKER APPLICATION CONTAINER 16](#_Toc407530970)

[INTEGRATION SERVICES MANAGEMENT 17](#_Toc407530971)

[SECURITY MODEL 18](#_Toc407530972)

[FRAMEWORK UTILIZATION GUIDELINES 21](#_Toc407530973)

[GLOSSARY OF TERMS 22](#_Toc407530974)

# OVERVIEW

The following document will provide a base framework architecture for the extension of the current Shipping company ERP system.

The current state of the system, at the time of the writing of this document, was a single close sourced monolithic application deployed on top of a DB2 database. The current ERP system provides all the base functions required by the shipping organization, but it lacks some fundamental features and it’s the desire of the company to extend it with more manageable and extensible system.

On top of the basic ERP operations the current system relies on several third party data providers which provide various types of data sources (XML feeds, manual downloads and web services) with real time (or as close as possible) information about various shipping statues of ships and containers currently an route around the world (detailed description of third party shipping providers can be found in the requirements document).

The ERP extension architecture document will focus on providing an architectural framework used to solve the following technical and process problems:

1. Unified system wide event logging (who what and where inside the entire system)
2. Unified third party data retrieval and processing system
3. Unified system event generation based on data changes inside the original ERP system
4. Framework for extending the original system with application silos based on event generation and loose core system integration based on published core system web services and message queues.
5. Management system for all integration components (e.g. custom built ETL systems)
6. General DevOPS strategy for managing and deploying applications

The document will be organize into the following sections:

* General architectural principles
* High level architecture model
* Specific architectural models
* Framework utilization guidelines

# GENERAL ARCHITECTURAL PRINCIPLES

Regardless of the specific technological implementation, functional role and system area of a specific architectural slice the following architectural principles will be followed during the implementation of this architectural framework and will serve as a basis for its future extensions.

## LINUX BASED INTEGRATION POINTS

The core of the framework will be formed by an custom built ETL system centered around the DB2 database (ERP Web Services) and Message Queue (RabbitMQ) and various silo databases specific to internal third party systems.

Given the large amount of data (Fleet tracking and shipping, general business data) generated by the system the core of the ETL processes must be developed in such a way that they are:

* Cheap to deploy ,manage and scale based on the system load
* Easy to write and change
* Can manipulate large amounts of data and react to a high volume of requests
* Can connect to various data end points and manipulate different data sources

The system must be deployed on a customizable platform for which there multiple open source solutions specialized in large scale system deployments, management and data manipulation.

Thus all supporting servers (logging, load balancing, second level databases and integration services) will be built on top of Linux servers (Ubuntu), managed and deployed with Docker (Shipyard) and all the application integration code will be built using Node.JS.

Although all the above can be developed using a purely Windows (.NET) based system (or Java based integration services) it is much faster and easier to manage, and write those supportive applications and server components using a Linux based stack.

## NODE.JS FOR INTEGRATION COMPONENTS

Node.JS is a fast rising server application framework best suited for highly responsive message driven system. Based on its architecture it is capable to handle and respond to a much higher request load then a standard application server.

The technology has reached fast maturity, thus providing a good based of ready available solutions, a wide developer pool and easy to use programming language (any web developer can switch easily to Node.JS given that it uses the same programming language – Javascript).

Leveraging the dynamic and processing oriented functions of Javascript (its more functional than object oriented) the language is very suited to build and enable data processing between web systems, databases and various output endpoints (message queues, and databases). Especially so if those endpoints utilize JSON as a transfer protocol thus keeping much of the stack on the same language.

## COMPONENT DECOUPLING

Moving from a single monolithic system to a framework of decoupled tiers where each tier provides either a supportive function (data transformation and passing) or a business function (reporting engine, fleet tracking application, ERP web services) it is important to:

* Provide a simple way to integrate different endpoints in a collective whole where:
  + There is no loss of data
  + Failure of one component doesn’t affect the whole system
* Components communicate with a limited set of components serving as gatekeepers for the wider organization

Principally each component will have the following three connections only:

* A RabbitMQ message queue cluster which will serve as a system wide event originator. Components will never communicate directly, they will use the message queue to send messages (except for the ERP Web services layer)
* Local (silo oriented) data sources and data stores (e.g. application specific services and application specific databases – Fleet Tracker APP and DB)
* ERP web services

## OFF THE SHELF OPEN SOURCE COMPONENT INTEGRATION

Leveraging current team knowledge, and mirroring the most common industry practices, as much as possible use must be made of existing open source libraries, languages, operating systems, monitoring tools and application servers in order to minimize cost (development and operations) and reduce the overall development cost and complexity.

The following open source components are strongly suggested to be used:

* Ubuntu linux
* HAProxy
* MongoDB
* Node.JS (with Express application framework)
* Docker (Shipyard)
* JQuery (and all related libraries)
* Knockout.JS
* ASP.NET MVC
* Entity Framework
* MySQL

## .NET DEVELOPMENT OF USER CENTRIC APPLICATION SYSTEMS

Although all the secondary elements of the system should be developed on Linux based system due to the speed of development, ease of integration and scalability all business relation application should be developed using .NET technologies due to the specific speed and easy which the Microsoft .NET toolkit provides for the development of Enterprise Application solutions.

All web related applications should be developed with the following stack:

* ASP.NET MVC,
* Jquery,
* Knockout.JS (AJAX),
* Entity Framework
* SQL Server,
* Lightswitch (Silverlight) - administration

The current Microsoft toolkit (with the extension of the NuGET package manger) provides easy integration and enough complete libraries, online documentation to easily and fast solve any business related application problem. The technological support in form of intellisense and component integration runs deeply trough the entire stack.

Also these are the most widely used .NET technologies at the time of writing of this document thus ensuring a wide and cheap pool of developer labor.

Each application requires at least some level of administration (security, configuration etc). By storing all those administrative options inside the application related database it is possible to quickly and easily build separate

## COMPONENTIZATION OF APPLICATION DEPLYOMENTS

The ERP extension application framework will be built atop two generic platforms:

* Node.js (Linux)
* .NET (Linux)

To simplify the application development each platform deployment and management solution will be delivered through the use of components - tight coupling of applications, servers and libraries.

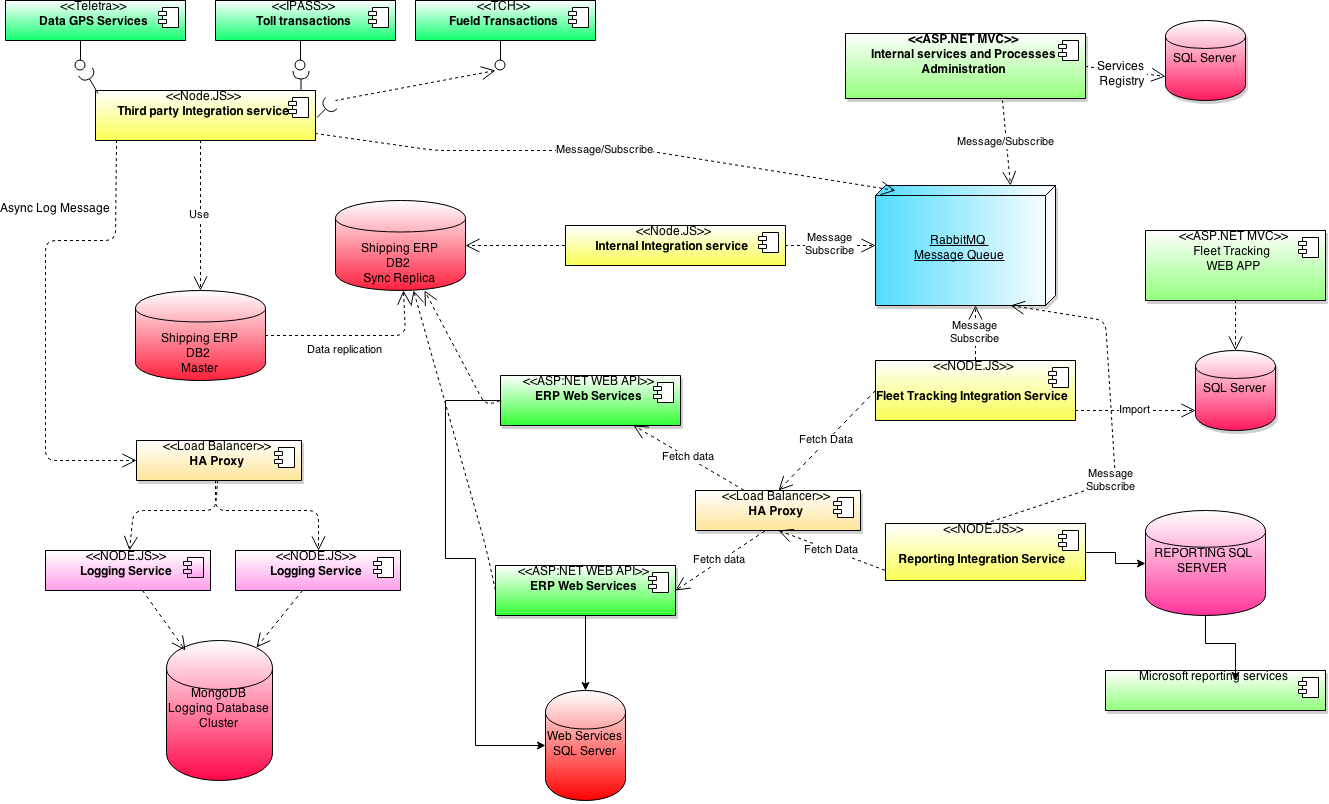
On the Linux platform the usage of Docker to build single units of deployment where the application, code and all dependencies will be bundled into separate Docker images and deployed as needed and when needed as single units with no outside dependencies.

On the Windows platform with deployment technologies will be used:

* NuGET for storing and publishing shared libraries
* WebDeploy for deploying and installing applications on top of ISS

The main goal with this setup is to simplify completely the way applications are deployed and managed in production.

# HIGH LEVEL ARCHITECTURE MODEL



The High level Architecture model of the system can be logical thought out with the following parts in mind:

* Core ERP database cluster
* Core ERP web services
* Logging system
* Third party data integration for the Core ERP database cluster
* Internal data integration system
* Integration application administration system
* Silo applications (business units)

Given the relative size of the model each section will be described separately preceded by global, brush stroke, overview of the system.

## BRUSH STROKE OVERVIEW

The framework will be centered around a replicated instance of the DB2 database (thus freeing the master database to work normally and do not increase its load by adding third party systems and processes continuously pinging it for data).

All integration points with the core system will be executed from this database as the main data source. On top of it two integration points will be built:

* Internal integration service, a node.js integration application which will raise period event when data is changed in the core database
* ERP web services which will provide core data access and business functions on the DB2 database (fleet data, single sign on etc)

Data access from outside third party services (GPS fleet tracking data and altri) will be handled by a node.js integration service which periodically query those services (based on a predefined, updatable configuration) and integrated a filtered data set into the main database.

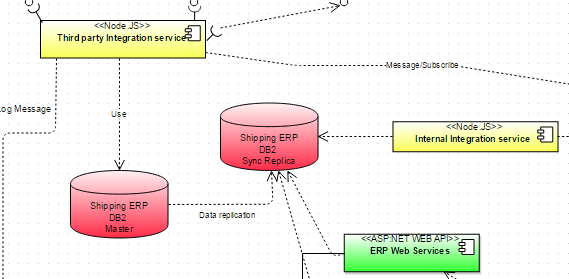
When data is changed in the core system the internal integration application will create specific message which will be sent into the RabbitMQ cluster. It will be responsible to disseminate the event to interested parties inside the system (e.g. Fleet tracking or accounting reporting).

The RabbitMQ system will be responsible for handling the integration and communication between components. As a specific example the Integration Management System will listen to Create/Stop message launched by specific integration applications going live or dying in the system, register those state and when the human administrator of the system issues a specific command to a specific integration application it will send that message trough the RabbitMQ system indirectly to the targeted application which will execute the received message at the earliest possible moment (thus enabling it to finish its work and making the administration of the entire interaction system much more easier to administer and less prone to IP and port management issues).

All published web services will be available trough a Load balancer thus enabling their easy scalability due to increased or decreased workload.

If at any point a system wide logging event is raised (e.g, data import from third party systems) the Logging subsystem will be able to asynchronously handle and store large amounts of structured logging data into its MongoDB cluster.

## CORE ERP FRAMEWORK



The Core Framework element of the ERP system is composed of the following elements:

* ERP Master and Replica database
* Third party integration service
* Internal integration service
* ERP Web Services

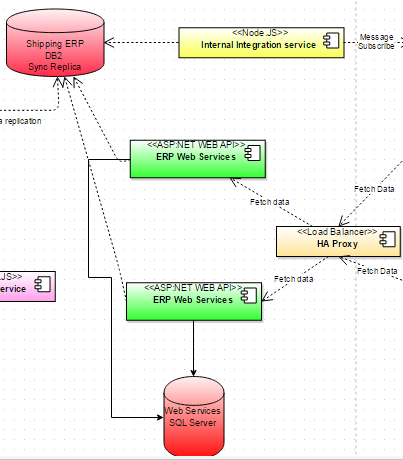
ERP Master database is the normal central ERP system DB2 database. In order to keep it functioning at its current performance level a replica will be set (using the DB2 replication technology). The replica database will serve as the data integration point for all internal components.

Beside having the most up to date data set of the main database it will support a smaller set of table which will store a log of all updates, those tables will be used by the Internal integration system to react to data changes without having to search and query the entire database all the time.

Third party integration services ( on node, many nodes) will provide specific , configured integration points and data retrieval (processing and filter) with all third party services which provide data to the core ERP system.

Internal integration services will query the transfer log table of the ERP replica database and generate event messages inside the Messaging System whenever a specific part of the ERP data set is change. The generated message will have only the minimal amount of data needed to be useful to subscribing systems. All extra data retrieval will be executed by issueing calls to the ERP Web services layer.

## CORE ERP WEB SERVICES



The ERP web services will be built with ASP.NET MVC Web Services (using the predefined .NET technology stack) and will have the following general characteristics:

* Stateless
* REST API (Json as the default END point)
* Versioning will be provided with version URLs
* Will utilize the same identity security framework for web service clients as all the rest of the .NET applications (e.g. there is no difference from the security framework point of view between a human user or a machine user)
* Will be scaled using a load balancer (e.g. HAProxy)
* Will connect and store web service specific configuration and data inside their own database
* Will access the ERP DB2 replica for data

Regardless of their intent each web service will follow the same request response structure thus enabling a much more easier integration between different clients.

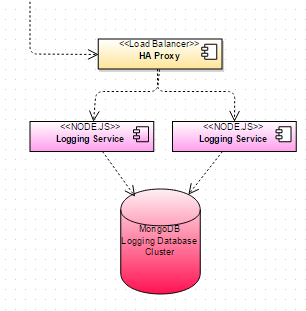
Each request will be composed (similar to basic HTTP messages) of a header (identity data, generic message parameters) and a body (specific message parameters). Similar each response will have the same structure a header (response status, and generic system messages) and a body (specific message response).

A common client library with all the message response and requests and .NET client classes will be built and published via a NuGEt server in order to easy client integration with web services and reduce code duplication within the system.

The following generic services will form the core of the initial web service delivery:

* Single sing on (authenticate ERP users with credentials from the ERP system)
* Fleet tracking data
* Accounting data

## LOGGING SYSTEM



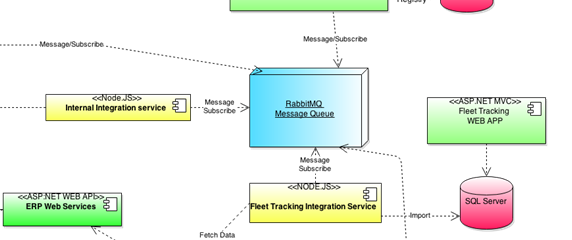
The logging system, in the hierarchy of the architecture framework will form a third level system whose purpose is to catch and log structured data into the logging database.

MongoDB is a NOSQl data store well suited for fast reads and writes of document oriented data without having to defined (or even keep) that structure in advance of the logging event. It provides fast and reliable query and aggregation options.

It uses JSON as the model for storing data and Javascript as its query language. It provides easy to use clustering and sharding features.

The input (adapter) part of the logging system a cluster of Node.JS REST web services will be deployed behind a HAProxy load balancer. Node.JS asynchronous model and javascipt orientation provides a non-blocking and fast way to save data inside the logging database.

## INTERNAL DATA INTEGRATION SYSTEM



The internal data integration system provides the principal pivot point for all extensions of the core ERP system and internal component integrations.

Its is composed of the following three components:

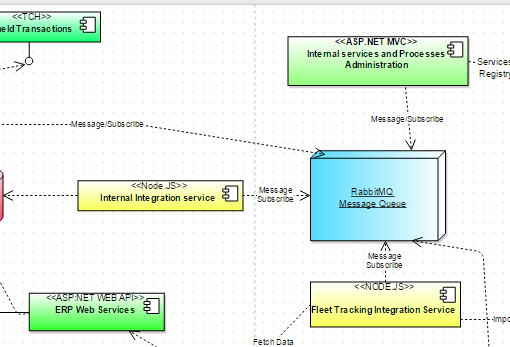
* RabbitMQ Message queue cluster
* Internal integration service
* ERP Web Services

At the center of everything lies the RabbitMQ message cluster, a distributed, fault safe enterprise ready message queue. Every system wide event from the change to a data set in the core database to specific application events or commands will pass through it. All components will have an integration point with it and trought it will interact with the majority of the system.

The Internal integration service will query the change log database of the replicated DB2 data store and will create appropriate system wide events inside the message queue system.

Applications like the Fleet tracking system will subscribe to those events and request the appropriate data from the ERP web services layer, transform it and place in their separate storages.

## INTEGRATION APPLICATION ADMINISTRATION SYSTEM



The most common application type deployed within the system is the face less long running integration service, which will developed as a Node.JS application and deployed as a part of an Docker container.

Each integration point will be built in such a way that there could be multiple instances of each integration service running at the same time (each configured to covere just a subset of transformation tasks based on the loads it must serve). More over integration points can go live, die, react or send events in the system at any point of time.

Managing all those integration point will be a problem. First of all fixing a random poping up and dieing nodes on specific IP addresses and forcing direct connections is an extra cognitive and administrative burden. Not only a docker instance must be pushed to the production and setup to work (if any setup is required) it also must be registered within a common administration system.

Another solution would be to automate the deployment and orchestration of Docker containers with a deployment and orchestration tool (like Flocker or Shipyard) and to let integration services make them selves discoverable to an centralized administrative application and react to in messages sent from it through the messaging system.

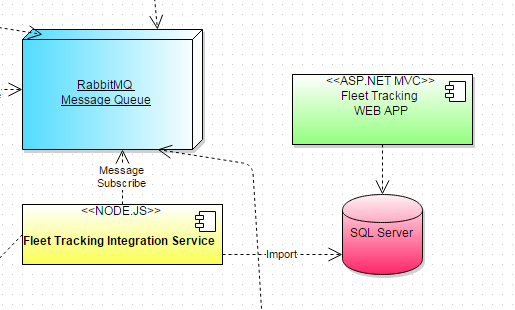
A sample use case would go like this:

1. Integration service XYZ goes live
2. It sends its Go Live event with its Unique Domain name, command list description to the message queue
3. The Administrative app react to the go live event of the instance, registers it and its command list (think MAN pages)
4. The Administrator of the system sends commands through the Admin interface (over the Messaging system) and waits for feedback from the integration service.
5. The Integration service goes down (or dies or finishes work and has no reason to take resource) and it stops working it sends a die event.
6. The Administrative app reacts to the Die event and deregisters the service from its registry.

At no point in time the specific IP address or port of the Administrative application or the Container instance is registered anywhere, all communication is executed trough the messaging system.

Given the wide array of possible integration services the best way to interact with each service is through a command line type of integration (Linux commands). Each integration service will publish a manifest of its commands which will be registered by the administrative application, in turn the administrator of the system will send those commands over the messaging queue to the integration service.

## SILO APPLICATIONs (BOUNDED COnTEXT)



The end goal of the Shipping company architecture framework is not to have a cutting edge, distributed data delivery and transformation system. It is only the founding, the data highway upon which the business cities, the actual goal of this system, are built.

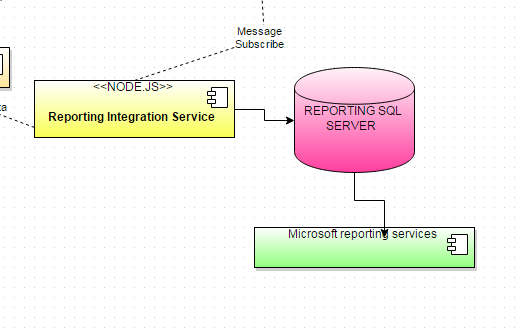
Borrowing a term from Domain Driven Design , Bounded context, each business centric extension of the system e.g. a user operated endpoint which operates as a standalone operative stack will deployed as a semiautonomous software solution.

Each bounded context will be deployed with its own data store and applicative UI. The data store will be filled with data from the Main system by reacting to events with its own Integration service and will execute changes, extensions and other bounded context specific domain changes in it without affecting the rest of the system.

Since the application UI is business centric, a .NET related approach will be taken since the .NET development stack is best suited for the speedy development of Business related applications.

The above pictured stack covers the Fleet Tracking Bounded Context (Application Silo) and is composed of the following tiers:

* Integration service – NODE.js application that reacts to events from the message queue and fetches data from the ERP Web Services layer, transforms it and persists it into the Fleet Tracking SQL Server database
* Fleet Tracking SQL Server database – Database with all related fleet tracking information
* Fleet Tracking WEB – ASP.NET MVC Fleet tracking application

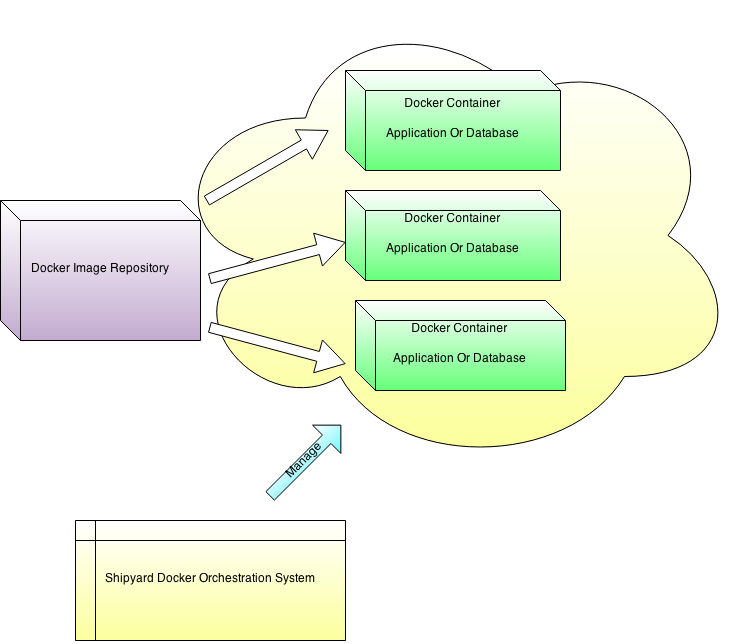


Another bounded context example is the Reporting Profitability engine which will gather data related to the accounting domain transform it into vectored views in the reporting SQL server and publish reports upon the gathered data.

# SPECIFIC ARCHITECTURE PATTERNS

Beside a model/deployment view of the architecture there a specific technical patterns applied within the wide architectural framework which warrant specific notice in this document.

## DOCKER APPLICATION CONTAINER

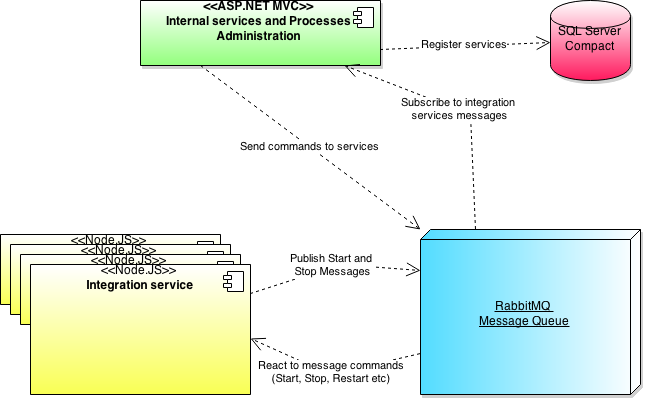


Any distributed application architecture which focuses on loosely coupled, bounded context, applications, background services and long running tasks encounters severe and difficult problems during the deployment and management of nodes on multiple servers (including ports, firewalls, permissions, configurations and library dependencies which may or may not be conflicting between different tiers).

Docker comes to the rescoue providing (Linux and soon Windows based) application virtualization. Each application stack (e.g. Node.JS, specific libraries and node specific databases) can be built and deployed as part of lightweight virtualized sandboxes where everything inside them is isolated from the rest of the deployed on OS.

Thus deploying a complex application stack multiple times becomes a single line of code in the command line or just a click by using a graphical management tool like Shipyard.

## INTEGRATION SERVICES MANAGEMENT



As noted before in the High level architecture model there will not be a direct tie between the administrative UI of all the Integration services (long running tasks or processing deamons). The complexity of manually managing connections, live and die events or specific instance commands is generally not cost effective.

By making each service responsible for its discovery and contract (e.g. command) communication and by leveraging the communication between each service and the admin application only through the RabbitMQ message queue application it is possible to scale the operations and management of a diverse ecosystem of distributed processing units with incurring a great development and production cost.

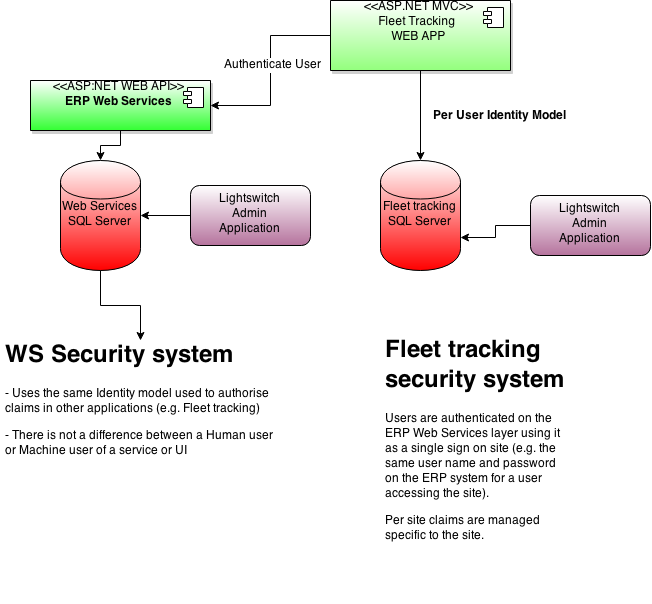
Thus the system in its essence would be quite simple to develop, produce and manage but will render great cost savings in the increased ease of administration and faster task resolution.

Although the main patter is to build specific UI for administration (click based interaction) given the wide variety of services in play it is much more simpler and easier to administer to use a text based input and command approach, must similar to the linux shell.

Each service will publish its commands list (with patters of use like the MAN command on Linux) which will be stored on the Administrative UI.

The administrative UI will then enable the user to type and send commands directly through the Messaging network to each specific node, or node group.

## SECURITY MODEL

Security is pervasive, it covers all layers of the system and travels deep into every tier and node. The prescribed architectural framework prescribes the following two general types of security interactions:

* User accessing bounded context applications
* Systems accessing each other services

The proposed model will answer the above mentioned security interactions with the following framework:

* The same security object model will be used everywhere
* Users will be managed in the ERP DB2 database as are they currently and authenticated using single sign on services exposed by the ERP Web Services layer
* Context specific permissions will be kept in the context local data store
* Web services will authenticate users(e.g. other services) using the same security model.

Thus the same security model can be:

* Developed as a single database solution (e.g. table sets)
* Developed as a single deployable library (over NuGET and NPM for .NET and Node.js specifically)

Thus each developer will use a very simple but powerful security model that generalizes the important elements while keeping local customization at its maximum.

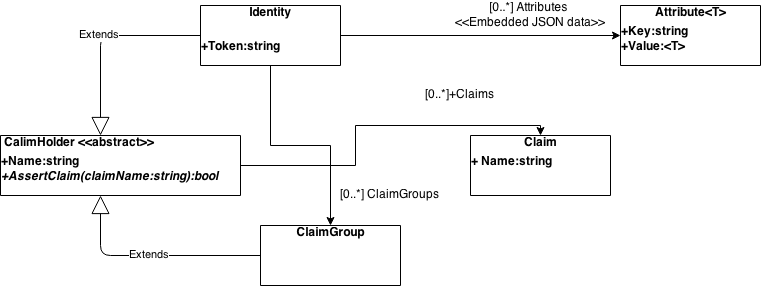
Security is based on the Identity – Claim model. An Identity is an entity (human or machine) that makes claims on operations on resources. The security model will ascertain that the specific identity with given credentials has the required claim in order to execute and operation on a specific resource.

The operation can be access a web service, see a link or report on a web page. From the security standpoint it is relevant.

The security model will be managed within the database system of each bounded context. In order to simplify application development all administrative tasks on each bounded context database should be done trough custom developed Lightswitch applications.

Ligthswitch is a .NET technology for building quickly and easily database oriented applications where the only goal is to see and change data from the database quickly and without too much spending effort on the UI or long running processing of said data.

Application administration, a long running second level task, is well suited for such application types given the ease and speed they can be developed.



The Identity domain model is quite simple. It is composed of the following entities:

|  |  |
| --- | --- |
| **Entity** | **Description** |
| ClaimHolder | Abstract entity defining a Name (name of the holder) , a collection of Claims and a method to ascertain if the Holder has indeed such a Claim in its collection.  The assert claim method is abstract thus each claim holder child can implement its own specific claim assert logic. |
| Identity | A user or machine assessing the system. It can have its own claims or can be attached to several claim groups. The AssertClaim method will check for claims in all those collections.  An identity can be extended with a key value attribute chain accepting any and all descriptive data. The only caveat is to embed the Attribute list as a JSON object when persisting the Identity into a persistence store instead of storing it into their specific tables. |
| ClaimGroup | A simplified collection of Claims. |
| Claim | A specific operation that can be performed on a specific resource. E.g. “Filter trough accounting data” or “See fleet tracking data”. |

# FRAMEWORK UTILIZATION GUIDELINES

The architecture prescribed here defines a data delivery and application construction infrastructure upon which organizational and technical extensions can be built.

It was a critical design thought to design a framework that can be:

* Easily extended
* Easily change the interaction mode of components
* Easily change and introduce new technologies into the development stack
* Easily orchestrate and monitor

The following guidelines should be followed when implementing an organizational architecture upon this framework:

* Focus on keeping data transformation to a minimum (standardize on JSON and Javascript where applicable on the stack)
* Focus on building stateless application blocks
* Focus on indirect node communication, direct communication should be set to a minimum and then only trough rest based web services
* Focus on minimizing the development time spent on administrative (non core business value) applications and infrastructures
* Reuse existing open source components where possible and plan future replacements of existing solutions with possible open source solutions when the system matures enough.

In its essence, Linux development is all about networks, interactions and distributed micro services interactions. That is what is good for. Its cheap, malleable with a general large amount of available technical documentation and support. .NET as a technological stack is great for building and delivering Enterprise software which is business oriented.

By combining the strength of both technologies a successful and scalable system, with reduced operational cost can and will be built by following the guidelines specified in this architectural framework.

# GLOSSARY OF TERMS

|  |  |
| --- | --- |
| **TERM** | **DEFINITION** |
| DevOps | DevOps (a portmanteau of "development" and "operations") is a software development method that stresses communication, collaboration, and integration between software developers and Information Technology (IT) professionals. DevOps is a response to the interdependence of software development and IT operations. It aims to help an organization rapidly produce software products and services. |
| Docker | Docker is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating system–level virtualization on Linux. Docker uses resource isolation features of the Linux kernel such as cgroups and kernel namespaces to allow independent "containers" to run within a single Linux instance, avoiding the overhead of starting virtual machines. |
| Entity framework | **Entity Framework** (EF) is an open source[object-relational mapping](http://en.wikipedia.org/wiki/Object-relational_mapping) (ORM) framework for [ADO.NET](http://en.wikipedia.org/wiki/ADO.NET), part of [.NET Framework](http://en.wikipedia.org/wiki/.NET_Framework). |
| ETL | In [computing](http://en.wikipedia.org/wiki/Computing), **Extract, Transform and Load** (**ETL**) refers to a process in [database](http://en.wikipedia.org/wiki/Database) usage and especially in [data warehousing](http://en.wikipedia.org/wiki/Data_warehouse) that:   * [Extracts](http://en.wikipedia.org/wiki/Data_extraction) data from homogeneous or heterogeneous data sources * [Transforms](http://en.wikipedia.org/wiki/Data_transformation) the data for storing it in proper format or structure for querying and analysis purpose * [Loads](http://en.wikipedia.org/wiki/Data_loading) it into the final target (database, more specifically, [operational data store](http://en.wikipedia.org/wiki/Operational_data_store), [data mart](http://en.wikipedia.org/wiki/Data_mart), or [data warehouse](http://en.wikipedia.org/wiki/Data_warehouse))   Usually all the three phases execute in parallel since the data extraction takes time, so while the data is being pulled another transformation process executes, processing the already received data and prepares the data for loading and as soon as there is some data ready to be loaded into the target, the data loading kicks off without waiting for the completion of the previous phases.  ETL systems commonly integrate data from multiple applications(systems), typically developed and supported by different vendors or hosted on separate computer hardware. The disparate systems containing the original data are frequently managed and operated by different employees. For example a cost accounting system may combine data from payroll, sales and purchasing. |
| HAProxy | **HAProxy** is a [free, open source](http://en.wikipedia.org/wiki/Free_and_open-source_software) [high availability](http://en.wikipedia.org/wiki/High_availability) solution, providing [load balancing](http://en.wikipedia.org/wiki/Load_balancing_(computing))and proxying for [TCP](http://en.wikipedia.org/wiki/Transmission_Control_Protocol) and [HTTP](http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol)-based applications by spreading requests across multiple servers. It is written in [C](http://en.wikipedia.org/wiki/C_(programming_language)) and has a reputation for being fast and efficient (in terms of processor and memory usage).  HAProxy is used by a number of high-profile websites including [GitHub](http://en.wikipedia.org/wiki/GitHub), [Stack Overflow](http://en.wikipedia.org/wiki/Stack_Overflow),[Reddit](http://en.wikipedia.org/wiki/Reddit), [Tumblr](http://en.wikipedia.org/wiki/Tumblr), and [Twitter](http://en.wikipedia.org/wiki/Twitter) and is used in the OpsWorks product from [Amazon Web Services](http://en.wikipedia.org/wiki/Amazon_Web_Services). |
| Javascript | **JavaScript** is a[dynamic](http://en.wikipedia.org/wiki/Dynamic_programming_language) computer [programming language](http://en.wikipedia.org/wiki/Programming_language).It is most commonly used as part of [web browsers](http://en.wikipedia.org/wiki/Web_browser), whose implementations allow[client-side scripts](http://en.wikipedia.org/wiki/Client-side_scripting) to [interact with the user](http://en.wikipedia.org/wiki/User_interface), control the browser, communicate[asynchronously](http://en.wikipedia.org/wiki/Ajax_(programming)), and alter the [document content](http://en.wikipedia.org/wiki/Document_Object_Model) that is displayed. It is also used in server-side network programming with frameworks such as [Node.js](http://en.wikipedia.org/wiki/Node.js), game development and the creation of desktop and mobile applications.  JavaScript is classified as a [prototype-based](http://en.wikipedia.org/wiki/Prototype-based_programming)[scripting language](http://en.wikipedia.org/wiki/Scripting_language) with [dynamic](http://en.wikipedia.org/wiki/Dynamic_language) typing and[first-class functions](http://en.wikipedia.org/wiki/First-class_functions). This mix of features makes it a [multi-paradigm](http://en.wikipedia.org/wiki/Multi-paradigm) language, supporting [object-oriented](http://en.wikipedia.org/wiki/Object-oriented_programming), [imperative](http://en.wikipedia.org/wiki/Imperative_programming), and[functional](http://en.wikipedia.org/wiki/Functional_programming) programming styles.  Despite some naming, [syntactic](http://en.wikipedia.org/wiki/Syntax_(programming_languages)), and[standard library](http://en.wikipedia.org/wiki/Standard_library) similarities, JavaScript and[Java](http://en.wikipedia.org/wiki/Java_(programming_language)) are otherwise unrelated and have very different [semantics](http://en.wikipedia.org/wiki/Semantics_(computer_science)). The syntax of JavaScript is actually derived from [C](http://en.wikipedia.org/wiki/C_(programming_language)), while the semantics and design are influenced by [Self](http://en.wikipedia.org/wiki/Self_(programming_language))and [Scheme](http://en.wikipedia.org/wiki/Scheme_(programming_language)) programming languages.  JavaScript is also used in environments that aren't web-based, such as [PDF](http://en.wikipedia.org/wiki/Portable_Document_Format) documents, [site-specific browsers](http://en.wikipedia.org/wiki/Site-specific_browser), and [desktop widgets](http://en.wikipedia.org/wiki/Desktop_widget). Newer and faster JavaScript [virtual machines](http://en.wikipedia.org/wiki/Virtual_machine)(VMs) and platforms built upon them have also increased the popularity of JavaScript for server-side [web applications](http://en.wikipedia.org/wiki/Web_application). On the client side, JavaScript has been traditionally implemented as an [interpreted](http://en.wikipedia.org/wiki/Interpreter_(computing))language, but more recent browsers perform [just-in-time compilation](http://en.wikipedia.org/wiki/Just-in-time_compilation).  JavaScript has been standardized in the [ECMAScript](http://en.wikipedia.org/wiki/ECMAScript) language specification. |
| JSON | **JSON**  or **JavaScript Object Notation**, is an [open standard](http://en.wikipedia.org/wiki/Open_standard) format that uses [human-readable](http://en.wikipedia.org/wiki/Human-readable) text to transmit data objects consisting of [attribute–value pairs](http://en.wikipedia.org/wiki/Attribute%E2%80%93value_pair). It is used primarily to transmit data between a server and web application, as an alternative to [XML](http://en.wikipedia.org/wiki/XML).  Although originally derived from the[JavaScript](http://en.wikipedia.org/wiki/JavaScript) scripting language, JSON is a[language-independent](http://en.wikipedia.org/wiki/Language-independent_specification) data format. Code for[parsing](http://en.wikipedia.org/wiki/Parsing) and generating JSON data is readily available in many [programming languages](http://en.wikipedia.org/wiki/Programming_languages).  The JSON format was originally specified by[Douglas Crockford](http://en.wikipedia.org/wiki/Douglas_Crockford). It is currently described by two competing standards, [RFC 7159](http://tools.ietf.org/html/rfc7159)and ECMA-404. The ECMA standard is minimal, describing only the allowed grammar syntax, whereas the RFC also provides some semantic and security considerations. The official [Internet media type](http://en.wikipedia.org/wiki/Internet_media_type) for JSON isapplication/json. The JSON filename extension is .json. |
| Lightswitch | Microsoft Visual Studio LightSwitch is an extension and framework specifically tailored for creating line-of-business applications built on existing .NET technologies and Microsoft platforms. The applications produced are architecturally [3-tier](http://en.wikipedia.org/wiki/Multitier_architecture): the user interface runs on either [Microsoft Silverlight](http://en.wikipedia.org/wiki/Silverlight), [HTML 5 client](http://en.wikipedia.org/wiki/HTML5)[[157]](http://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-157) or as a SharePoint 2013 app;[[158]](http://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-158) the logic and data-access tier is built on [WCF Data Services](http://en.wikipedia.org/wiki/WCF_Data_Services) and exposed as an [OData](http://en.wikipedia.org/wiki/OData) feed hosted[[159]](http://en.wikipedia.org/wiki/Microsoft_Visual_Studio#cite_note-159) in ASP.NET; and the primary data storage supports [Microsoft SQL Server Express](http://en.wikipedia.org/wiki/SQL_Server_Express), [Microsoft SQL Server](http://en.wikipedia.org/wiki/Microsoft_SQL_Server) and [Microsoft SQL Azure](http://en.wikipedia.org/wiki/SQL_Azure). LightSwitch also supports other data sources including [Microsoft SharePoint](http://en.wikipedia.org/wiki/SharePoint), OData and WCF RIA Services. |
| Linux | **Linux** or, less frequently used, [/](http://en.wikipedia.org/wiki/Help:IPA_for_English)[ˈlaɪnəks](http://en.wikipedia.org/wiki/Help:IPA_for_English#Key)[/](http://en.wikipedia.org/wiki/Help:IPA_for_English) [***lyn****-uks*](http://en.wikipedia.org/wiki/Wikipedia:Pronunciation_respelling_key)) is a[Unix-like](http://en.wikipedia.org/wiki/Unix-like) and mostly [POSIX](http://en.wikipedia.org/wiki/POSIX)-compliantcomputer [operating system](http://en.wikipedia.org/wiki/Operating_system) assembled under the model of [free and open-source software](http://en.wikipedia.org/wiki/Free_and_open-source_software)development and distribution. |
| Load balancer | In [computing](http://en.wikipedia.org/wiki/Computing), **load balancing** distributes [workloads](http://en.wikipedia.org/wiki/Workload) across multiple computing resources, such as computers, a [computer cluster](http://en.wikipedia.org/wiki/Computer_cluster), [network](http://en.wikipedia.org/wiki/Computer_network) links, [central processing units](http://en.wikipedia.org/wiki/Central_processing_unit) or [disk drives](http://en.wikipedia.org/wiki/Disk_drives). Load balancing aims to optimize resource use, maximize [throughput](http://en.wikipedia.org/wiki/Throughput), minimize response time, and avoid overload of any single resource. Using multiple components with load balancing instead of a single component may increase reliability through [redundancy](http://en.wikipedia.org/wiki/Redundancy_(engineering)). Load balancing usually involves dedicated software or hardware, such as a [multilayer switch](http://en.wikipedia.org/wiki/Multilayer_switch#Layer_4_Load_Balancer) or a [Domain Name System](http://en.wikipedia.org/wiki/Domain_Name_System) server process. |
| MongoDB | **MongoDB** (from "hu**mongo**us") is a cross-platform [document-oriented database](http://en.wikipedia.org/wiki/Document-oriented_database). Classified as a [NoSQL](http://en.wikipedia.org/wiki/NoSQL) database, MongoDB eschews the traditional table-based [relational database](http://en.wikipedia.org/wiki/Relational_database) structure in favor of [JSON](http://en.wikipedia.org/wiki/JSON)-like documents with dynamic [schemas](http://en.wikipedia.org/wiki/Database_schema) (MongoDB calls the format [BSON](http://en.wikipedia.org/wiki/BSON)), making the integration of data in certain types of applications easier and faster. Released under a combination of the [GNU Affero General Public License](http://en.wikipedia.org/wiki/GNU_Affero_General_Public_License) and the [Apache License](http://en.wikipedia.org/wiki/Apache_License), MongoDB is [free and open-source software](http://en.wikipedia.org/wiki/Free_and_open_source_software). |
| MySQL | **MySQL**  "My S-Q-L",officially, but also called "My Sequel") is (as of March 2014) the world's second most[[a]](http://en.wikipedia.org/wiki/MySQL#cite_note-9) widely used open-source [relational database management system](http://en.wikipedia.org/wiki/Relational_database_management_system)(RDBMS). It is named after co-founder [Michael Widenius](http://en.wikipedia.org/wiki/Michael_Widenius)'s daughter, My. The [SQL](http://en.wikipedia.org/wiki/SQL) phrase stands for[Structured Query Language](http://en.wikipedia.org/wiki/Structured_Query_Language). |
| Node.JS | **Node.js** is an [open source](http://en.wikipedia.org/wiki/Open_source), [cross-platform](http://en.wikipedia.org/wiki/Cross-platform)[runtime environment](http://en.wikipedia.org/wiki/Runtime_system) for server-side and networking applications. Node.js applications are written in [JavaScript](http://en.wikipedia.org/wiki/JavaScript), and can be run within the Node.js runtime on [OS X](http://en.wikipedia.org/wiki/OS_X), [Microsoft Windows](http://en.wikipedia.org/wiki/Microsoft_Windows), [Linux](http://en.wikipedia.org/wiki/Linux) and [FreeBSD](http://en.wikipedia.org/wiki/FreeBSD).  Node.js provides an [event-driven architecture](http://en.wikipedia.org/wiki/Event-driven_architecture)and a non-blocking I/O API that optimizes an application's throughput and scalability. These technologies are commonly used for real-time applications.  Node.js uses the [Google V8](http://en.wikipedia.org/wiki/V8_(JavaScript_engine)) JavaScript engine to execute code, and a large percentage of the basic modules are written in JavaScript. Node.js contains a built-in library to allow applications to act as a Web server without software such as [Apache HTTP Server](http://en.wikipedia.org/wiki/Apache_HTTP_Server) or [IIS](http://en.wikipedia.org/wiki/Internet_Information_Services). |
| NoSQL | A **NoSQL** (often interpreted as **N**ot **O**nly **SQL** database provides a mechanism for [storage](http://en.wikipedia.org/wiki/Computer_data_storage) and [retrieval](http://en.wikipedia.org/wiki/Data_retrieval) of data that is modeled in means other than the tabular relations used in [relational databases](http://en.wikipedia.org/wiki/Relational_database). Motivations for this approach include simplicity of design, [horizontal scaling](http://en.wikipedia.org/wiki/Horizontal_scaling#Horizontal_and_vertical_scaling) and finer control over availability. The data structures used by NoSQL databases (e.g. key-value, graph, or document) differ from those used in relational databases, making some operations faster in NoSQL and some faster in relational databases. The particular suitability of a given NoSQL database depends on the problem it must solve. |
| NuGET | **NuGet** is a [free](http://en.wikipedia.org/wiki/Free_software) and [open source](http://en.wikipedia.org/wiki/Open_source_software) [package manager](http://en.wikipedia.org/wiki/Package_manager) for the Microsoft development platform (formerly known as NuPack). It supports both the [.NET Framework](http://en.wikipedia.org/wiki/.NET_Framework) and native packages written in [C++](http://en.wikipedia.org/wiki/C%2B%2B). NuGet is distributed as a [Visual Studio](http://en.wikipedia.org/wiki/Visual_Studio) extension. Starting with Visual Studio 2012, NuGet comes preinstalled by default. NuGet is also integrated with [SharpDevelop](http://en.wikipedia.org/wiki/SharpDevelop). NuGet can also be used from the command line and automated via scripts. |
| RabbitMQ | **RabbitMQ** is open source [message broker](http://en.wikipedia.org/wiki/Message_broker)software (sometimes called [message-oriented middleware](http://en.wikipedia.org/wiki/Message-oriented_middleware)) that implements the[Advanced Message Queuing Protocol](http://en.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol)(AMQP). The RabbitMQ server is written in the [Erlang programming language](http://en.wikipedia.org/wiki/Erlang_(programming_language)) and is built on the [Open Telecom Platform](http://en.wikipedia.org/wiki/Open_Telecom_Platform) framework for clustering and failover. Client libraries to interface with the broker are available for all major programming languages. |
| REST | **Representational state transfer** (**REST**) is an abstraction of the architecture of the [World Wide Web](http://en.wikipedia.org/wiki/World_Wide_Web); more precisely, REST is an architectural style consisting of a coordinated set of architectural constraints applied to components, connectors, and data elements, within a distributed [hypermedia](http://en.wikipedia.org/wiki/Hypermedia) system. REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements. |
| Shipyard | Built on the Docker cluster management toolkit [Citadel](https://github.com/citadel/citadel), Shipyard gives you the ability to manage Docker resources including containers, hosts and more.  Shipyard differs from other management applications in that it promotes composability. At the core, Shipyard only manages Docker (containers, etc). However, using "[Extension Images](http://shipyard-project.com/docs/extension_images/)" you can add functionality such as application routing and load balancing, centralized logging, deployment and more. You decide which components to use that fit your needs. |
| SQL Server | **Microsoft SQL Server** is a [relational database management system](http://en.wikipedia.org/wiki/Relational_database_management_system) developed by[Microsoft](http://en.wikipedia.org/wiki/Microsoft). As a database, it is a software product whose primary function is to store and retrieve data as requested by other software applications, be it those on the same computer or those running on another computer across a network (including the Internet). There are at least a dozen different editions of Microsoft SQL Server aimed at different audiences and for workloads ranging from small single-machine applications to large Internet-facing applications with many[concurrent users](http://en.wikipedia.org/wiki/Concurrent_user). Its primary [query languages](http://en.wikipedia.org/wiki/Query_language) are [T-SQL](http://en.wikipedia.org/wiki/Transact-SQL) and [ANSI SQL](http://en.wikipedia.org/wiki/SQL). |
| Ubuntu | **Ubuntu**  is a [Debian](http://en.wikipedia.org/wiki/Debian)-based [Linux](http://en.wikipedia.org/wiki/Linux)[operating system](http://en.wikipedia.org/wiki/Operating_system), with [Unity](http://en.wikipedia.org/wiki/Unity_(user_interface)) as its default [desktop environment](http://en.wikipedia.org/wiki/Desktop_environment). It is based on [free software](http://en.wikipedia.org/wiki/Free_software) and named after the Southern African philosophy of [*ubuntu*](http://en.wikipedia.org/wiki/Ubuntu_(philosophy)) (literally, "human-ness"), which often is translated as "humanity towards others" or "the belief in a universal bond of sharing that connects all humanity". |
| Web service | A **Web service** is a method of communication between two electronic devices over a[network](http://en.wikipedia.org/wiki/Computer_network). It is a software function provided at a network address over the Web with the service *always on* as in the concept of [utility computing](http://en.wikipedia.org/wiki/Utility_computing). |
| Webdeploy | Web Deploy (msdeploy) simplifies deployment of Web applications and Web sites to IIS servers. Administrators can use Web Deploy to synchronize IIS servers or to migrate to newer versions of IIS. Web Deploy Tool also enables administrators and delegated users to use IIS Manager to deploy ASP.NET and PHP applications to an IIS server. |