

**Edition V 1. 0**

**Published by:**

TAVSS team a division of Mansoura University CIS Faculty team

Copyright © 2019 by TAVSS Team

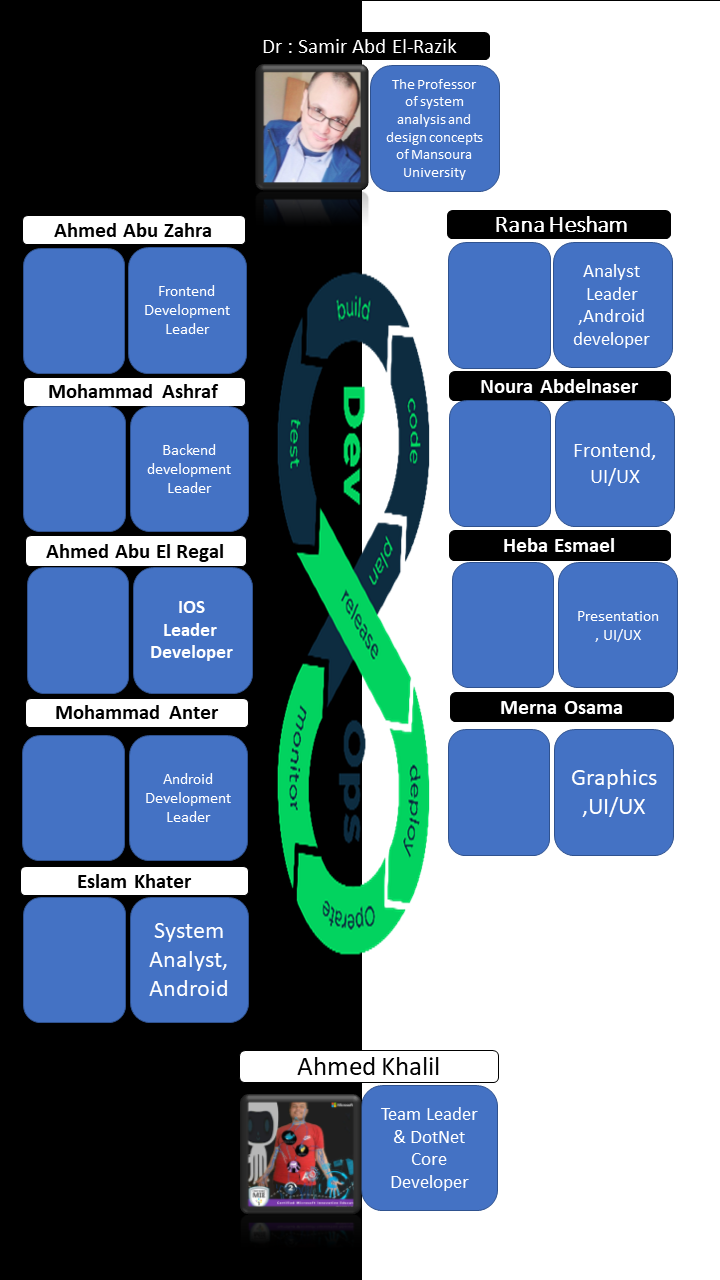


Table of Contents

[Introduction 4](#_Toc24768212)

[Defining Services 4](#_Toc24768213)

[Objectives 5](#_Toc24768214)

[SWOT Analysis: 6](#_Toc24768215)

[Internal Vs External Services 7](#_Toc24768216)

[Dimensions of Solution 9](#_Toc24768217)

[A traditional model to solve problem 10](#_Toc24768218)

[The Biggest deals of Architectures {Monolithic vs SOA VS MICROSERVICES} 11](#_Toc24768219)

[Monolith 11](#_Toc24768220)

[A service-oriented architecture (SOA): 11](#_Toc24768221)

[Microservice 12](#_Toc24768222)

[To Go Macro, Go Micro 12](#_Toc24768223)

[Pros and Cons of Architectures: 13](#_Toc24768224)

[The Plan of The Microservices Development 14](#_Toc24768225)

[How Microservices Improve Agile Development 14](#_Toc24768226)

[The plan decomposition 14](#_Toc24768227)

[The 4 Agile Sprints: 16](#_Toc24768228)

[Architecture Building 16](#_Toc24768229)

[Microservice Design Patterns 16](#_Toc24768230)

[Domain Driven Design 17](#_Toc24768231)

[Communication in a microservice architectur 19](#_Toc24768232)

[Asynchronous microservice integration enforces microservice’s autonomy 21](#_Toc24768233)

[What is Docker? 22](#_Toc24768234)

[Introduction to Docker 23](#_Toc24768235)

[why Docker 24](#_Toc24768236)

[Docker terminology 24](#_Toc24768237)

[Docker Orchestration 27](#_Toc24768238)

[Clustering 27](#_Toc24768239)

[Building Blocks 27](#_Toc24768240)

[Domain Driven Design Pattern 27](#_Toc24768241)

[Bounded Contexts 27](#_Toc24768242)

[Conceptual Design Model 27](#_Toc24768243)

[Architecture Building 27](#_Toc24768244)

[Summary 27](#_Toc24768245)

[Bibliography 27](#_Toc24768246)

# Chapter 1 : Introduction

## Defining Services

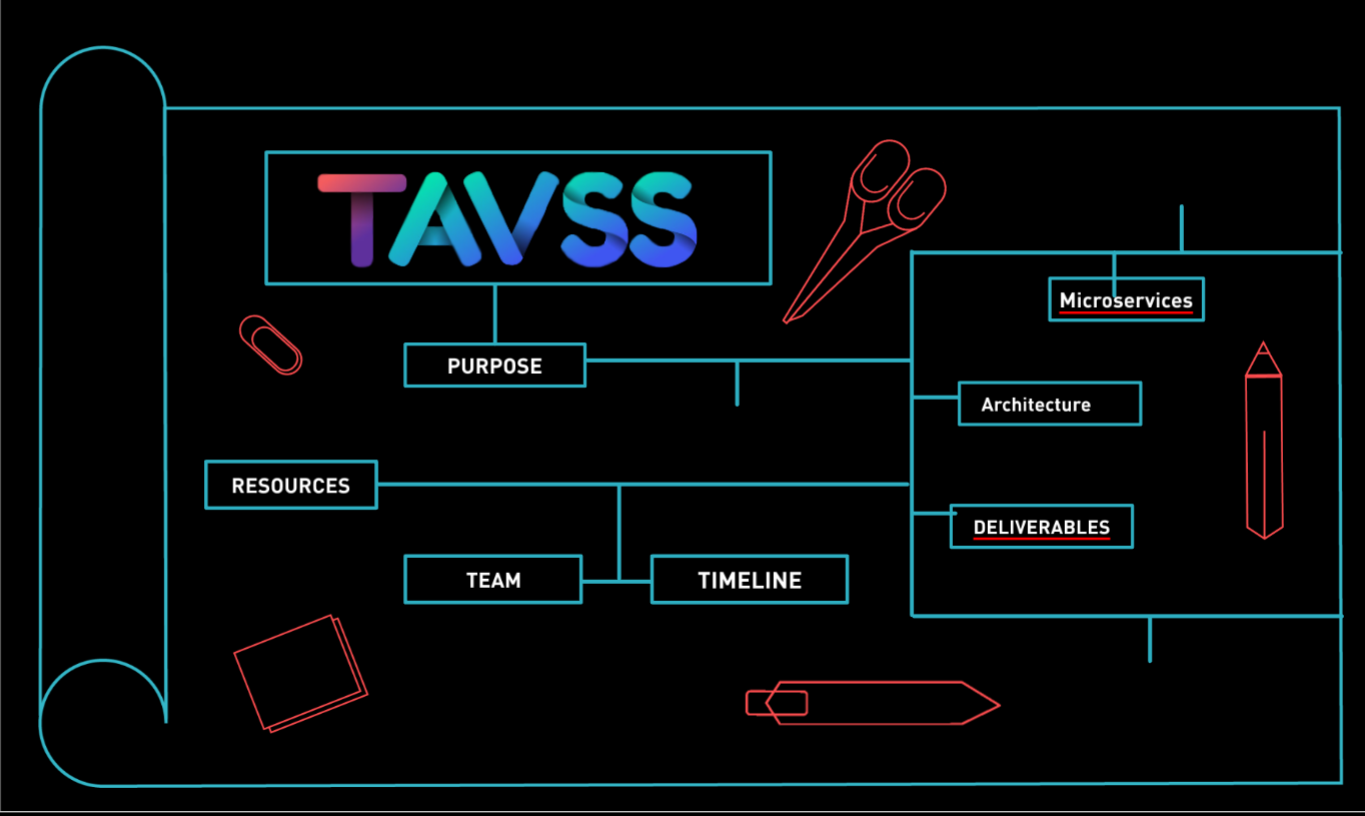
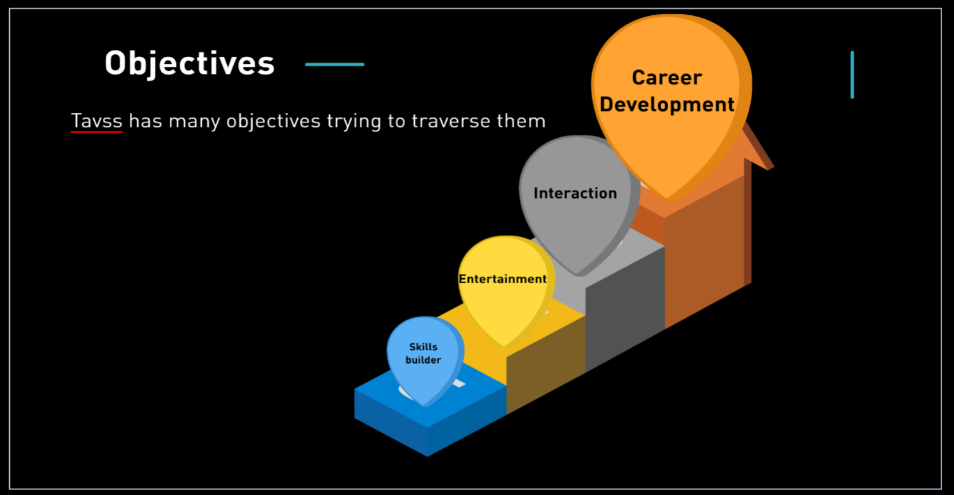
 **A**ftertouching the tip of the E-learning Iceberg and Career Development concepts but when you are on the tip you can discover all the iceberg and make a solution to go to the next. The learning, career development and skills building are the obstacles on our lifecycle in our faculty. Many students suffer so we build our hypothesis on four steps that we discovered. the lifecycle starts with the student in all faculty years.

Figure .1 the map of work a presentation source for Prezi website <https://prezi.com/p/h3u0u9zheol/tavss/>

the life cycle form of learning by entertainment to interaction learning to skills building to develop the career of the student by supporting the integration of many system services that we tell every student loses these services in faculty, dividing them to internal and external services. Internal services like project, course, schedule, tasks, games and assignment management; external services like communities, teams, companies, training centers, market places and integration by the business using new methods of interaction like evaluation, chatting, blogging and cv development using standards as illustrated as Figure 1.1.

## Objectives



**T**he best goal we need to achieve is by creating channels between student, TAs, doctors and companies by supporting the entire services of communication and career development techniques by 4 steps of process integration as shown as Figure 1,2.

2 Figure 1.2 The steps of students on tavss website

* **Learning by entertainment:** is an internal integration process by use gaming, challenges, project challenging, scheduling times, assignments.
* **Interaction learning:**  Internal-External integration process by use development communities of faculty, training centers, companies, training courses, project challenging, gaming.
* **Skills Building:** Internal-External integration process by use soft skills building and development skills with communities and Skills Training centers like UCCD and ITI and other training models, with the acclaims of every students.
* **Career Development:** Internal-External Integration, every student here have cv , every cv built by the  previous 3 steps and the doctors, companies, training centers achieve the standards of cv making and career development technique.

If we need to Identify the problem, we need to define The **SWOT**

A SWOT analysis is a technique used to determine and define your Strengths, Weaknesses, Opportunities, and Threats – SWOT.

## SWOT Analysis:

SWOT analyses can be applied to an entire company or organization, or individual projects within a single department. Most commonly, SWOT analyses are used at the organizational level to determine how closely a business is aligned with its growth trajectories and success benchmarks, but they can also be used to ascertain how well a particular project – such as [an online advertising campaign](https://www.wordstream.com/blog/ws/2017/07/05/online-advertising-costs) – is performing according to initial projections as explained in figure 1.3.



Figure 1.4 the Main Swot Analysis components for decision making

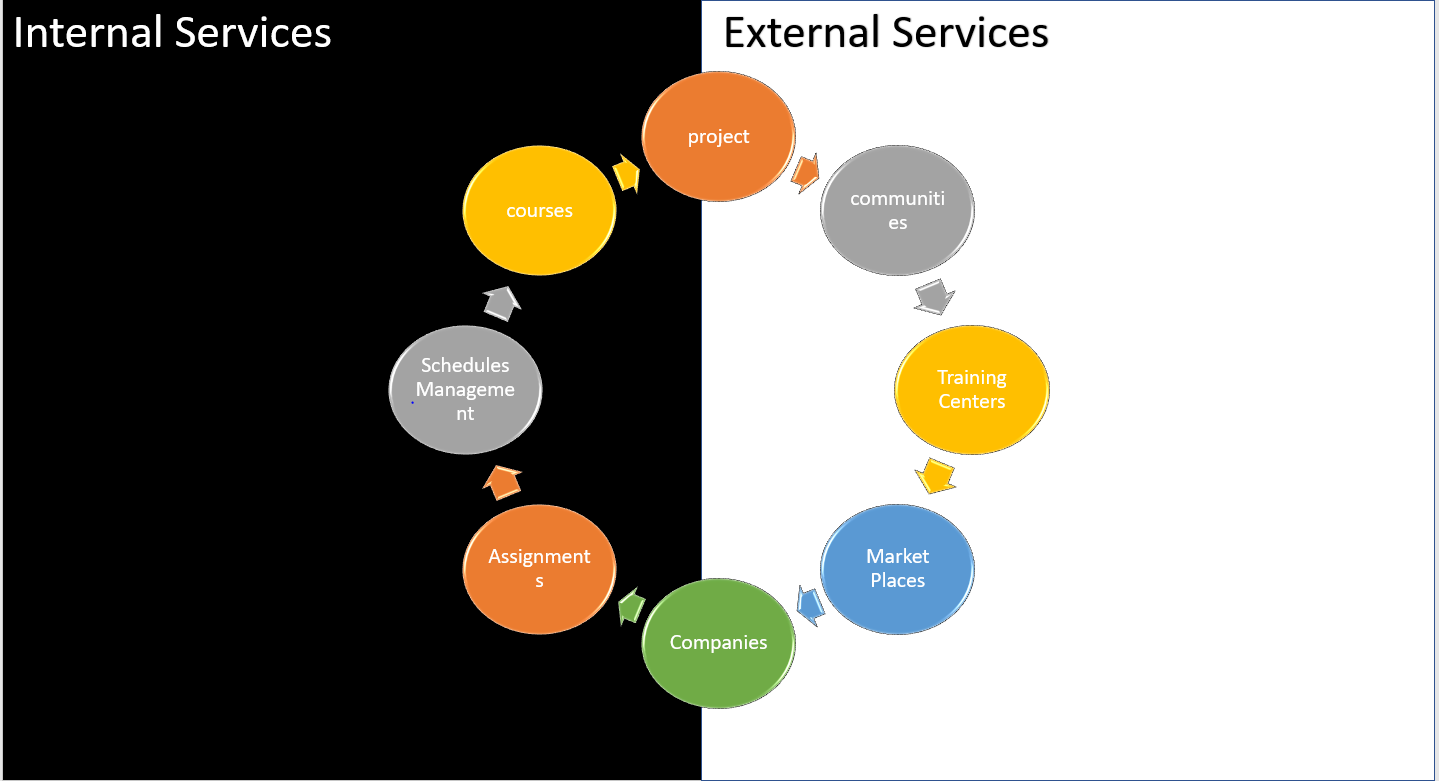
3 Figure 1.3 have a long term plan from SWOT Analysis

Figure 1.5 Swot Analysis of CIS Faculty

Resources:

<https://www.wordstream.com/blog/ws/2017/12/20/swot-analysis>

## Internal Vs External Services



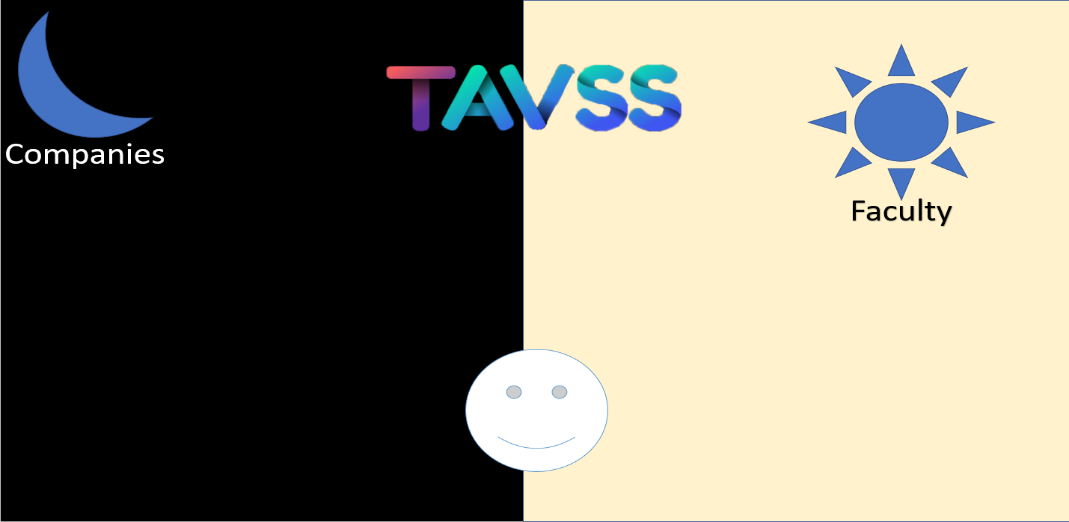
Figure 1.6 Internal vs External services

Figure 1.7 how to combine the east of faculty and the west of company

Solving problem of students needs great effort to build a solution, and the swot analysis get the right action to predict, how to combine the light with dark, the dawn and night, but we can try to define the best solution for them as a system to serve them, we can define some services some of them are internal and the other are external:

**project service:** it is a service to manage the projects in the educational organization it provides download and upload projects, enable project evaluation, project documentation and interaction between different student’s project, between doctors, student and TAs

**communities:** provide a high communication with external communities to improve the performance and contribute in the organization in different ways even financial or worker etc.….

**Training centers:** provide a good relationship and powerful interaction with training centers for making the workers provide a higher performance and improve their skills

**market places:** Provide a high powerful communication technique to provide a high marketing for the organization so it will attract as much as we can of users

**courses:** it is a service to manage the courses in the educational organization it provides Module systems, module assignment, virtual classes and topic discussion

**schedule management:** it is a service to manage the schedule in the educational organization it provides Lecture alarm to notify students, doctors and TAs about lecture times and lecture director to notify them about the place and the lecture specific time

**Assignments:** it is a service to manage the assignments in the educational organization it provides prizes when success in specific assignment, provides a question and short quizzes which is even open or close and enable evaluation techniques for the assignments

**companies:** it is a service to manage the interaction between the system and any company that will use the system even for free or companies will buy it

## Dimensions of Solution

Figure 1.8 The Dimensions of TAVSS

**We** need to redefine the new system to deal with the new Requirements with 3 dimensions:

**Business Layer:** To get the logic of building and combine multi-business architectures.

“The business logic layer is where you tackle the problems your program was created to solve. In the logic layer, classes decide what information they need in order to solve their assigned problems, request that information from the [accessor layer](https://www.sciencedirect.com/topics/computer-science/accessor-layer), manipulate that information as required, and return the ultimate results to the presentation layer for formatting.”[2]

**Architecture Pattern:** What is the best Pattern to recognize the requirements of multiple changing business?

refers to the fundamental structures of a [software system](https://en.wikipedia.org/wiki/Software_system) and the discipline of creating such structures and systems. Each structure comprises software elements, relations among them, and properties of both elements and relations. The *architecture* of a software system is a metaphor, analogous to the [architecture](https://en.wikipedia.org/wiki/Architecture) of a building.It functions as a blueprint for the system and the developing project, laying out the tasks not necessary to be executed by the design teams.[1]

**Student Requirements:** What is the best conceptual model can observe all these requirements?

Resources:

<https://www.sciencedirect.com/topics/computer-science/business-logic-layer>

## A traditional model to solve problem

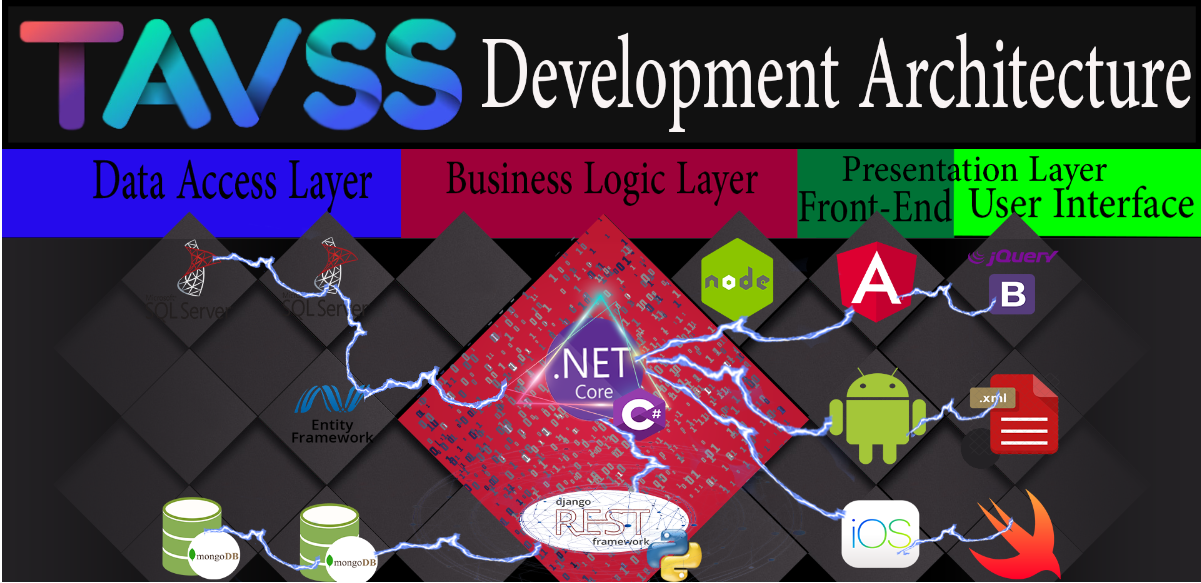


Figure 1.9 The Traditional Distributed System To solve the problem

This traditional Architecture that support student has a greet establishments, but it’s not sustainable for long term business, but this is the first architecture on our heads to support as illustrated in figure 1.7.

It’s not a Monolithic architecture, something more, something to higher up our faculty for business.

To Define this architecture, we have 2 phases a Modular one and a Structure one:

* **Modular Architecture:** The Applications are decomposed over the 3-tier architecture:
* **Data Access Layer+ Data Layer:** {SQL Server + NoSQL Mongo + some Middleware for integrations => Entity Framework}.
* **Business Logic Layer: {**Dotnet Core APIS+ djangorestframework API: to control the processing of transmission and the operations of business**}**
* **Presentation Layer: {**Web UIs+ Angular frontend, Android, IOS platforms for mobile user**}**
* **Structure:** Each app has its own design like:
* **Data Centric:** {MVC, MC} patterns
* **NoSQL vs SQL DB: {**Mongo vs SQL Server**}**
* **Http Restful Services: {**Get, Post, Put, Delete**}**
* **Asynchronous Messaging: {**RabbitMQ It will be explained in next Modules**,** SignalR: Chatting services for client server publish subscribe Design Pattern**}**

## The Biggest deals of Architectures {Monolithic vs SOA VS MICROSERVICES}

Let’s now compare the Architectures of Designing a system and decide to take the best alternative of architectures:

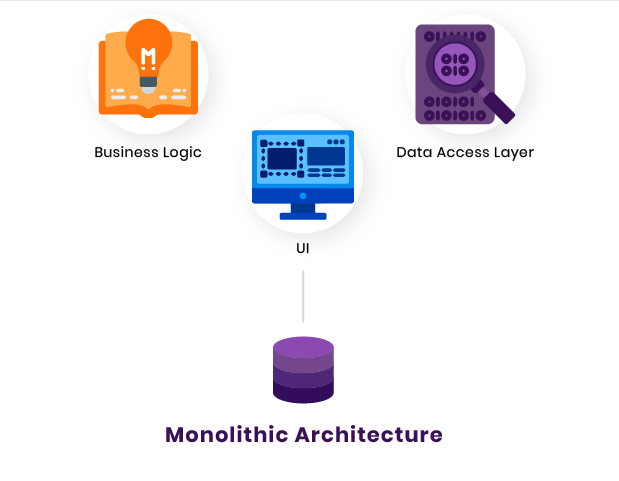
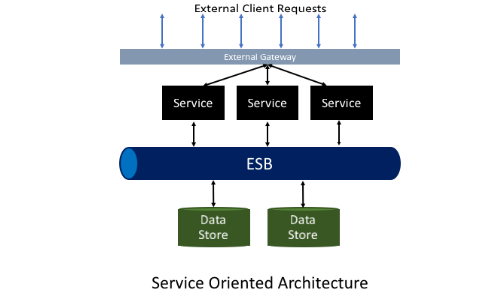
Monolith**:**is an ancient word referring to a huge single block of stone. Though this term is used broadly today, the image remains the same across fields. In software engineering, a monolithic pattern refers to a single indivisible unit. The concept of monolithic software lies in different components of an application being combined into a single program on a single platform. Usually, a monolithic app consists of a database, client-side user interface, and server-side application. All the software’s parts are unified, and all its functions are managed in one place. Let’s look at the structure of the monolithic software in detail.

Figure 1.10 The simple Architecture of Monolithic app

A service-oriented architecture (SOA): is a software architecture style that refers to an application composed of discrete and loosely coupled software agents that perform a required function. SOA has two main roles: a service provider and a service consumer. Both of these roles can be played by a software agent. The concept of SOA lies in the following: an application can be designed and built in a way that its modules are integrated seamlessly and can be easily reused.

Microservice: is a type of service-oriented software architecture that focuses on building a series of autonomous components that make up an app. Unlike monolithic apps built as a single indivisible unit, microservice apps consist of multiple independent components that are glued together with APIs.

Figure 1.11 Simple Architecture Of SOA

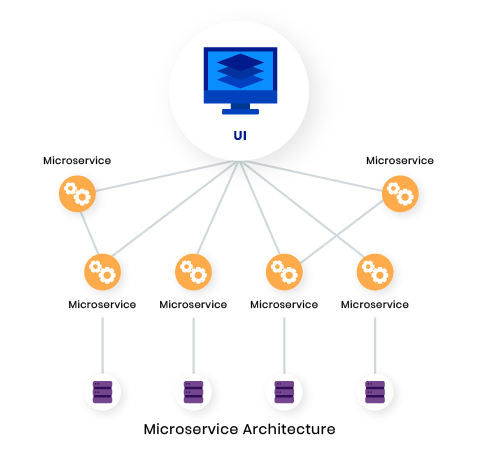
****

Figure 1.12 Simple Microservice Architecture

**Microservices** are important simply because they add unique value in a way of simplification of complexity in systems. By breaking apart your system or application into many smaller parts, you show ways of reducing duplication, increasing cohesion and lowering your coupling between parts, thus making your overall system parts easier to understand, more scalable, and easier to change.

As organizations scale, their monolithic systems become [progressively complex and more difficult to manage](https://dzone.com/articles/microservices-in-practice-1). [6]This results in problems such as:

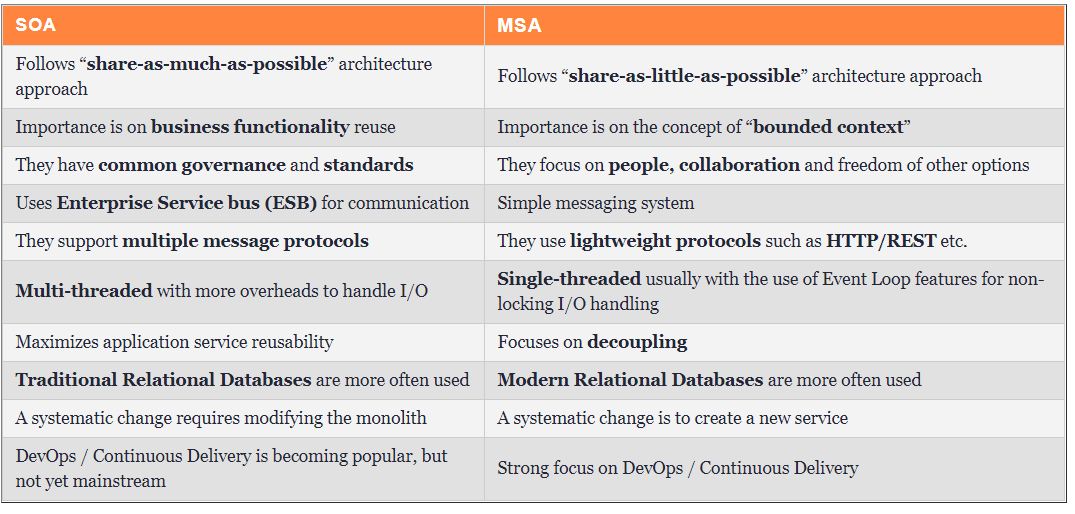
* **Lengthy downtime** **to undergo updates**: Since each service in a monolithic system is overly reliant with one another, updating one part of the system would require you to update the entire the system. Thus, requiring a significant amount of development time and resources.
* **High costs for expansions**: For scaling, monolithic systems usually require investment into more servers. Again, more cost is involved to acquire more servers.
* **Platform-wide Malfunctions**: This is perhaps the biggest pain point of a monolithic system. If one service within a monolithic system goes down, then this may initiate a chain reaction and could cause the entire system to go down.

### To Go Macro, Go Micro

To put it bluntly, there’s a reason why some of the biggest players in the software industry to dumping the monolith in favor of the microservice. The former presents significant pains when it comes to scalability, deployment, and agile development — while the latter presents the antidote to those pains.

Investing in microservices may be costlier up front, but with speedier development processes, faster scalability, and the ability to more easily adapt to constantly fluctuating IoT-driven market, that investment makes a whole lot of sense.  [6]

Table 1.1 from dzone research as a comparison between SOA and Microservices



### Pros and Cons of Architectures:

Table 1.2 The pros and cons of Many Distributed systems

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | | Monolithic | SOA | Microservices |
| * Pros * Cons | | * 1)simple development and deployment * 2)fewer cross-cutting concerns * 3)Better performance | | * 1)Reusability of services * 2) Better Maintainability * 3) Higher reliability | * 1)Easy to develop, test, and deploy * 2) Increase Agility * 3)Ability to scale horizontally |
| * 1)code base come cumbersome over time * 2)Limited agility * 3)Difficult to adopt new technologies | | * 1)Complex management * 2) Extra overload * 3) High investment costs | * 1)Complexity: * 2) Security Concerns * 3)Different programming languages |

So After this short definition for each architecture if we want to decide which one of these strategies we will choose to implement TAVSS we will choose  microservice strategy which will make the system consists of independent components so any update or any attack or problems happens with any service will not affect the other services ,will choose the strategy that will focus on business priorities to improve TAVSS as an educational business ,Microservices are important simply because they add unique value in a way of simplification of complexity in the system. By breaking apart our system into many smaller parts, show ways of reducing duplication, increasing cohesion and lowering your coupling between parts, thus making your overall system parts easier to understand, and easier to change. One of the most important reasons for choosing this strategy because it increases agility which agility is the most powerful development strategies and finally scaling our system will be much easier.

Resources:

<https://rubygarage.org/blog/monolith-soa-microservices-serverless>

<https://dzone.com/articles/microservices-vs-soa-whats-the-difference>

## The Plan of The Microservices Development

The best project management process for the microservices concept is the agile approach, Organizations as large as Amazon, Soundcloud and Netflix have already made successful transitions from monolithic architecture to microservices in recent years. But it’s not just the big guns; a Gartner [study](https://www.belatrixsf.com/blog/agile-development-is-ideally-suited-for-microservices/) from 2016 found that approximately 68 percent of organizations were either investigating or developing a microservice architecture.

### How Microservices Improve Agile Development

Agile Development, which some refer to as ‘DevOps’, means having a number of small teams working on individual, smaller projects so that those projects are able to get a team’s undivided attention. This enables the individual projects to be completed quicker. Microservice architecture fits into this development model perfectly, as each small team can own and focus on one service. A recent study showed that projects managed under agile development tend to be [28 percent more successful](https://blog.capterra.com/agile-project-management-statistics-for-2018/) than traditional project management techniques.

By assigning a small team of 6-8 people to work on each microservice using the DevOps model, developers can, in theory at least, work more effectively and efficiently. But in order for DevOps to successfully develop and maintain a microservice architecture, strong project management is essential. Since multiple teams will be working on different elements, there is a possibility that some conflicts between the different teams will arise. For example, one team will feel that a certain task is not associated with their assigned microservice and could come back to management and say, “well, that’s not our problem”.

With a strong culture of collaboration and project management, DevOps can enable developers to work more effectively than they otherwise would with a monolithic system. This is reflected in the aforementioned Lightbend survey, with by 30 percent of respondents claiming that microservice architecture helped them increase development velocity for new releases.[6]

### The plan decomposition



Figure 1.13 The decomposition mind map of project processes

### Defining The Processes

Explaining every plan execution and delivery at the next tables:

Table 1.2 The planning phase Decomposition

|  |  |
| --- | --- |
| The Planning Phase | |
| SWOT consulting | Defining organization building blocks of {Strengths, Weaknesses, Opportunities, threats}. |
| Ideation | Design Thinking process to solve problems and making Automation |
| Problem Identification | Or problem statement to get the specified problem of an organization |
| Business Model Canvas | To follow strategic plan without losing the resources and control all our business |
| Alternative Analysis | Decision Making to get the action of the architecture to make or stop |
| Milestones | To Finnish the process |

|  |  |
| --- | --- |
| Conceptual Design of the models {DDD pattern} | |
| Bounded contexts | Breaking down the business to support many layer architecture for business to Microservices. |
| Modular system | Defining new technologies to support business |
| Microservice Architecture | Defining the real Architecture and how to connect Microservices |
| Devops | automates the processes between software development and IT teams, in order that they can build, test, and release software faster and more reliably. |
| Agile scrum framework | Subset of Agile a framework that helps teamwork together much like a rugby team |
| Microservice | arranges an application as a collection of loosely coupled services |

|  |  |
| --- | --- |
| Sprint1 {1/10:1/11 2019} | |
| breaking a [complex topic](https://en.wikipedia.org/wiki/Complexity) or substance into smaller parts in order to gain a better understanding of it | **Analysis (user story/Req class diagrams)** |
| Design API for identity service as software that intermediary allows identity service to talk to other services. | **Identity. API (SQL server + file system)** |
| Design API for course service as software that intermediary allows course service to talk to other services. | **Course.API**  **(Mongo+FSDB)** |
| Design API for project service as software that intermediary allows project service to talk to other services. | **Project.API**  **(Mongo + FSDB)** |
| Using sql server and mongo DB as a technique to develop data access layer | **SQL server +Mongo DB migration with data access layer** |
| process of verifying the identity of a person or device. + specifying access rights/privileges to resources, which is related to information security and computer security in general and to access control in particular | **Authentication**  **+ Authorization** |
| Brings a design-centric approach to user interface and user experience and offers a practical skill-based instruction -centered around a visual communication perspective with angular | **UI/UX +Angular** |
| automates the processes between software development and IT teams, in order that they can build, test, and release software faster and more reliably by (CI/CD +Testing +HCI). | **) Devops CI/CD +Testing +HCI** |
| Make it easier to create deploy and run applications by using containers to allow developer to backup an application with all parts it need | **Docker compose deployment** |
| allows describing the structure of APIs so that machines can read them | **{Swaggers + writing} documenting service** |

|  |  |
| --- | --- |
| Sprint3 {1/2:1/6 2020 } | |
| Increasing the project release and fix errors and problems | **Versioning up+ maintenance** |
| Releasing the last versions of all APIs | **Finishing up all APIs** |
| Making test to make sure if all services will satisfy its functions when integrate them together | **Integration test** |
| Test if the entire system will satisfy it's functions and work well | **System testing** |
| Releasing the last versions of the frontend | **Finishing up frontend** |
| Releasing the last versions of Android mobile application | **Android finishing** |
| Releasing the last versions of IOS mobile application | **IOS finishing** |
| Starting automates the processes between software development and IT teams, in order that they can build, test, and release software faster and more reliably | **Devops starting operations** |
| Writing the documentation for the system to document every step, tool and function in the system | **Documenting system** |
| Releasing TAVSS to the users to use it | **Finishing the project** |

|  |  |
| --- | --- |
| Sprint 2{1/11:1/2 2020} | |
| Start developing android and IOS mobile applications | **Android +IOS starting** |
| Design API for community’s service as software that intermediary allows course service to talk to other services | **Communities. APIs** |
| Design API for Acclaims service as software that intermediary allows course service to talk to other services | **Acclaims. APIs** |
| enables calling methods on connected clients from the server | **SinalR HUB** |
| open source message brokers. From [T-Mobile](https://www.youtube.com/watch?v=1qcTu2QUtrU) to [Runtastic](https://medium.com/@runtastic/messagebus-handling-dead-letters-in-rabbitmq-using-a-dead-letter-exchange-f070699b952b), RabbitMQ , is lightweight and easy to deploy on premises and in the cloud | **Rabbit MQ service Bus** |
| (CQRS)every [method](https://en.wikipedia.org/wiki/Method_(computer_science)) should either be a *command* that performs an action, or a *query* that returns data to the caller, but not both .+ (Ocelot BFF)Ocelot is an Open Source .NET Core based API Gateway especially made for microservices architecture that need unified points of entry into their system | **CQRS+Ocelot BFF{Mediators+**  **Aggregators}** |
| Deploy Make it easier to create deploy and run applications by using containers to allow developer to backup an application with all parts it need | **Docker Deployment** |
| automates the processes between software development and IT teams, in order that they can build, test, and release software faster and more reliably by (CI/CD +Testing +HCI) . | **Devops(CI/CD+**  **Testing+HCI)** |
| Writing the documentation for the system to document every step ,tool and function in the system | **Documenting** |
| brings a design-centric approach to user interface and user experience design, and offers practical, skill-based instruction centered around a visual communications perspective with angular | **UI/UX Angular** |

### Scrum as Agile framework

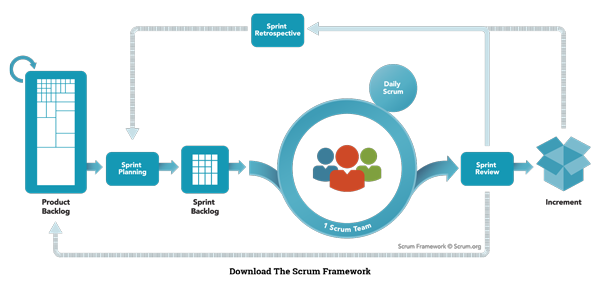


Figure 1.14 The Scrum Product framework

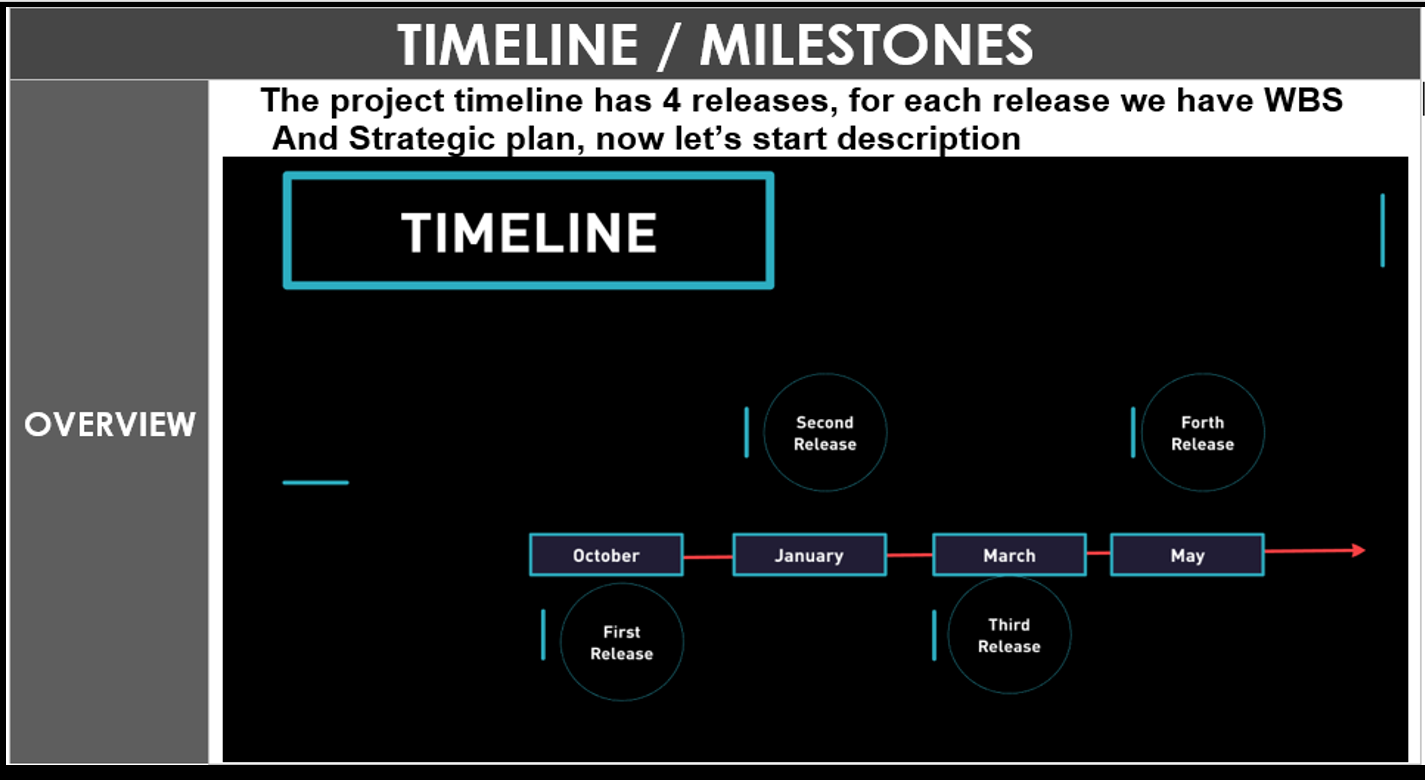
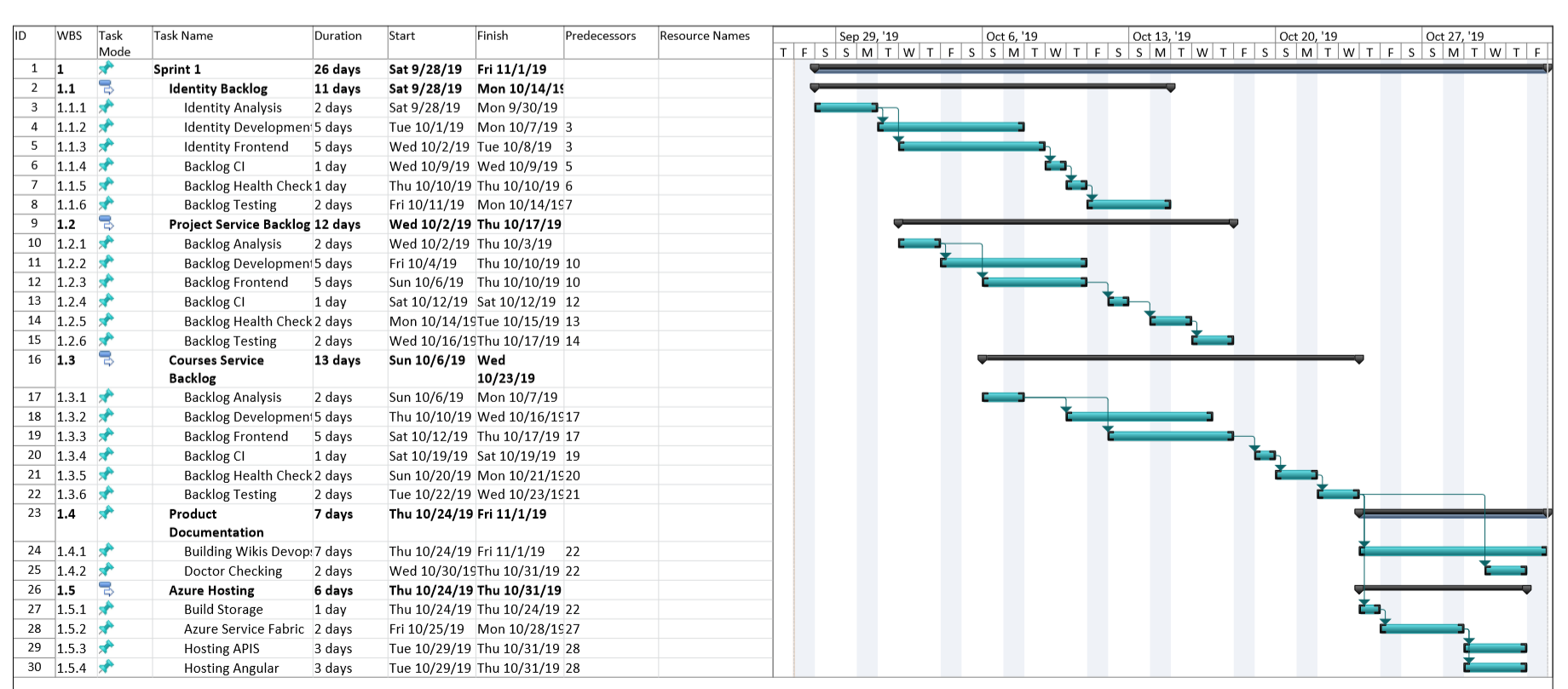


Figure 1.15 The Releasing timeline



### The 4 Agile Sprints:

# Architecture Building

## Microservice Design Patterns

Because of the complexity and the large scale of Microservices architecture we need to define how can solve the complexity and using many patterns that can help the project to be more valuable, so if there a need to get more accurate architecture, must follow greet patterns:

6 Figure 2.1 The Design Pattern image

### Domain Driven Design

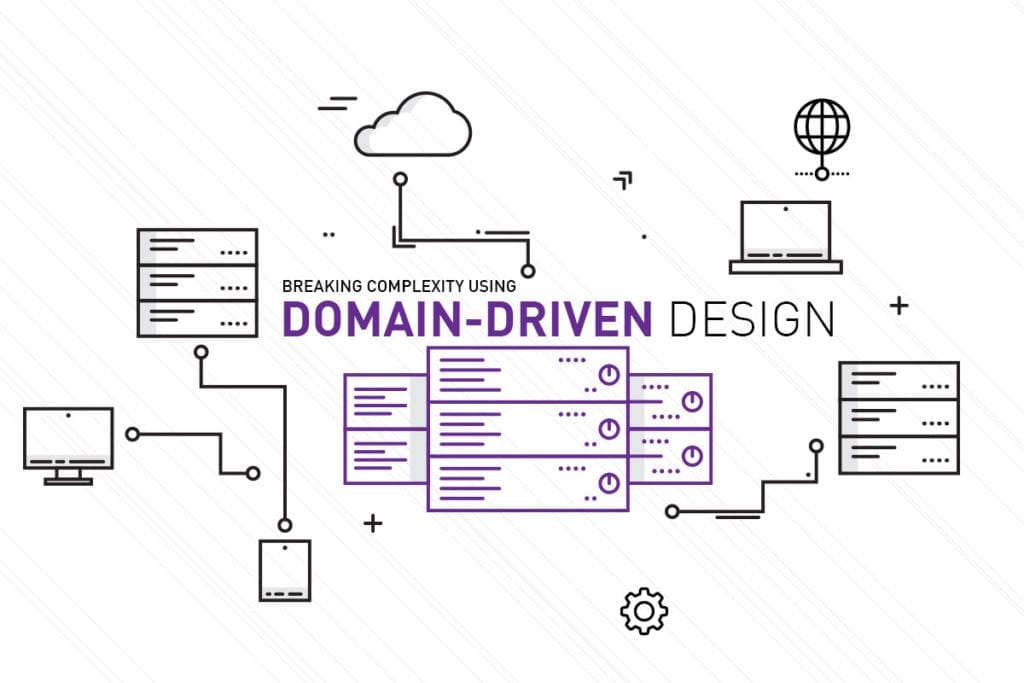
Domain-Driven Design is a set of tools or patterns that are used in designing and implementing the software that delivers high value, both strategically and tactically. By using DDD strategic development tools, you and your team will be able to create the competitively best software design choices and integration decisions for your business.

Figure 2.2 The Domain Driven Design

In order to design a useful software, which accurately models the unique operations of the business, you can use DDD tactical development tools to assist you and your team.  
DDD is the most effective software design and implementation needed to succeed in today’s competitive business landscape. Complicated? Not Necessarily. However, it does involve a set of advanced techniques to be used on complex software projects. Domain-Driven Design will bring domain experts and software developers together to create a software solution that reflects the mental model of business experts. It is essential to have the same understanding of the business and provide a solution to help business processes.

Therefore, the DDD method will need to invest in resources.   
Within our team, we have domain experts using the DDD. This way, the team will not only deliver the application but the true business value application. [2]

Domain-driven design (DDD), a software development technique first proposed by [Eric Evans](http://domainlanguage.com/ddd/), includes strategic, philosophical, tactical, and technical elements and is related to many specific practices. I've written about [why you need DDD (even though you think you don't)](http://techbeacon.com/why-you-need-domain-driven-design).

#### DDD strategy and philosophy

You probably know that DDD has strategic value; that’s why so many companies with extremely complex domains rely on it to produce software that can rapidly evolve with the business. But did you know that DDD also has a philosophical theme? You might have heard the term “ubiquitous language,” which is a mouthful when you’re speaking about it, but it’s a shorthand way of emphasizing the fundamental principle of DDD: Use domain terminology everywhere; make it ubiquitous. When practicing DDD, this basic philosophy of the primacy of domain terminology can be distributed across three guiding principles:

1. Capture the domain model, in domain terms, through interactions with domain experts.

In other words, talk to the people in the businesses where you are solving problems and understand them from their point of view first and foremost. This is how you form the [ubiquitous language](http://martinfowler.com/bliki/UbiquitousLanguage.html) of the domain and set the foundation for harmonious models.

1. Embed the domain terminology in the code.

This means naming things the way the domain expert would name them, including classes, methods, commands, and especially domain events. This is how you reflect the domain model in the code.

1. Protect the domain knowledge from corruption by other domains, technical subdomains, etc.

If you find that your code is talking about two different things—e.g., the domain solution and the technical implementation—separate those components to keep the subdomains apart. This strategy tends to result in classes with single responsibilities and a terse, focused vocabulary. Put "translators" at the [boundaries](http://techbeacon.com/domain-driven-design-boundaries-leaking) between subdomains to keep them from depending on each other’s structures unnecessarily, and also to prevent blurring of the meaning of domain terms.

These three principles guide and inform DDD. Knowing and using them provides benefits, even without the rest of the DDD practices and patterns. Let’s look at how we might reap some benefits from just using this information in our software development projects.

### Communication in a microservice architectur

In a monolithic application running on a single process, components invoke one another using language-level method or function calls. These can be strongly coupled if you’re creating objects with code (for example, new ClassName()), or can be invoked in a decoupled way if you’re using Dependency Injection by referencing abstractions rather than concrete object instances. Either way, the objects are running within the same process. The biggest challenge when changing from a monolithic application to a microservices-based application lies in changing the communication mechanism. A direct conversion from in-process method calls into RPC calls to services will cause a chatty and not efficient communication that won’t perform well in distributed environments. The challenges of designing distributed system properly are well enough known that there’s even a canon known as the Fallacies of distributed computing that lists assumptions that developers often make when moving from monolithic to distributed designs.

There isn’t one solution, but several. One solution involves isolating the business microservices as much as possible. You then use asynchronous communication between the internal microservices and replace fine-grained communication that’s typical in intra-process communication between objects with coarser-grained communication. You can do this by grouping calls, and by returning data that aggregates the results of multiple internal calls, to the client.

**A microservices-based application** is a distributed system running on multiple processes or services, usually even across multiple servers or hosts. Each service instance is typically a process. Therefore, services must interact using an inter-process communication protocol such as **HTTP, AMQP**, or a binary protocol like TCP, depending on the nature of each service.

The microservice community promotes the philosophy of “smart endpoints and dumb pipes” This slogan encourages a design that’s as decoupled as possible between microservices, and as cohesive as possible within a single microservice. As explained earlier, each microservice owns its own data and its own domain logic. But the microservices composing an end-to-end application are usually simply choreographed by using REST communications rather than complex protocols such as WS-\* and flexible event-driven communications instead of centralized business-process-orchestrators.

The two commonly used protocols are HTTP request/response with resource APIs (when querying most of all), and lightweight asynchronous messaging when communicating updates across multiple microservices.

#### Communication types

Client and services can communicate through many different types of communication, each one targeting a different scenario and goals. Initially, those types of communications can be classified in two axes.

##### The first axis defines if the protocol is synchronous or asynchronous:

* + **Synchronous protocol.** HTTP is a synchronous protocol. The client sends a request and waits for a response from the service. That’s independent of the client code execution that could be synchronous (thread is blocked) or asynchronous (thread isn’t blocked, and the response will reach a callback eventually). The important point here is that the protocol (HTTP/HTTPS) is synchronous and the client code can only continue its task when it receives the HTTP server response. [4]

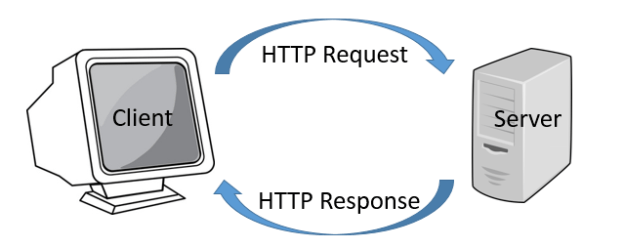


Figure 2.5 The Request Response synchronous transmission

* + **Asynchronous protocol**. Other protocols like AMQP (a protocol supported by many operating systems and cloud environments) use asynchronous messages. The client code or message sender usually doesn’t wait for a response. It just sends the message as when sending a message to a RabbitMQ queue or any other message broker. [4]

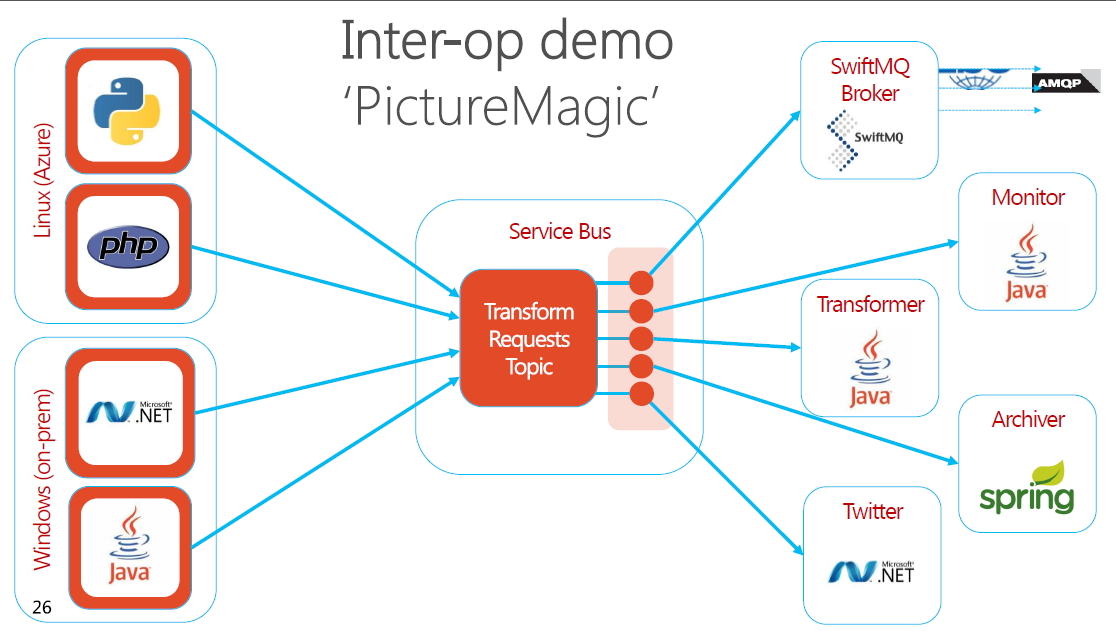


Figure 2.6 The Magic about Asynchronous transmission is to develop multi-programming language application

##### The second axis defines if the communication has a single receiver or multiple receivers:

* + **Single receiver**. Each request must be processed by exactly one receiver or service. An example of this communication is the Command pattern.
  + **Multiple receivers**. Each request can be processed by zero to multiple receivers. This type of communication must be asynchronous. An example is the publish/subscribe mechanism used in patterns like Event-driven architecture. This is based on an event-bus interface or message broker when propagating data updates between multiple microservices through events; it’s usually implemented through a service bus or similar artifact like Azure Service Bus by using topics and subscriptions.

A microservice-based application will often use a combination of these communication styles. The most common type is single-receiver communication with a synchronous protocol like HTTP/HTTPS when invoking a regular Web API HTTP service. Microservices also typically use messaging protocols for asynchronous communication between microservices.

These axes are good to know so you have clarity on the possible communication mechanisms, but they’re not the important concerns when building microservices. Neither the asynchronous nature of client thread execution nor the asynchronous nature of the selected protocol is the important points when integrating microservices.

What is important is being able to integrate your microservices asynchronously while maintaining the independence of microservices.

### Asynchronous microservice integration enforces microservice’s autonomy

As mentioned, the important point when building a microservices-based application is the way you integrate your microservices. Ideally, you should try to minimize the communication between the internal microservices. The fewer communications between microservices, the better. But in many cases, you’ll have to somehow integrate the microservices. When you need to do that, the critical rule here is that the communication between the microservices should be asynchronous. That doesn’t mean that you must use a specific protocol (for example, asynchronous messaging versus synchronous HTTP). It just means that the communication between microservices should be done only by propagating data asynchronously but try not to depend on other internal microservices as part of the initial service’s HTTP request/response operation. [4]

If possible, never depend on synchronous communication (request/response) between multiple microservices, not even for queries. The goal of each microservice is to be autonomous and available to the client consumer, even if the other services that are part of the end-to-end application are down or unhealthy. If you think you need to make a call from one microservice to other microservices (like performing an HTTP request for a data query) to be able to provide a response to a client application, you have an architecture that won’t be resilient when some microservices fail as illustrated In figure 2.7.

Moreover, having HTTP dependencies between microservices, like when creating long request/response cycles with HTTP request chains, as shown in the first part of the Figure 2.7, not only makes your microservices not autonomous but also their performance is impacted as soon as one of the services in that chain isn’t performing well.

The more you add synchronous dependencies between microservices, such as query requests, the worse the overall response time gets for the client apps.

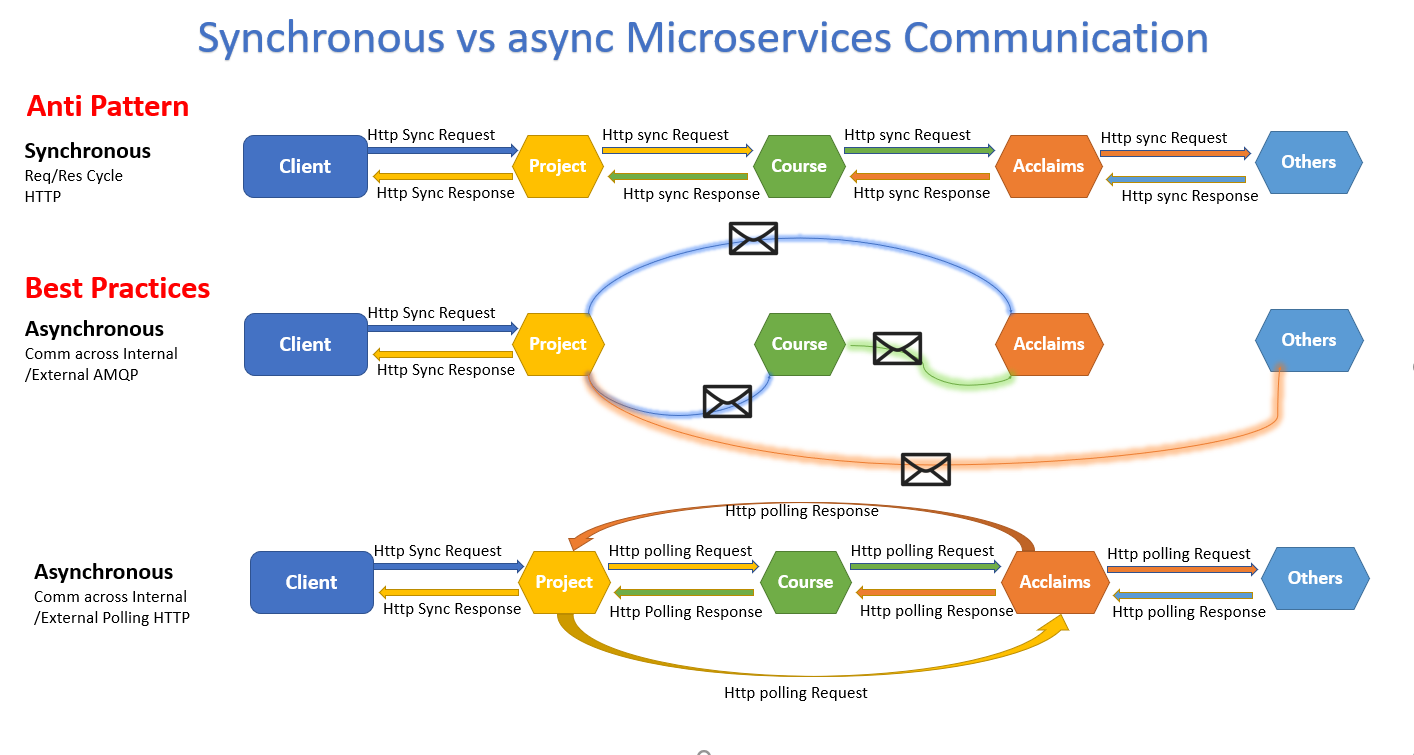


Figure 2.7 The 3 ways of Microservices communication

If your microservice needs to raise an additional action in another microservice, if possible, do not perform that action synchronously and as part of the original microservice request and reply operation. Instead, do it asynchronously (using asynchronous messaging or integration events, queues, etc.). But, as much as possible, do not invoke the action synchronously as part of the original synchronous request and reply operation.

And finally (and this is where most of the issues arise when building microservices), if your initial microservice needs data that’s originally owned by other microservices, do not rely on making synchronous requests for that data. Instead, replicate or propagate that data (only the attributes you need) into the initial service’s database by using eventual consistency (typically by using integration events, as explained in upcoming sections).

As noted earlier in the section Identifying domain-model boundaries for each microservice, duplicating some data across several microservices isn’t an incorrect design—on the contrary, when doing that you can translate the data into the specific language or terms of that additional domain or Bounded Context. For instance, in the Tavss application you have a microservice named identity.api that’s in charge of most of the user’s data with an entity named User. However, when you need to store data about the user within the Project microservice, you store it as a different entity named Acclaim. The Acclaim entity shares the same identity with the original User entity, but it might have only the few attributes needed by the Acclaim domain, and not the whole user profile.

You might use any protocol to communicate and propagate data asynchronously across microservices in order to have eventual consistency. As mentioned, you could use integration events using an event bus or message broker or you could even use HTTP by polling the other services instead. It doesn’t matter. The important rule is to not create synchronous dependencies between your microservices.

The following sections explain the multiple communication styles you can consider using in a microservice-based application.

## What is Docker?

### Introduction to Docker

Docker is an open-source project for automating the deployment of applications as portable, selfsufficient containers that can run on the cloud or on-premises. Docker is also a company that promotes and evolves this technology, working in collaboration with cloud, Linux, and Windows vendors, including Microsoft.[3]

Figure 2.3 Docker Logo

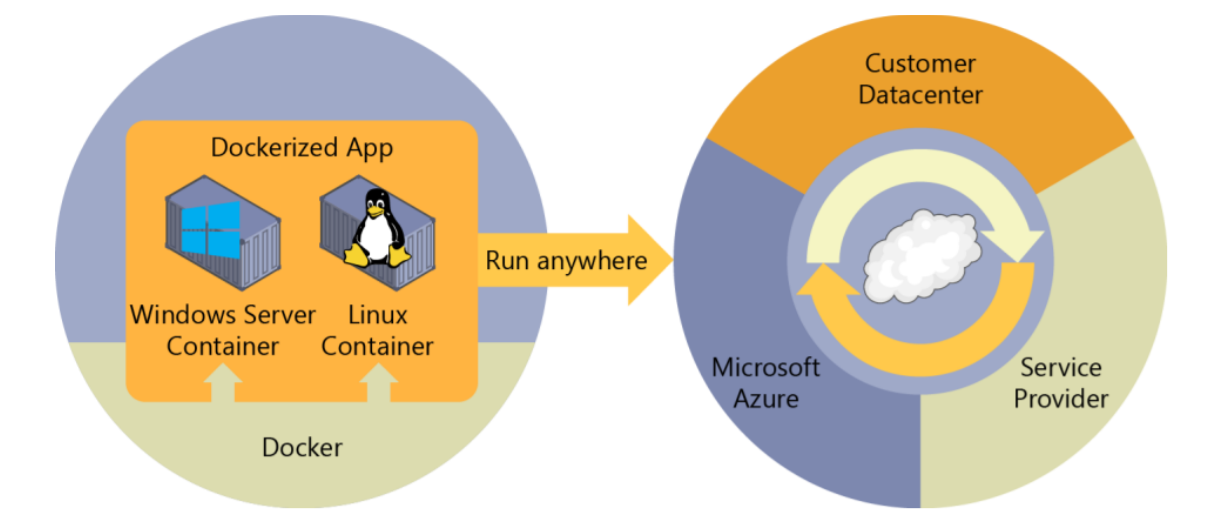
Docker image containers can run natively on Linux and Windows. However, Windows images can run only on Windows hosts and Linux images can run on Linux hosts and Windows hosts (using a Hyper-V Linux VM, so far), where host means a server or a VM. Developers can use development environments on Windows, Linux, or macOS as shown as figure 2.4

Figure 2.4. Docker deploys containers at all layers of the hybrid cloud

On the development computer, the developer runs a Docker host where Docker images are deployed, including the app and its dependencies. Developers who work on Linux or on the Mac, use a Docker host that’s Linux based, and they can only create images for Linux containers. (Developers working on the Mac can edit code or run the Docker command-line interface (CLI) from macOS, but as of this writing, containers don’t run directly on macOS.)

Developers who work on Windows can create images for either Linux or Windows Containers. To host containers in development environments and provide additional developer tools, Docker ships Docker Community Edition (CE) for Windows or for macOS.

These products install the necessary VM (the Docker host) to host the containers. [4]

Docker also makes available Docker Enterprise Edition (EE), which is designed for enterprise development and is used by IT teams who build, ship, and run large business-critical applications in production. To run Windows Containers, there are two types of runtimes:

* + **Windows Server Containers** provide application isolation through process and namespace isolation technology. A Windows Server Container shares a kernel with the container host and with all containers running on the host. 3 Introduction to containers and Docker.
  + **Hyper-V Containers** expand on the isolation provided by Windows Server Containers by running each container in a highly optimized virtual machine. In this configuration, the kernel of the container host isn’t shared with the Hyper-V Containers, providing better isolation. The images for these containers are created and work just the same way. The difference is in how the container is created from the image—running a Hyper-V Container requires an extra parameter. For details, see Hyper-V Containers.

### why Docker

### Docker terminology

This section lists terms and definitions you should be familiar with before getting deeper into Docker. For further definitions, see the extensive glossary provided by Docker.

* **Container image**: A package with all the dependencies and information needed to create a container. An image includes all the dependencies (such as frameworks) plus deployment and execution configuration to be used by a container runtime. Usually, an image derives from multiple base images that are layers stacked on top of each other to form the container’s filesystem. An image is immutable once it has been created.
* **Dockerfile**: A text file that contains instructions for how to build a Docker image. It’s like a batch script, the first line states the base image to begin with and then follow the instructions to install required programs, copy files and so on, until you get the working environment you need.
* **Build**: The action of building a container image based on the information and context provided by its Dockerfile, plus additional files in the folder where the image is built. You can build images with the Docker docker build command.
* **Container**: An instance of a Docker image. A container represents the execution of a single application, process, or service. It consists of the contents of a Docker image, an execution environment, and a standard set of instructions. When scaling a service, you create multiple instances of a container from the same image. Or a batch job can create multiple containers from the same image, passing different parameters to each instance.
* **Volumes**: Offer a writable filesystem that the container can use. Since images are read-only but most programs need to write to the filesystem, volumes add a writable layer, on top of the container image, so the programs have access to a writable filesystem. The program doesn’t know it is accessing a layered filesystem, it is just the filesystem as usual. Volumes live in the host system and are managed by Docker.
* **Tag**: A mark or label you can apply to images so that different images or versions of the same image (depending on the version number or the target environment) can be identified.
* **Multi-stage Build**: Is a feature, since Docker 17.05 or higher, that helps to reduce the size of the final images. In a few sentences, with multi-stage build you can use, for example, a large base image, containing the SDK, for compiling and publishing the application and then using the publishing folder with a small runtime-only base image, to produce a much smaller final image
* **Repository** (repo): A collection of related Docker images, labeled with a tag that indicates the image version. Some repos contain multiple variants of a specific image, such as an image containing SDKs (heavier), an image containing only runtimes (lighter), etc. Those variants can be marked with tags. A single repo can contain platform variants, such as a Linux image and a Windows image.
* **Registry:** A service that provides access to repositories. The default registry for most public images is Docker Hub (owned by Docker as an organization). A registry usually contains repositories from multiple teams. Companies often have private registries to store and manage images they’ve created. Azure Container Registry is another example.
* **Multi-arch image**: For multi-architecture, is a feature that simplifies the selection of the appropriate image, according to the platform where Docker is running, e.g. when a Dockerfile requests a base image FROM mcr.microsoft.com/dotnet/core/sdk:2.2 from the registry it actually gets 2.2-sdknanoserver-1709, 2.2-sdk-nanoserver-1803, 2.2-sdk-nanoserver-1809 or 2.2-sdk-stretch, depending on the operating system and version where Docker is running.
* **Docker Hub:** A public registry to upload images and work with them. Docker Hub provides Docker image hosting, public or private registries, build triggers and web hooks, and integration with GitHub and Bitbucket.
* **Azure Container Registry:** A public resource for working with Docker images and its components in Azure. This provides a registry that is close to your deployments in Azure and that gives you control over access, making it possible to use your Azure Active Directory groups and permissions.
* **Docker Trusted Registry (DTR):** A Docker registry service (from Docker) that can be installed onpremises so it lives within the organization’s datacenter and network. It is convenient for private images that should be managed within the enterprise. Docker Trusted Registry is included as part of the Docker Datacenter product. For more information, see Docker Trusted Registry (DTR).
* **Docker Community Edition (CE):** Development tools for Windows and macOS for building, running, and testing containers locally. Docker CE for Windows provides development environments for both Linux and Windows Containers. The Linux Docker host on Windows is based on a Hyper-V virtual machine. The host for Windows Containers is directly based on Windows. Docker CE for Mac is based on the Apple Hypervisor framework and the xhyve hypervisor, which provides a Linux Docker host virtual machine on Mac OS X. Docker CE for Windows and for Mac replaces Docker Toolbox, which was based on Oracle VirtualBox.
* **Docker Enterprise Edition (EE):** An enterprise-scale version of Docker tools for Linux and Windows development.
* **Compose:** A command-line tool and YAML file format with metadata for defining and running multi container applications. You define a single application based on multiple images with one or more .yml files that can override values depending on the environment. After you have created the definitions, you can deploy the whole multi-container application with a single command (dockercompose up) that creates a container per image on the Docker host.
* **Cluster:** A collection of Docker hosts exposed as if it were a single virtual Docker host, so that the application can scale to multiple instances of the services spread across multiple hosts within the cluster. Docker clusters can be created with Kubernetes, Azure Service Fabric, Docker Swarm and Mesosphere DC/OS.
* **Orchestrator**: A tool that simplifies management of clusters and Docker hosts. Orchestrators enable you to manage their images, containers, and hosts through a command line interface (CLI) or a graphical UI. You can manage container networking, configurations, load balancing, service discovery, high availability, Docker host configuration, and more. An orchestrator is responsible for running, distributing, scaling, and healing workloads across a collection of nodes. Typically, orchestrator products are the same products that provide cluster infrastructure, like Kubernetes and Azure Service Fabric, among other offerings in the market

### Docker Orchestration

### Clustering

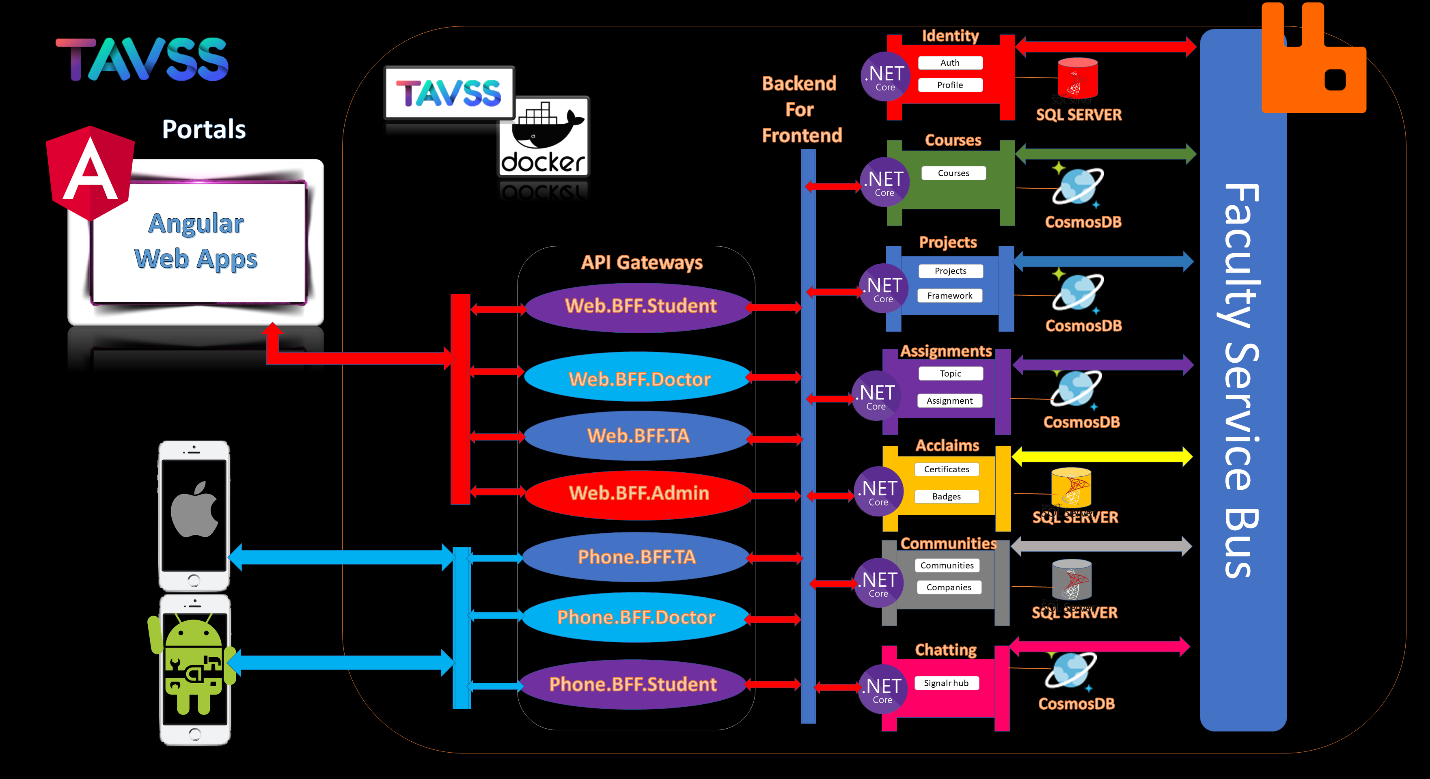
## Building Blocks

## Domain Driven Design Pattern

## Bounded Contexts

## Conceptual Design Model

## Architecture Building



## Summary

# Bibliography

1. David Wall, in [Multi-Tier Application Programming with PHP](https://www.sciencedirect.com/book/9780127323503), 2004
2. Prins Carl Santoso, Analyst at Mitrais <https://www.mitrais.com/news-updates/breaking-complexity-using-domain-driven-design/>
3. Cesar de la Torre, Sr. PM, .NET product team, Microsoft Corp, Containerized Application Lifecycle with Microsoft platform and tools 2019.
4. Cesar de la Torre, Bill Wagner, Mike Rousos .Net Microservice Architecture using Containerized Application
5. Dan Shewan, How to Do a SWOT Analysis for Your Small Business (with Examples) , <https://www.wordstream.com/blog/ws/2017/12/20/swot-analysis>
6. Jason Smit Chief User Experience Office, assessing the microservice development using agile approach 2018, <https://dotcms.com/blog/post/what-are-microservices-and-how-do-they-aid-agile-development->
7. Jeff Sutherland, Ken Schwaber, the co-creator of Scrum and a leading expert on how the framework has evolved to meet the needs of today’s business, <https://www.scrum.org/resources/what-is-scrum>