

آموزش میکروسرویس

فهرست

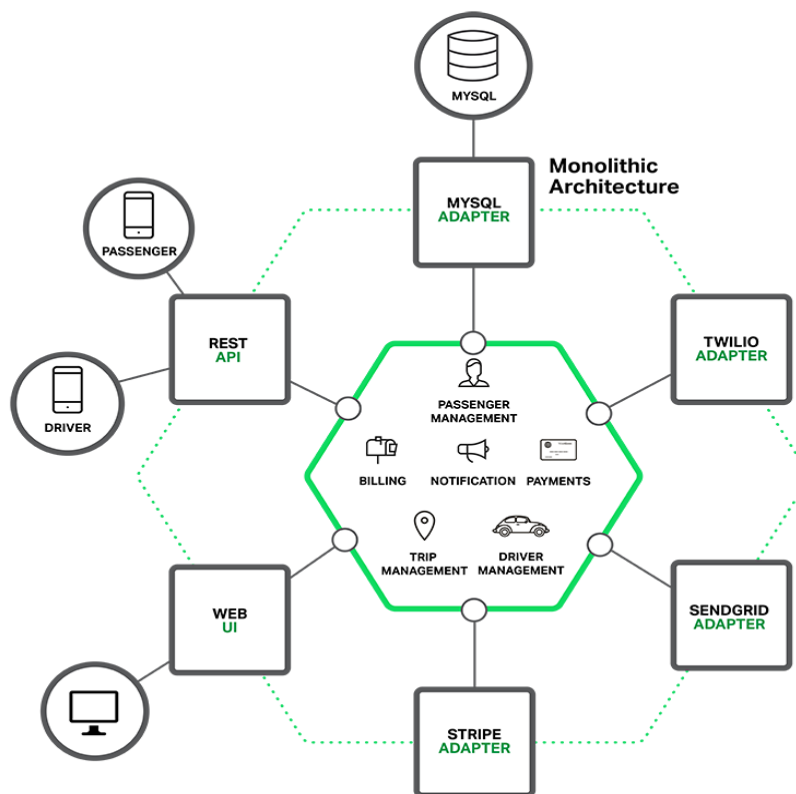
۲.....	<u>قسمت اول</u>
۴.....	<u>قسمت دوم</u>
۵.....	<u>قسمت سوم</u>
۶.....	<u>قسمت چهارم</u>
۸.....	<u>قسمت پنجم</u>
۹.....	<u>قسمت ششم</u>

قسمت اول:

معماری میکروسرویس:

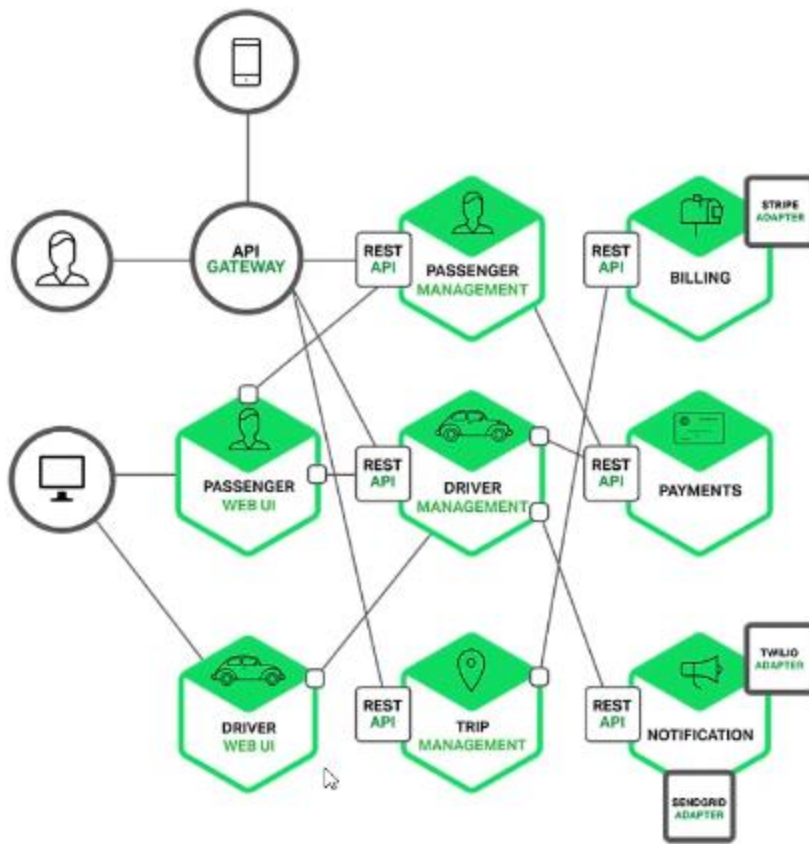
معماری میکروسرویس به شما اجازه می‌دهد اپلیکیشن‌های خود را و یا پروژه خود را به اپلیکیشن‌های کوچک (نه داده‌های کوچکتر) و سرویس‌های کوچکتر تقسیم کنید و بتوانید از آنها استفاده کنید. این کار بسیار مزایایی دارد. این امر باعث می‌شود که اپلیکیشن‌های شما مستقل کار کنند و می‌توانند روی سرورهای مستقل و متفاوت کار بکنند.

الگوی معماری میکروسرویس مزایای قابل توجهی دارد، به ویژه هنگامی که با توانایی توسعه سریع و تحویل برنامه‌های سازمانی پیچیده همراه می‌شود.



شکل ۱-۱ ساختار میکروسرویس

دستگاه‌ها و موبایل‌ها با GATEWAY در ارتباط هستند و سایر جزئیات سرویس‌های میکروسرویس هستند که می‌توانند پیام‌ها، احراز هویت و سایر موارد را بین خود تقسیم کنند.



شکل ۲ - ۱ ساختار میکروسرویس

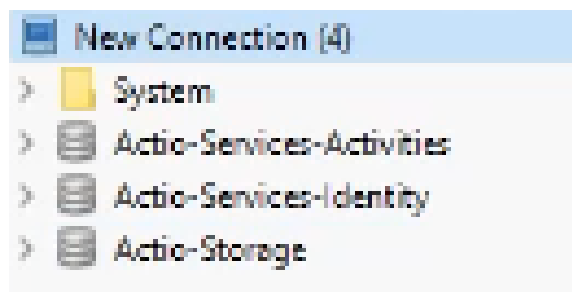
در شکل بالا ، هر کدام از مکعب ها یک سرویس می باشند و زمانی که دستگاه یا کاربر درخواستی را ارسال میکنند از GATEWAY عبور میکنند و API این درخواست را به سرویس های مربوطه به صورت PUSH کردن ارسال می کند و پس از کارکرد سرویس ها، GATEWAY دوباره API را به کاربر ارسال میکند.

پیشنیاز های میکروسرویس:

در این دوره GATEWAY ها با ASP.NET CORE پیاده سازی می شود و پیشنیاز این دوره C# ، ASP.NET CORE و WEB API می باشد.

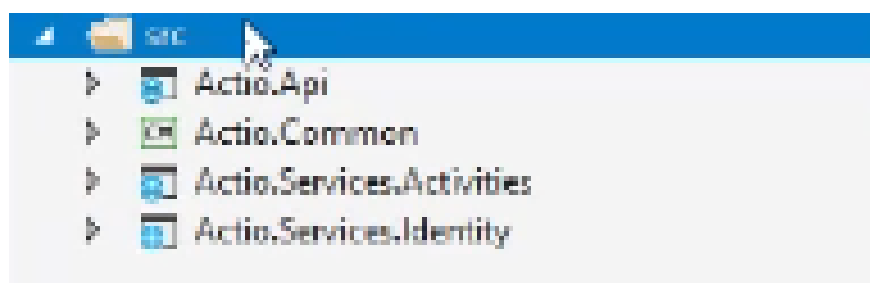
هر سرویس یک بانک اطلاعاتی نیاز دارد و بانک اطلاعاتی مورد استفاده در این دوره Mongo DB و No Sql می باشد. سرویس باس (Service Bus) این دوره که وظیفه آن کنترل کردن GATEWAY ها است، Rabbit MQ می باشد.

در این دوره از داکر (Docker) استفاده می شود و تمام سرویس ها و GATEWAY ها از داکر استفاده شده است.



شکل ۳-۱ بانک اطلاعاتی

در بانک اطلاعاتی سرویس های Activities, Identity, Storage را شامل می شود و در Solution پروژه Common لایه اصلی پروژه است و Gateway در API قرار گرفته، Activity عملیاتی است که در سیستم برنامه نویسی پیاده سازی می شود و سرویس Identity مسئولیت احراز هویت را به عهده دارد.



شکل ۴-۱ solution پروژه

برای سیستم احراز هویت از سیستم JWT (JSON Web TOKEN) استفاده می شود. کاربران از این سیستم احراز هویت می شوند و TOKEN ارسال می شود و تحویل لایه API داده می شود. این سرویس ها به طور مستقل در کنار هم کار میکنند تا بتوانند اپلیکیشن را کنترل بکنند. برای استفاده و تست سرویس ها از Postman استفاده می کنیم.

قسمت دوم:

اگر یک پروژه ساده را بخواهیم در نظر بگیریم که اپلیکیشن موبایل داشته باشد که شامل لایه

Data -> API -> Infrastructure -> Core -> presentation (View)

می باشد که این لایه ها در قالب یک پروژه عمل می کنند . تمامی فایل های DLL پروژه در لایه آخر Presentation قرار می گیرند و شروع به کار می کنند. هر چه اپلیکیشن با ترافیک بیشتر باشد، سرعت پروژه کمتر می شود و این لایه بندی در بهینه سازی و سرعت بهتر در کدنویسی به ما کمک می کند.

معماری میکروسرویس میگوید بجای یک اپلیکیشن، برنامه را به اپلیکیشن های کوچک تبدیل کرد که به صورت مستقل و با منابع مستقل اپلیکیشن ها کار کنند.

بطور مثال لایه های

- Data Service
- API Service -> Gateway
- Identity Service
- Activity Services

زمانی که کاربر درخواست سرویس API ارسال میکند، سرویس API دو پیغام به سرویس به Identity و Activity ها می دهد. این لایه ها در اپلیکیشن های متفاوت قرار دارند و حتی می توانند در سرور های متفاوت قرار بگیرند.

این به ما کمک می کند که سیستم ما یک سیستم توزیع شده باشد و نه یک پارچه. وقتی سیستم توزیع شده باشد ، منابع هم توزیع شده می باشند و می توان درخواست های زیاد را کنترل کرد. مثلا API یک درخواست به لایه Identity می دهد که آیا کاربر احراز هویت شده است و یا امکان استفاده از آن را دارد و Identity به API پاسخ را میدهد و ضمن تایید API، Activity مورد نظر فراخوان می شود.

به طور مثال یک وب سایت فروشگاهی فایل دانلودی، بیشترین بار برای دانلود ها می باشد. برای این پروژه، یک لایه Download Service داریم که وظیفه این لایه دانلود فایل ها می باشد. روند کار به این صورت است که API به Identity پیغام میدهد و تایید می شود و اجازه دانلود را به کاربر می دهد و به لایه Download می رود. اگر بار دانلودی زیاد شد، میتوان در پروژه دوتا لایه Download Service موازی با هم قرار داد که در صورت بار ترافیک زیاد، از لایه اول به لایه دوم فرستاده می شود. این دو لایه می توانند از دو سرور متفاوت نیز باشند. به طور مثال کسی که IP ایرانی داشت از سرویس یک و کسی که آلمان بود از سرویس دوم دانلود کند.

قسمت سوم:

پروژه ما باید یک دروازه ورودی داشته باشد، زمانی که درخواست (Request) به سمت برنامه ما ارسال می شود، یک دروازه وجود دارد که آن سرویس های API پروژه ما است (میتوان از آن API HTTP نیز یاد کرد).

بعد از آنکه در این دروازه وارد شد، به سرویس باس (Service bus) تحویل داده می شود که این سرویس باس ها می توانند Identity باشند و سپس به سراغ Activity Service می رود

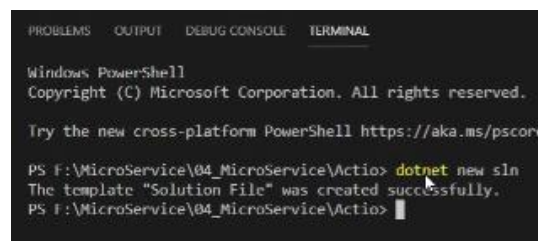
Request -> HTTP API [gateway] -> Service Bus -> Identity Service -> Activity Service
Solution پروژه ما شامل ۶ سرویس می باشد که شامل ۴ سرویس اصلی می باشد:

۱. Actio.API -> پل ورودی یا gateway: در این دوره از MongoDB [CQRS] استفاده می شود.
۲. Actio.Common -> وظیفه آن پیغام رسانی Message ها است: پیام هایی که بین سرویس ها تبادل می شود.
۳. Actio.Services.Identity -> MongoDB کار احراز هویت را انجام می دهد.
۴. Actio.Services.Activities -> Activities که نیاز داریم برای کار، بر روی SQL کار بکنند و یا روی MongoDB باشد که بتواند سرعت اطلاعات را برای بیت دیتا افزایش دهد.
- هر نوع سرویس فوق که ذکر شده می توانند به سرویس های بیشتر تکثیر شوند و معماری ما سرویس گرا می باشد.

۵. Actio.Tests -> تست های پروژه را در آن انجام دهیم.
۶. Actio.Tests.EndToEnd -> پایان سرویس های ما و تست های ما است که بتوان در این لایه تست ها را نوشت تا معماری ما تکمیل باشد.

قسمت چهارم:

یکی از امکاناتی که در این دوره می خواهیم با آن کار کنیم، محیط کدنویسی VsCode می باشد. در این برنامه می خواهیم پروژه خود را بسازیم. از طریق ترمینال دستور `dotnet new sln` را وارد میکنیم.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS F:\MicroService\04_MicroService\Actio> dotnet new sln
The template "Solution File" was created successfully.
PS F:\MicroService\04_MicroService\Actio> |
```

شکل ۱-۴ دستور ساخت پروژه

با دستور `mkdir src` یک پوشه برای پروژه های ما می سازیم.

```
PS F:\MicroService\04_MicroService\Actio> mkdir src
```

شکل ۲-۴ دستور ساخت پوشه

همچنین پوشه ای برای اسکریپت های پروژه با نام scripts و tests برای تست های پروژه می سازیم. حال میخواهیم در پوشه SRC پروژه های خود را بسازیم و با دستور `cd src` وارد پوشه SRC می شویم. اولین پروژه در آن web api می باشد که با دستور `dotnet new webapi -n Actio.Api` آن را میسازیم. عبارت `-n` برای این است که ما نام پروژه را نامگذاری کنیم و Actio.Api نام پروژه Api ما می باشد.

```
PS F:\MicroService\04_MicroService\Actio\src> dotnet new webapi -n Actio.Api
```

شکل ۳-۴ دستور ساخت پروژه Api

همچنین پروژه دیگر با نام Actio.Services.Identity و Actio.Services.Activities می سازیم. سپس می رویم به سراغ پروژه Common که تفاوتش با قبلی ها این است که دیگر این پروژه WebApi نیست که دستور ساخت آن عبارت است از: `dotnet new classlib -n Actio.Common`

```
PS F:\MicroService\04_MicroService\Actio\src> dotnet new classlib -n Actio.Common
```

شکل ۴-۴ ساخت پروژه Common

اکنون در SRC خود ۴ پروژه داریم. کاری که باید بکنیم این است که پروژه های در پوشه SRC به solution متصل شوند. در آدرس پروژه از با دستور `cd..` از پوشه SRC خارج می شویم و با زدن `/s` دایرکتوری ها و پوشه های در دسترس به ما نمایش داده می شود.

Mode	LastWriteTime	Length	Name
d----	8/15/2019 12:11 PM		scripts
d----	8/15/2019 12:14 PM		src
d----	8/15/2019 12:11 PM		tests
-a----	8/15/2019 12:11 PM	540	Actio.sln

شکل ۵-۴ دایرکتوری های پروژه

عبارت Actio.sln ، Solution ، پروژه ما است. حال باید ۴ پروژه خود را یکی یکی با دستور `dotnet sln add src/...` انجام داد.

```
PS F:\MicroService\04_MicroService\Actio> dotnet sln add src/Actio.Api/Actio.Api.csproj
Project 'src\Actio.Api\Actio.Api.csproj' added to the solution.
```

شکل ۶-۴ افزودن پروژه ها به Solution

- `dotnet sln add src/Actio.Api/Actio.Api.csproj`
- `dotnet sln add src/Actio.Common/Actio.Common.csproj`
- `dotnet sln add src/Actio.Services.Identity/Actio.Services.Identity.csproj`
- `dotnet sln add src/Actio.Services.Activities/Actio.Services.Activities.csproj`

حال می بایست پروژه خود را `restore` کرد تا پکیج های پروژه دانلود شوند و دستور `dotnet restore` را وارد میکنیم.

```
PS F:\MicroService\04_MicroService\Actio> dotnet restore
Restore completed in 17.08 ms for F:\MicroService\04_MicroService\Actio\src\Actio.Api\Actio.Api.csproj.
Restore completed in 17.08 ms for F:\MicroService\04_MicroService\Actio\src\Actio.Services.Activities\Actio.Services.Activities.csproj.
Restore completed in 17.08 ms for F:\MicroService\04_MicroService\Actio\src\Actio.Services.Identity\Actio.Services.Identity.csproj.
Restore completed in 22.46 ms for F:\MicroService\04_MicroService\Actio\src\Actio.Common\Actio.Common.csproj.
```

شکل ۷-۴ `dotnet restore`

سپس باید پروژه خود را `build` کرد تا اگر ارور و مشکلی داشته باشد رفع شود و پروژه آماده شود و دستور `dotnet build` را وارد میکنیم. اگر پیغام `build succeeded` مشاهده شد نشان دهنده درست بودن `build` ما می باشد.

```
Actio.Common -> F:\MicroService\04_MicroService\Actio\src\Actio.Common\bin\Debug\netstandard2.0\Actio.Common.dll
Actio.Services.Activities -> F:\MicroService\04_MicroService\Actio\src\Actio.Services.Activities\bin\Debug\netcoreapp3.0\Actio.Services.Activities.dll
Actio.Api -> F:\MicroService\04_MicroService\Actio\src\Actio.Api\bin\Debug\netcoreapp3.0\Actio.Api.dll
Actio.Services.Identity -> F:\MicroService\04_MicroService\Actio\src\Actio.Services.Identity\bin\Debug\netcoreapp3.0\Actio.Services.Identity.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:03.52
```

شکل ۸-۴ `build succeeded`

قسمت پنجم:

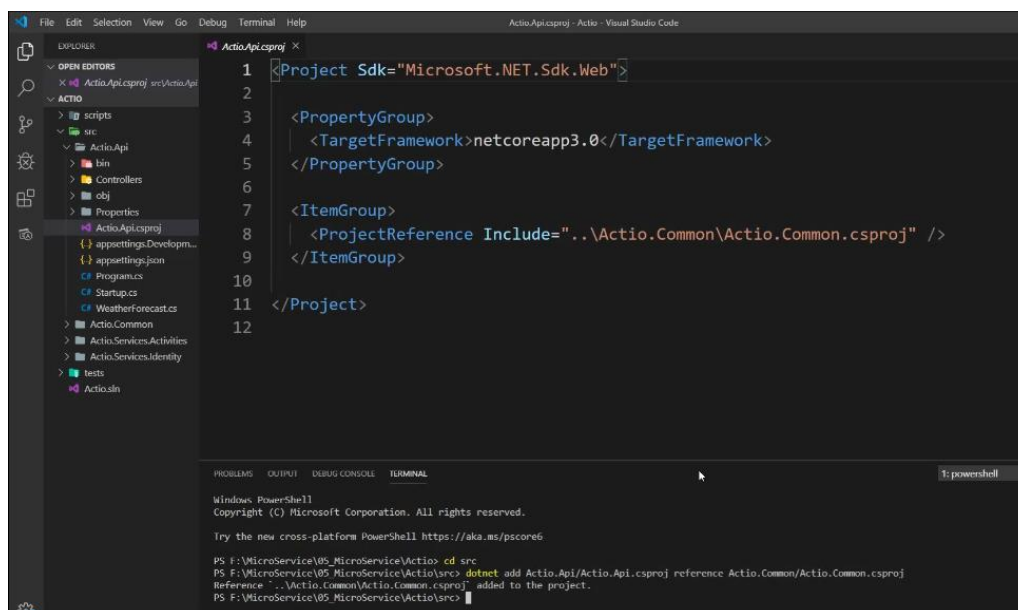
پس از اضافه کردن پروژه ها به `solution` حال باید `reference` دادن پروژه ها به یکدیگر را انجام داد. پروژه ها می بایست `reference` داشته باشند.

در ابتدا ما می بایست پروژه های `api` و `identity` و `activities` را به پروژه `common` رفرنس دهیم تا دسترسی `common` به سایر پروژه ها امکان پذیر باشد. حال باید در ترمینال وارد مسیر `src` شویم. سپس با دستور `dotnet add reference Actio.Common/Actio.Common.csproj` رفرنس می دهیم.

```
PS F:\MicroService\05_MicroService\Actio\src> dotnet add Actio.Api/Actio.Api.csproj reference Actio.Common/Actio.Common.csproj
```

شکل ۵-۱ رفرنس دادن

همچنین پس از رفرنس دادن به پروژه Common، در فایل csproj پروژه Api تگ رفرنس در آن نوشته می شود.



شکل ۵-۲ نتیجه رفرنس

- dotnet add Actio.Api/Actio.Api.csproj reference Actio.Common/Actio.Common.csproj
- dotnet add Actio.Services.Identity/Actio.Services.Identity.csproj reference Actio.Common/Actio.Common.csproj
- dotnet add Actio.Services.Activities/Actio.Services.Activities.csproj reference Actio.Common/Actio.Common.csproj

پس از رفرنس دادن تمام پروژه ها به پروژه Common حال با زدن `cd ..` به مسیر اصلی پروژه رفته و `build` را انجام می دهیم.

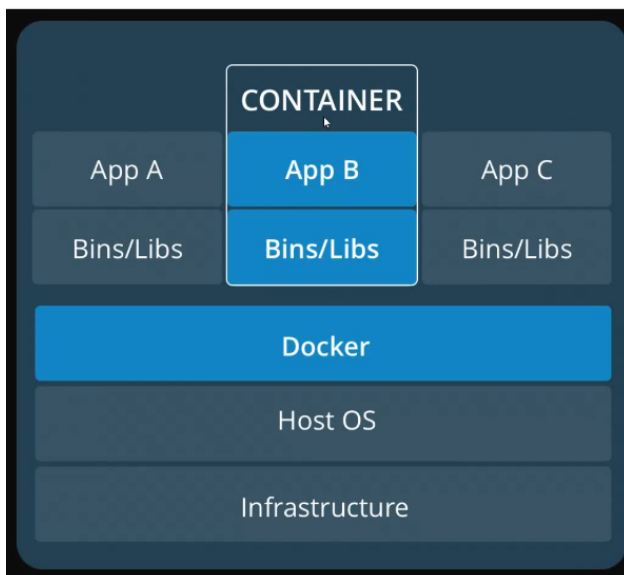
قسمت ششم:

در این دوره ما میخواهیم از داکر (Docker) استفاده بکنیم. داکر یک پلتفرم مبنی بر لینوکس است که مشابه ویرچوال باکس (Virtual Box) یا ماشین مجازی بر روی سیستم می باشد. داکر به ما این قابلیت را میدهد که با Container هایی که به ما ارائه می دهد، آن را بخش بندی کنیم.

بطور مثال سیستم عامل ما ویندوز می باشد و اگر بخواهیم php کدنویسی کنیم ، نیازمند نصب php, Mysql, xamp و سایر ابزار های مورد نیاز میباشد. اکنون ما بجای آنکه ویندوز خود را درگیر نصب این ابزار ها کنیم، به یک محیط ایزوله روی می آوریم که هرگونه رویداد و برنامه را در همان محیط انجام داد و به سیستم عامل ما کاری و لطمه ای نشده باشد. در این موارد پای ماشین های مجازی نیز به وسط کشیده می شود که به

ویرچوال باکس یا ماشین مجازی باید نصب شود، اما ماشین مجازی یا ویرچوال باکس برای سیستم ما سنگین می باشد.

داکر یک پلتفرم متن باز (Open Source) مبتنی بر سیستم عامل لینوکس می باشد که روی سیستم عامل های مطرح مک، لینوکس، ویندوز اجرا می شود. داکر کارکردش به صورت بخش بندی می باشد، بخش بندی به ۳ قسمت از container تشکیل می شود (App A-App B-App C) ، یعنی به سه بخش ایزوله و موازی از هم کار میکنند.



شکل ۶-۱ بخش بندی داکر

برای کار کردن با داکر ، کافیت آنرا نصب کنیم. برای صحت از نصب داکر می بایست WindowsPowershell را اجرا و دستور `docker --version` را وارد کرد.

```
PS C:\Users\Inan> docker --version
Docker version 19.03.1, build 74b1e89
```

شکل ۶-۲ صحت نصب داکر

داکر می تواند image هایی را دریافت کند و نصب کند، مانند image اپلیکیشن پروژه خود میتوانیم بر روی آن استفاده کنیم. بطور مثال برای تست hello world داکر، در پاورشیل خود دستور `docker pull hello-world` را وارد می کنیم.

```
PS C:\Users\Iman> docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:451ce787d12369c5df2a32c85e5a03d52cbcef6cb3586dd03075f3034f10adcd
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world:latest
```

شکل ۶-۳ image hello world

سپس برای خروجی image باید دستور *docker run hello-world* را وارد می کنیم. همچنین برای دریافت image هایی که روی سیستم داریم ، می توانیم دستور *docker image ls* را وارد کنیم. برای دریافت لیست دستورات داکر و جهت کمک گرفتن از آن، دستور *docker --help* را می توان استفاده کرد.