<u>DevOps</u>

DevOps is a combination of Development and Operations. It means Dev + Ops = DevOps. It is a culture which automates the system and improves and accelerates delivery to the client in a repeated manner. It’s basically a collaboration between the Development team and the Operations team for serving a better quality application. It is a culture for continuous integration and continuous delivery where we make the automated build system as well as automated deployment system. In other words, DevOps is practice collaboration between Development and Operations from the planning of a project to deployment on production. It involves the complete SDLC life cycle as well as monitoring.

Understanding DevOps, you should consider the following points as well.

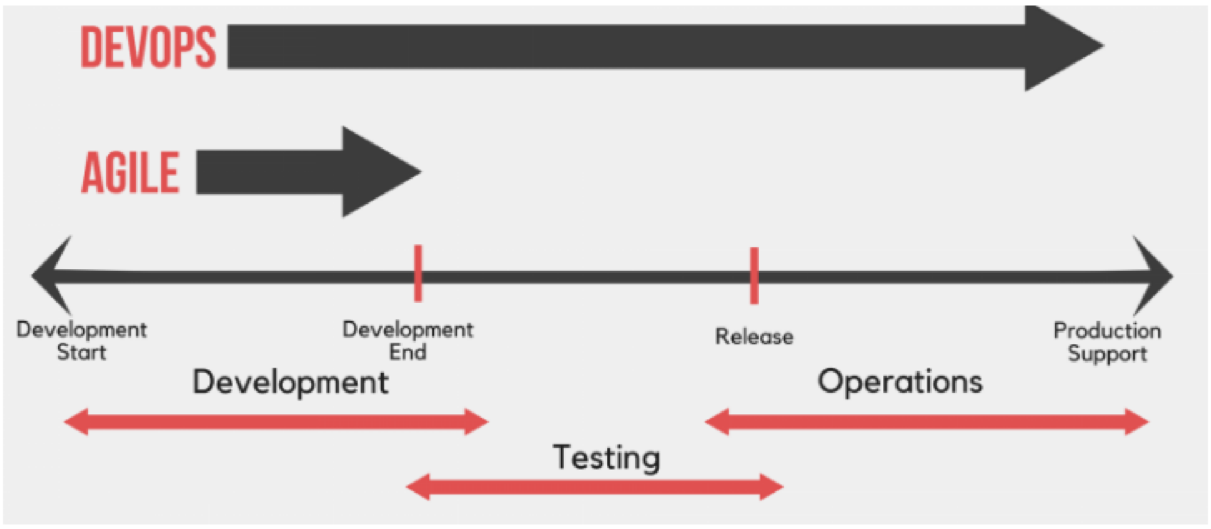
✔ DevOps means only combining the two teams of Development and Operations.

✔ It is not a separate team.

✔ It is not a product or tool.

✔ DevOps people do not hire from outside, they are internal team members who are working either in the development phase or in the operations phase.

It is a group of people, processes and tools. It basically brings the two or more different teams like development and operations together with a well-defined process, using some great tools for automatic software delivery to the client. It is a set of practices which are used by the DevOps teams to speed up the quality delivery. There are different kinds of tools or set of tools which are used in Continuous Integration (CI) and Continuous Delivery (CD) where it performs restoring the code, building the processes, executing the test cases, and deployment on the stage environment, etc.



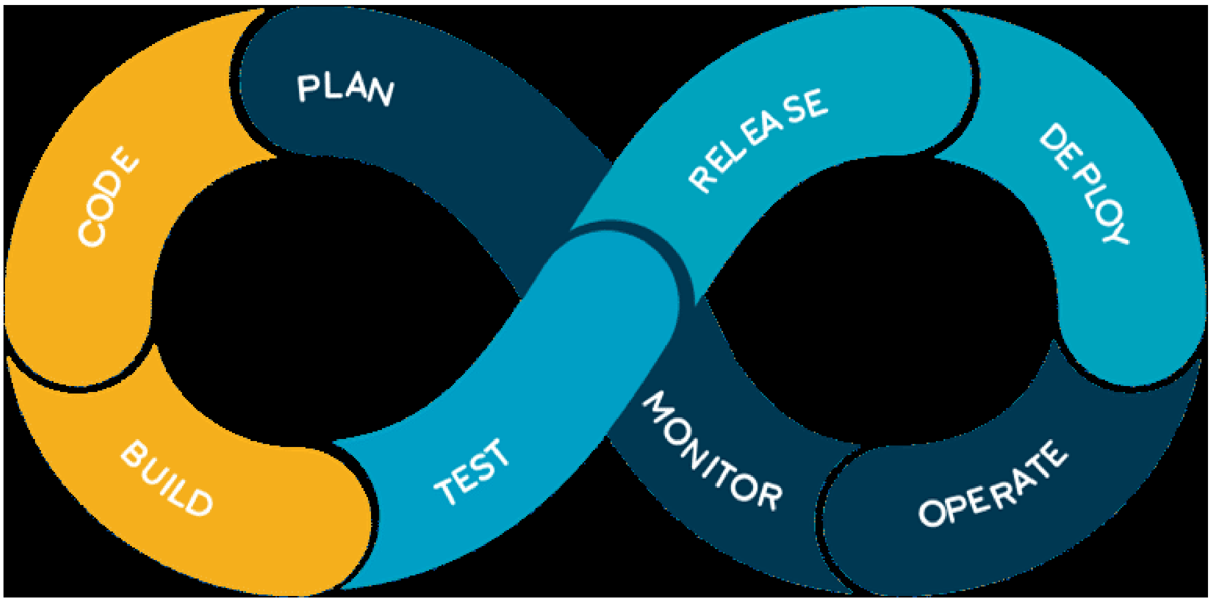
Why DevOps

To understand why DevOps is required, let's first understand, what happens without DevOps.
As an IT consulting firm, while initiating a new project, we have two different kinds of teams. First is the Development team, which is involved completely in developing and testing, including writing code and unit test cases. The other team is the Operations team which is involved in operating and monitoring the product.
✔ Without DevOps, both teams (Development and Operations) work completely in an isolated manner.
✔ If DevOps is not there then the team spends most of the time in building the code and deploying on multiple environments.
✔ Each team waits for others to be done. This means, if development is going on then the testing team waits for the deployment of the code (Artifact) on the QA environment and in the same manner the operations team waits for the deployment on the Production environment. So, due to this, a large amount of time gets wasted.
✔ As a human being, we make mistakes. Every time, building the code and deploying on specific environments in a manual fashion can increase the issues and resolving it will take a lot of time. Rather than doing it manually, we can make it automated using DevOps.
So, the above points show that we face lots of human issues and system issues if we are not following DevOps. Now, let's see how we can make it more systematic using DevOps and how DevOps helps us to achieve the same tasks in less time without any errors.
✔ DevOps increases the higher success rate of new releases without any error.

✔ It helps us to simplify the whole development and deployment process.
✔ Automates the manual process like build process, release process, etc.
✔ Automates executing the test cases.
✔ Configures the continuous delivery and continuous deployment in the release cycle.
✔ Live monitoring.
✔ It also helps in team collaboration.
✔ Reduces the failures and rollbacks
✔ Provides continuous improvement

DevOps Lifecycle

DevOps is a culture where Development and Operations teams get involved. The development phase has its own lifecycle and the Operations phase has its own as well. If we combine both lifecycles, we get the lifecycle of DevOps. If an organization is not following or considering some of the points from the DevOps lifecycle then we can say, they are not following DevOps culture. From planning to deployment, we have several stages which are very important and we cannot skip any of them. So, let's understand the DevOps lifecycle.



PLAN:

It is the first stage of any new project. Here, we plan for a new project from requirement gathering from the customer and planning to deliver the final project to the customer.

✔ What the requirements are.
✔ What types of product we have to create.
✔ What the timelines are for different sprints and the final product.
✔ What technologies we will use.
✔ What tools we will use.
✔ How many team members will be available for this project.
✔ What process we are going to follow, like agile.

CODE:

In this stage, we do the coding for creating the product with actual functionality as discussed with the customer. We use different types of methodologies for achieving the goal, like Agile methodology. Here we group the tasks in the sprint and their estimation as well. Sprints are basically for 2-3 weeks. Unit Test cases are also to be a part of the coding.

BUILD:

In this stage, we build the code. Code building happens two times; first when a developer is writing the code, then he/she has to build the code every time in their own local system to see the functionality. The second time, when a team member checks in the code in source control repository, then it automates the build for the code and makes the artifact for deployment.

TEST:

Testing is the heart of any development process. We write the Unit Test cases along with code. In DevOps, test cases auto execute and validate the build process.

RELEASE:

In this stage, it collects the build artifact which can deploy further.

DEPLOY:

Here, we start the deployment on the respective stages which are configured in the Release Pipeline. Actually after testing and validating the build artifact, it auto starts the deployment on the respective environment using a continuous delivery process. But before deploying it to the production environment, it asks for approval which can be done manually.

We can also automate the whole deployment process using continuous deployment. This is basically used when we have small changes which can be deployed on production as well without any approvals.

OPERATE & MONITOR: After successful deployment on the production environment, we have to operate the whole system and monitor the application. This monitoring is not only the performance but also the functionality.

Prerequisites
In order to do this practical demonstration, we will require a few tool and some accounts.
Visual Studio 2017 or Higher Version: It is a world class IDE which supports more than 40 programming languages. We will create the sample application along with a test project using Visual Studio 2017. So, before starting the demonstration, we will require Visual Studio 2017. If you would like to download Visual Studio 2017, we can download it from HERE.
Alternatively,
1. We can also use the Visual Studio Code, which is totally open source and can be download from HERE.
2. We can also create the Virtual Machine on the cloud (Cloud Account is required) and install Visual Studio 2017.

Azure DevOps: We need an Azure DevOps account. If we have an account with Azure DevOps that’s fine; otherwise we can sign up from HERE. Earlier it was known as VSTS (Visual Studio Team Service). We can also get the free trial Azure DevOps account. Just click on the button ‘Get Azure DevOps Free’ as follows.

CI/CD Pipeline

Now a days, the delivery process in software development is rapid and fast. With the help of several tools, we try to customize our delivery process. If delivery process will be smooth then we can deliver the high quality of product to customer within timeline. Actually, customer wants to get the small change or small functionality to be added in minimum time. To enable to faster delivery process, we take the help of some of the mechanism like agile and various tools and people and it is called DevOps.
Let see what the old process in development are and what happens if some issues are there.
1. Get the requirements from Business.
2. Make a plan for converting the requirements into the actual functionality.
3. Developers do the code and check in it to a repository like TFS, GitHub etc.
4. After all code checks in to the repository, a developer executes the build process.
5. Run the test cases against the code.
6. If everything is fine, code gets deployed on the DEV server and then on QA.
7. If QA confirms the OK then deployment goes started on PROD.

Above is the general development workflows which are used mostly in small organizations. But is this process correct? Let understand the problem with the above development and delivery process.

1. Every time a developer check in the code into repository, but he/she doesn't know about Build. Is build created successfully or not?
2. Developers should wait for other developers to do the check in the code and building the code before deploying to DEV/QA server.
3. If a developer has completed his/her work and has checked in the code, they have to wait for others developers to check in the code as well.
4. If something wrong with anyone code then builds cycle will stop and wait again till the issue is resolved.
5. As a human being, we do a lot of mistakes. We can also do a mistake while doing the manual deployment.
6. Chances to get more issues from development to deployment.

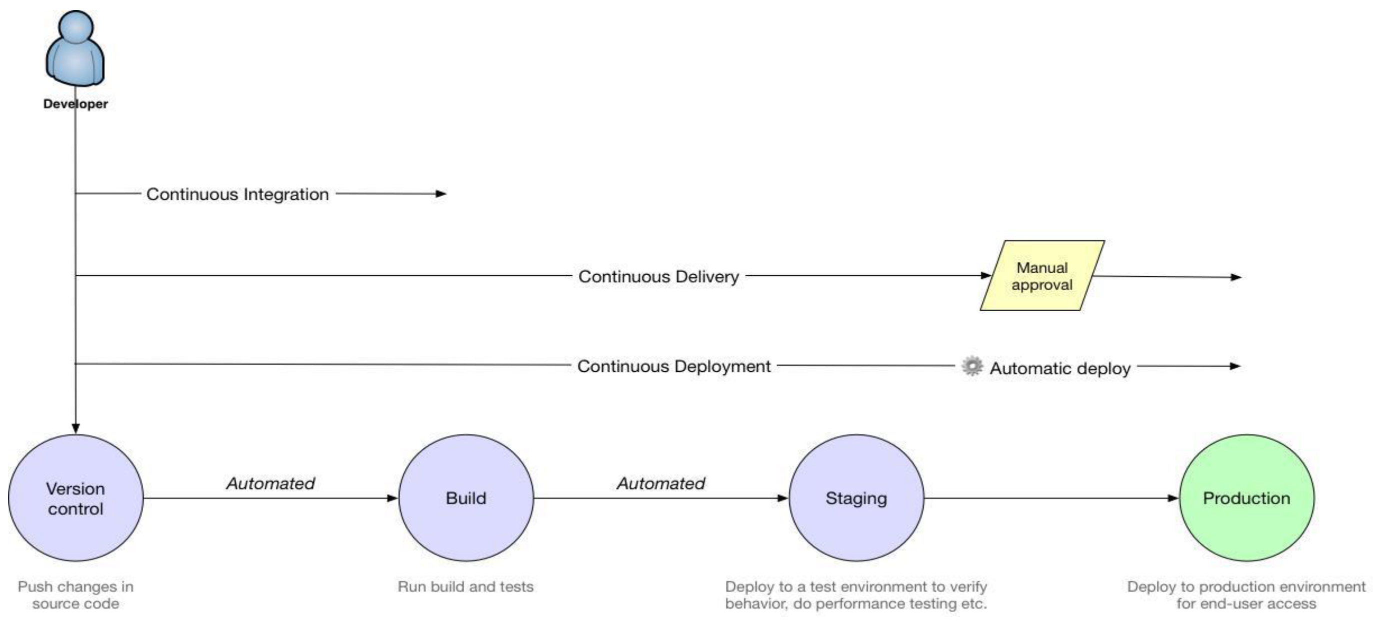
As we can see with above points, there are several issues while using the old deployment process. Which also need more time as doing manual deployment, more resource for completing manual deployment and obviously more money. We can resolve all the above issues if we implement the Azure DevOps. In Azure DevOps, we have Continuous Integration (CI) and Continuous Delivery (CD). These help us to make the whole process as automated. So, let understand what these are.

Continuous Integration (CI): It is an automated build process which starts automatically when a developer checks in the code into the respective branch. Once the code check in process is done, the Continuous Integration process starts. It fetches the latest code from the respective branch and restore the required packages and start building automatically. After build process, it auto starts it auto starts executing the Unit Test Cases and at the end, build artifact is ready. This build artifact further will use for deployment in Release process.

Continuous Delivery (CD): It is the next process after Continuous Integration. Once build artifact is ready for deployment then release process gets started. As per the Azure DevOps Release Pipeline configuration, it starts deploying the build artifact on the different stage server (Environments) automatically. But in the Continuous

Delivery, the deployment on the production goes manual. Manual does not mean here that we will deploy the build artifact manually, but here we have to approve before deploying on the PROD.

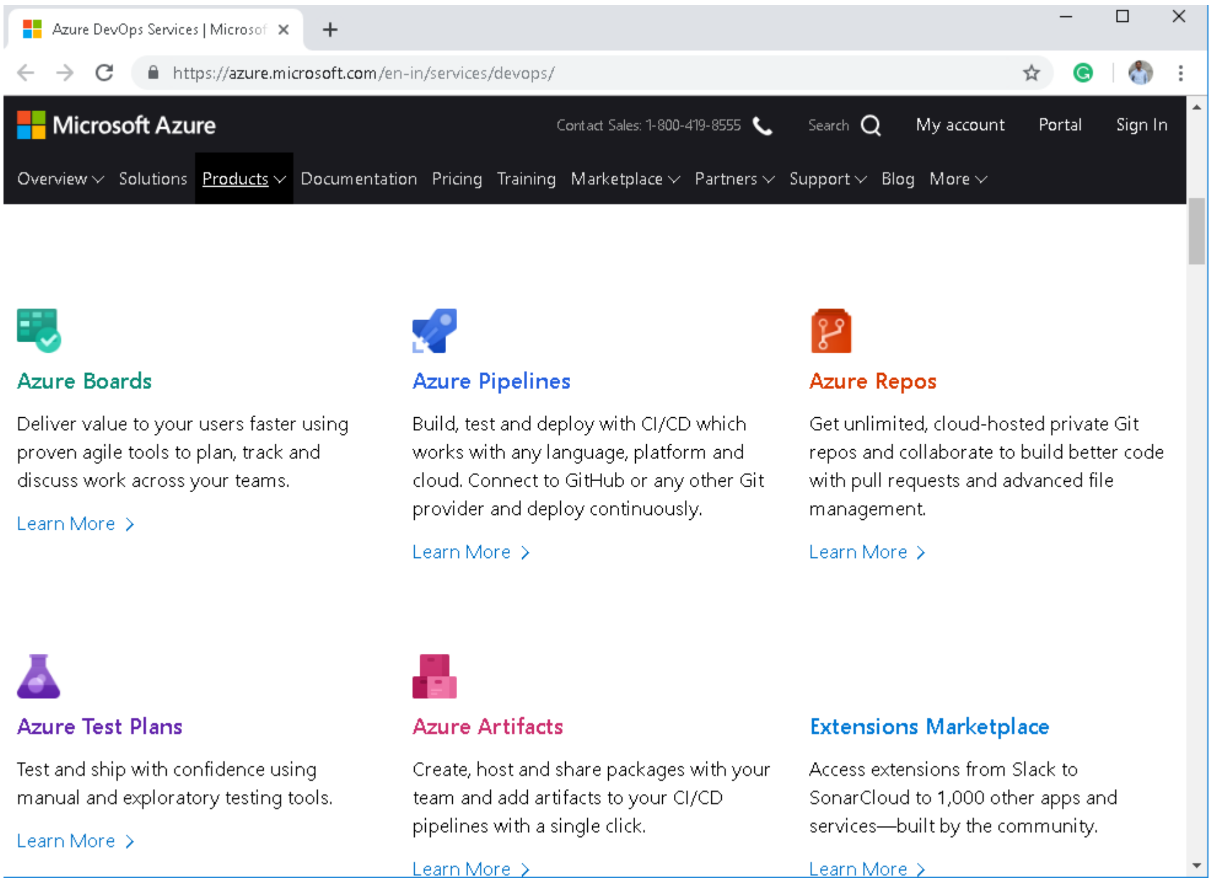
<u>Continuous Deployment: I</u>t is the combination of CI + CD and deployment on the production without any approval. It means, in the Continuous Deployment, everything goes automatically.
As per the above image, we can see that once a developer checks in the code into the Version Control, Azure DevOps Pipeline starts. From getting the code from Version Control, restoring it, restoring packages from NuGet, Maven etc., building the code, executing the unit tests cases are part of the Continuous Integration (CI).
Including Continuous



As per the above image, we can see that once a developer checks in the code into the Version Control, Azure DevOps Pipeline starts. From getting the code from Version Control, restoring it, restoring packages from NuGet, Maven etc., building the code, executing the unit tests cases are part of the Continuous Integration (CI).
Including Continuous Integration, if build artifact gets auto-deployed on any staging server like DEV or QA and deploys on the Production after manual approval is called Continuous Delivery (CD). If manual approval is converted into automatic deployment then it’s called Continuous Deployment. So, basically Continuous Deployment is the same as Continuous Delivery but without any manual approval. All goes in auto approval mode.

Why Azure DevOps

DevOps is a group of people, processes and tools which enable and automate the continuous delivery. Azure DevOps formally known as Visual Studio Team Service (VSTS) provides the repository management, project management, Build/Release Pipeline etc. Azure DevOps is free for a small project which has up to five users. But we can also use the paid version of it.
Azure DevOps provides unlimited cloud-hosted Git repos for a small or big project. Here developer can pull the code from Repos or push the code into Repos. It is basically a complete file management system.
While building the source code, it requires lots of third-party packages. Using Azure Artifacts, it enables the NPM, MAVEN or NuGet packages will available for private or public source code.
The most important feature of Azure DevOps is Azure pipeline. It enables the automated build and release pipeline. Azure Pipeline helps us to achieve Continuous Integration and Continuous Delivery for the project



Azure Boards

Deliver value to your users faster using proven agile tools to plan, track and discuss work across your teams.

Learn More >

Azure Pipelines

Build, test and deploy with CI/CD which works with any language, platform and cloud. Connect to GitHub or any other Git provider and deploy continuously.

Learn More >

Azure Repos

Get unlimited, cloud-hosted private Git repos and collaborate to build better code with pull requests and advanced file management.

Learn More >

Azure Test Plans

Test and ship with confidence using manual and exploratory testing tools.

Learn More >

Azure Artifacts

Create, host and share packages with your team and add artifacts to your CI/CD pipelines with a single click.

Learn More >

Extensions Marketplace

Access extensions from Slack to SonarCloud to 1,000 other apps and services—built by the community.

Learn More >

Microsoft Azure DevOps is most popular and widely use throughout the world because it is free for Open Source Project and Small Teams project. We can use lots of Azure DevOps features with free version like Azure Pipeline, Azure Boards, Azure Repos, and Azure Artifacts etc. It means, we don't need to go for paid version of Azure DevOps, if and only if we want to use those features which are in free versions.

Model Class

```
public class PostViewModel
   {
       public int PostId { get; set; }
       public string Title { get; set; }
       public string Description { get; set; }
       public string author { get; set; }


   }
```

HomeController.cs

```
using FirstCoreApp.Models;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading.Tasks;
using FirstCoreApp.Models.Repository;

namespace FirstCoreApp.Controllers
{
    public class HomeController : Controller
    {
        IPostRepository postRepository;
        public HomeController(IPostRepository _postRepository)
        {
            postRepository = _postRepository;
        }
        public IActionResult Index()
        {
            var model = postRepository.GetPosts();
            return View(model);
        }
        public IActionResult About()
        {
            ViewData["Message"] = "Your application description page.";
            return View();
        }
        public IActionResult Contact()
        {
            ViewData["Message"] = "Your contact page.";
            return View();
        }
    }
}
```

Index.cshtml

```
@model IList<FirstCoreApp.Models.PostViewModel>
@{ ViewData["title"] = "HomePage";
      }
<div class="row">
    <h2> POSTLIST</h2>
    <table class="table">
        <thead>
            <tr>
                <th>
                    PostID
                </th>
                <th>
                    Title
                </th>
                <th>
                    Description
                </th>
                <th>
                    Author
                </th>
            </tr>

        </thead>
        <tbody >
            @foreach (var item in Model)
            {
            <tr>
                <td>
                    @Html.DisplayFor(Modelitem => item.postId)
                </td>
                <td>
                    @Html.DisplayFor(Modelitem => item.title)
                </td>
                <td>
                    @Html.DisplayFor(Modelitem => item.description)
                </td>
                <td>
                    @Html.DisplayFor(Modelitem => item.author)
                </td>
            </tr>
            }
        </tbody>

    </table>

</div>
```

PostRepository.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using FirstCoreApp.Models;
using FirstCoreApp.Models.Repository;

namespace FirstCoreApp.Models
{
   public class PostRepository:IPostRepository
   {
      public List<PostViewModel> GetPosts()
      {
         var posts = new List<PostViewModel> { new PostViewModel()
         {
            postId=101, title="DevOps", description="DevOps Demo",
author="john"
         },new PostViewModel()
         {
            postId=102, title="DevOps2", description="DevOps Demo",
author="john"
         },
         new PostViewModel()
         {
            postId=103, title="DevOps3", description="DevOps Demo",
author="john"
         },
         };
         return posts;
      }


   }
}
```

<u>Repository(create folder)->IPostRepository.cs(interface)</u>

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using FirstCoreApp.Models;
```

```
namespace FirstCoreApp.Models.Repository
{
   public interface IPostRepository
   {
      List<PostViewModel> GetPosts();
   }
}
```
<u>FirstCoreApp.Test->PostTestController.cs</u>
```
using System;
using Xunit;
using FirstCoreApp.Controllers;
using FirstCoreApp.Models;
using FirstCoreApp.Models.Repository;
using FirstCoreApp.Test;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;

namespace FirstCoreApp.Test
{
   public class PostTestController
   {
      private PostRepository repository;
      public PostTestController()
      {
         repository = new PostRepository();
      }
      [Fact]
      public void Test_Index_View_Result()
      {
         //Arrange
         var controller = new HomeController(this.repository);

         //Act
         var result = controller.Index();
         //Assert
         var viewResult = Assert.IsType<ViewResult>(result);
         //var model = Assert.IsAssignableFrom >
(viewResult.ViewData.Model); Assert.Equal(3, model.Count);
Assert.Equal(101, model[0].PostId); Assert.Equal("DevOps Demo Title 1",
model[0].Title);
```

```
        }
        [Fact]
        public void Test_Index_Return_Result()
        {
            //Arrange
            var controller = new HomeController(this.repository);
            //Act
            var result = controller.Index();
            //Assert
            Assert.NotNull(result);
        }
        [Fact]
        public void Test_Index_GetPosts_MatchData()
        {
            //Arrange
            var controller = new HomeController(this.repository);
            //Act
            var result = controller.Index();
            //Assert
            var viewResult = Assert.IsType<ViewResult>(result);
            var model =
Assert.IsAssignableFrom<List<PostViewModel>>(viewResult.ViewData.Model
);
            Assert.Equal(3, model.Count);
            Assert.Equal(101, model[0].postId);
            Assert.Equal("DevOps", model[0].title);
        }
    }
}
```