PRACTICAL NO - 01

Develop an ASP.NET Core MVC based Stateless Web App

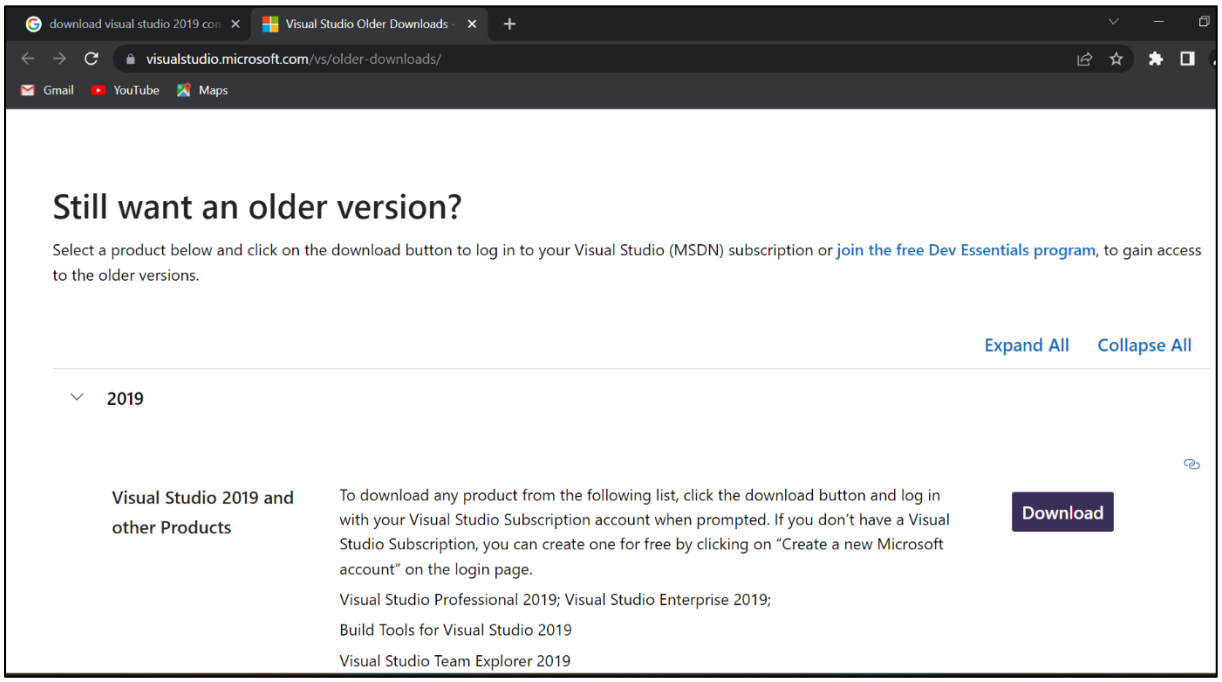
Prerequisites:

1. Visual Studio 2017 / Visual Studio 2019
2. Download and Install the Microsoft Azure Service Fabric

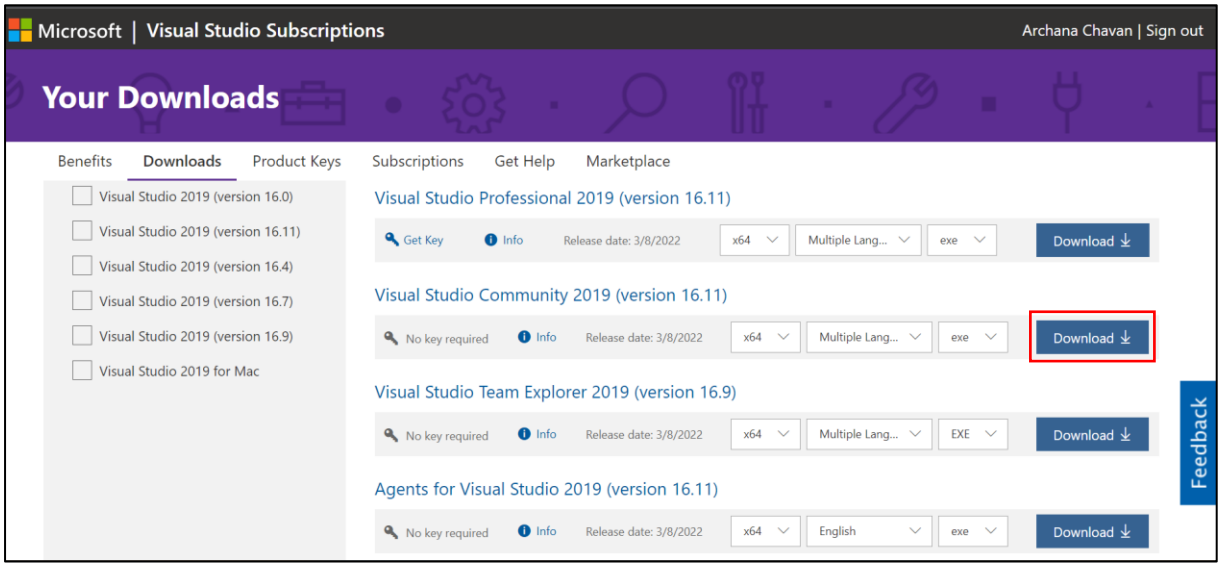
Downloading and Installation:

1. Visual Studio 2019 Community Edition

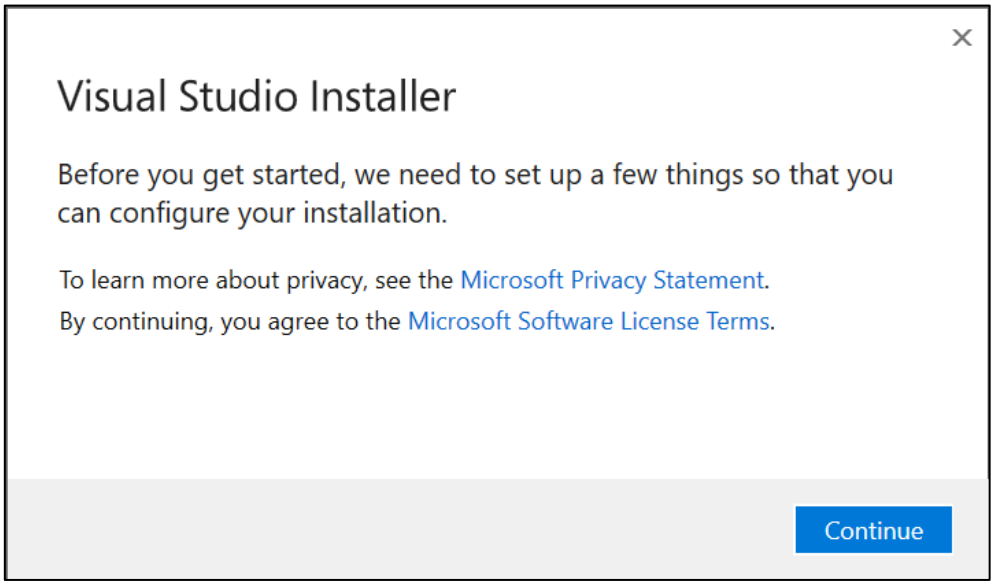
- Go to https://visualstudio.microsoft.com/vs/older-downloads/ > Expand 2019 > click Download

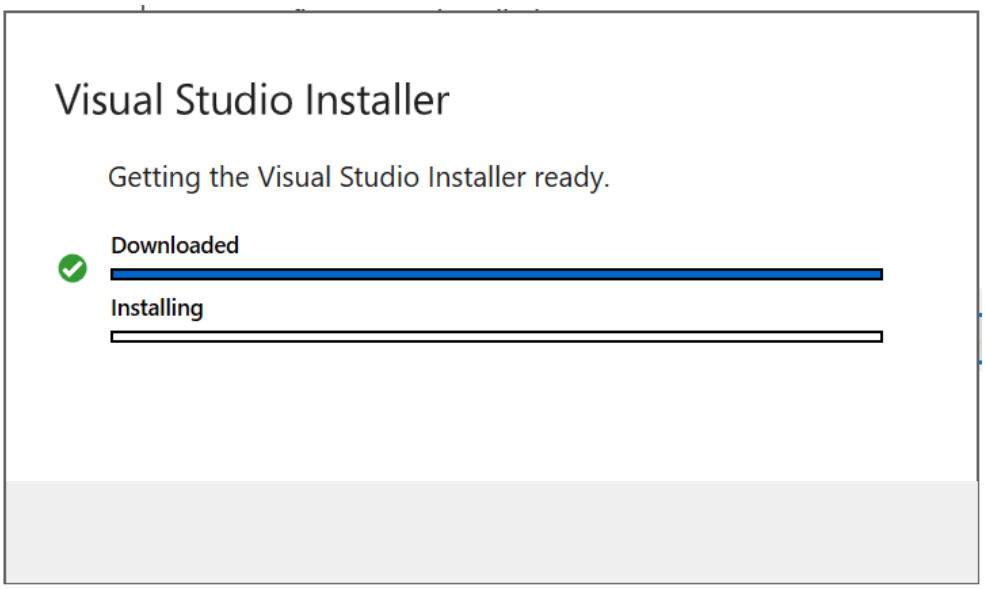


- It will ask you to log in to your Microsoft account. After successful login downloading window will open. Click on the Download button that appeared against Visual Studio Community 2019.

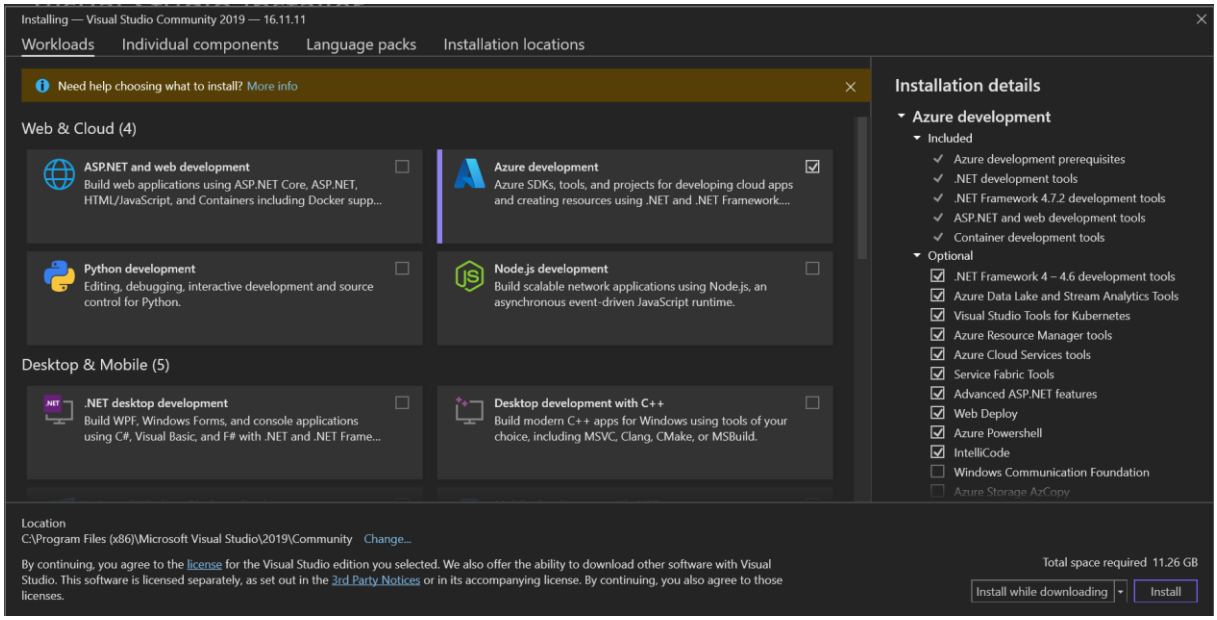


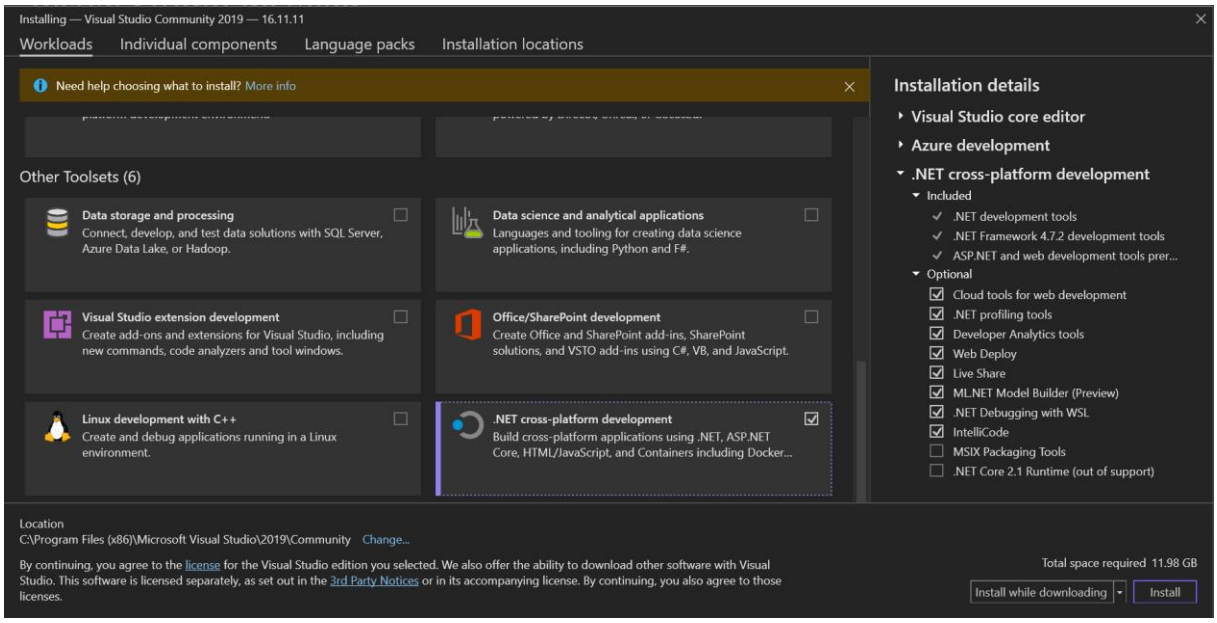
- It will download the Visual studio web installer file. Double click on it. It will ask you for permission. Click on Yes.
- The Visual Studio Installer Window would open. Click on Continue.



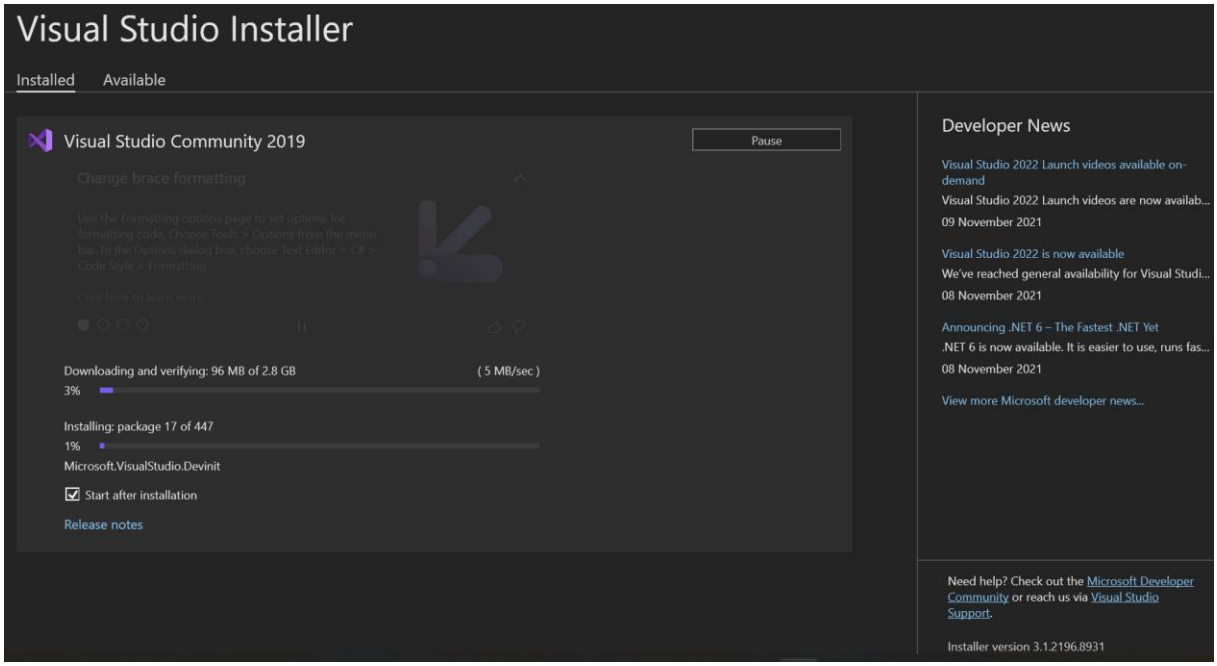


- The Installing window will open. Select *Azure development* from Web and Cloud and *.NET cross-platform development* from other toolsets > Click Install

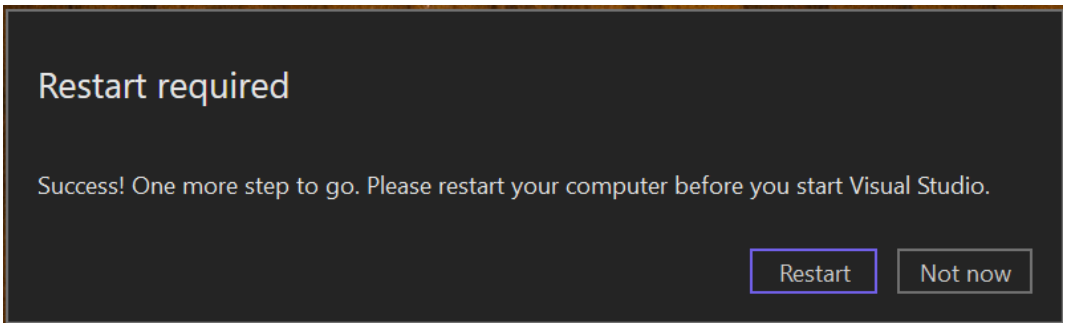




- It will start downloading and installing process for Visual Studio Community Edition at the same time. Wait for some time till the installation process is finished.

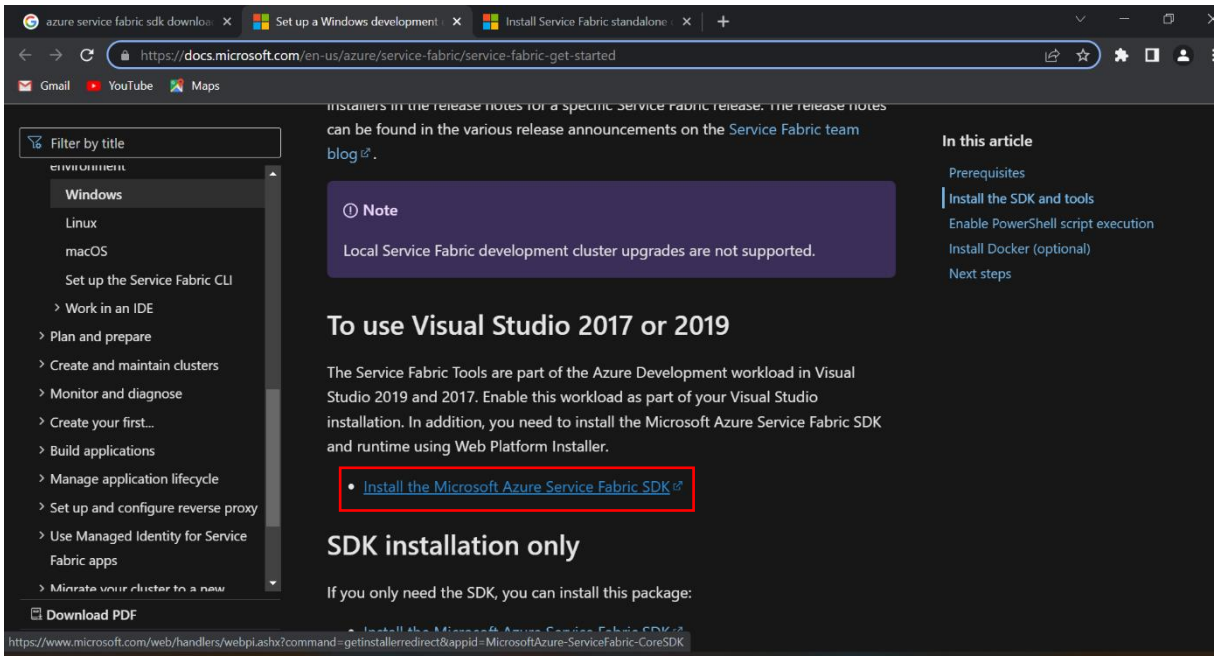


- After successful installation it will ask you to restart the machine. Click on Restart.

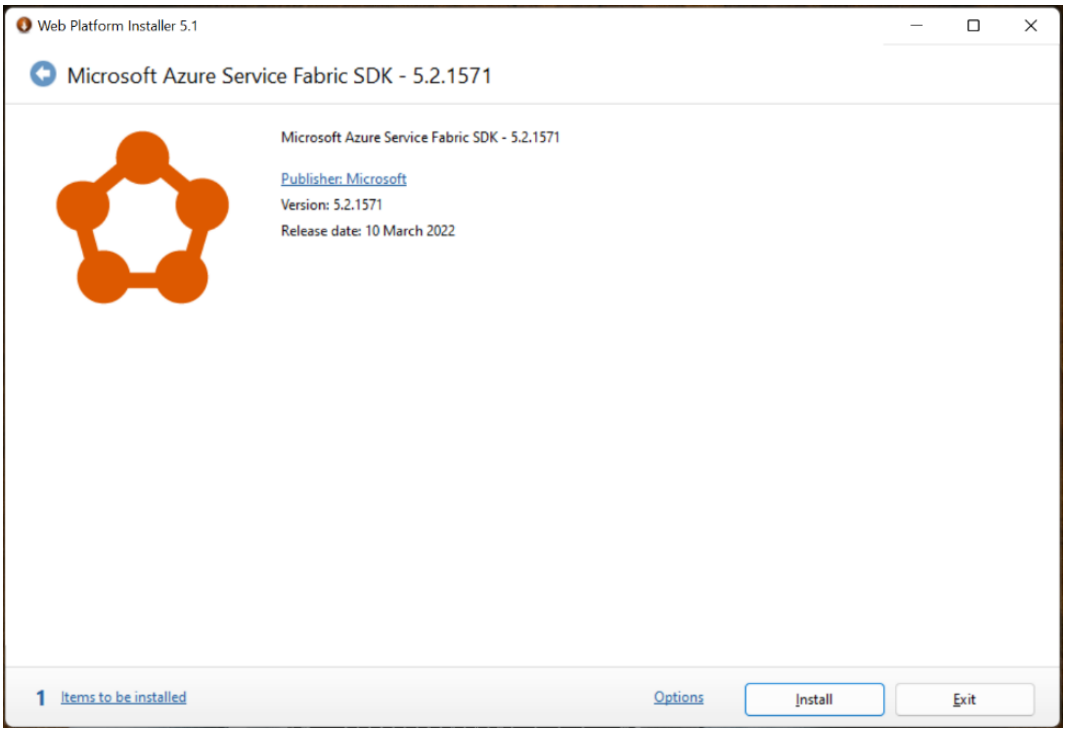


2. Install the Microsoft Azure Service Fabric SDK

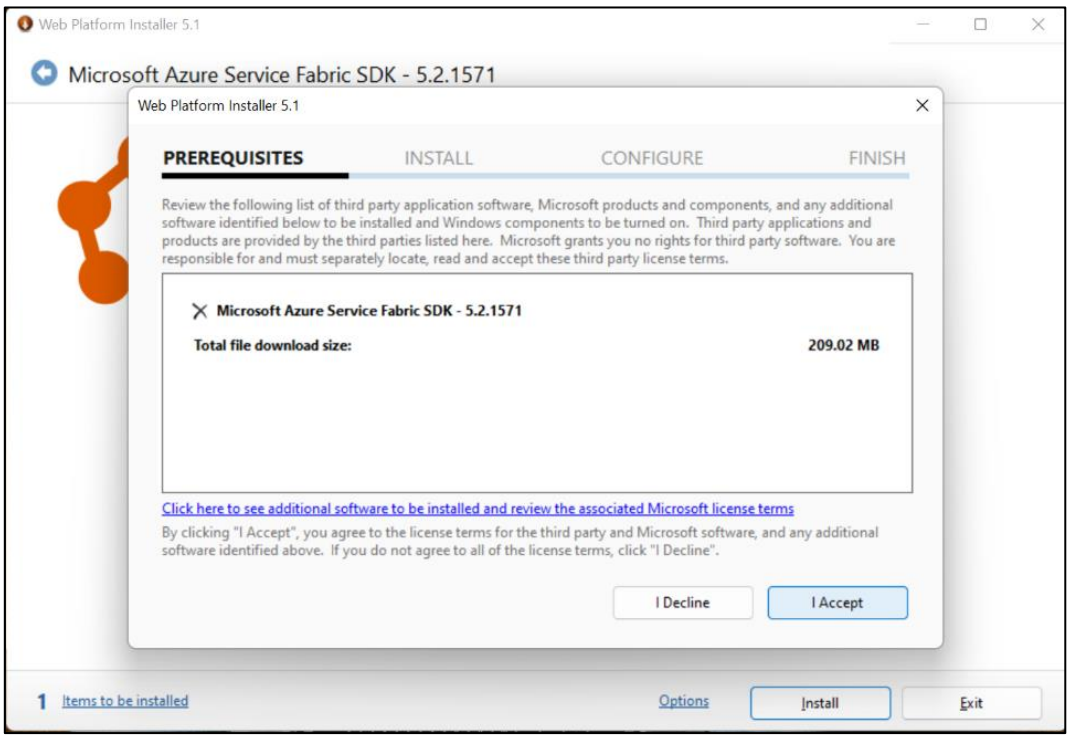
- Go to https://docs.microsoft.com/en-us/azure/service-fabric/service-fabric-get-started > Click on *Install the Microsoft Azure Service Fabric SDK*, To use Visual Studio 2017 or 2019



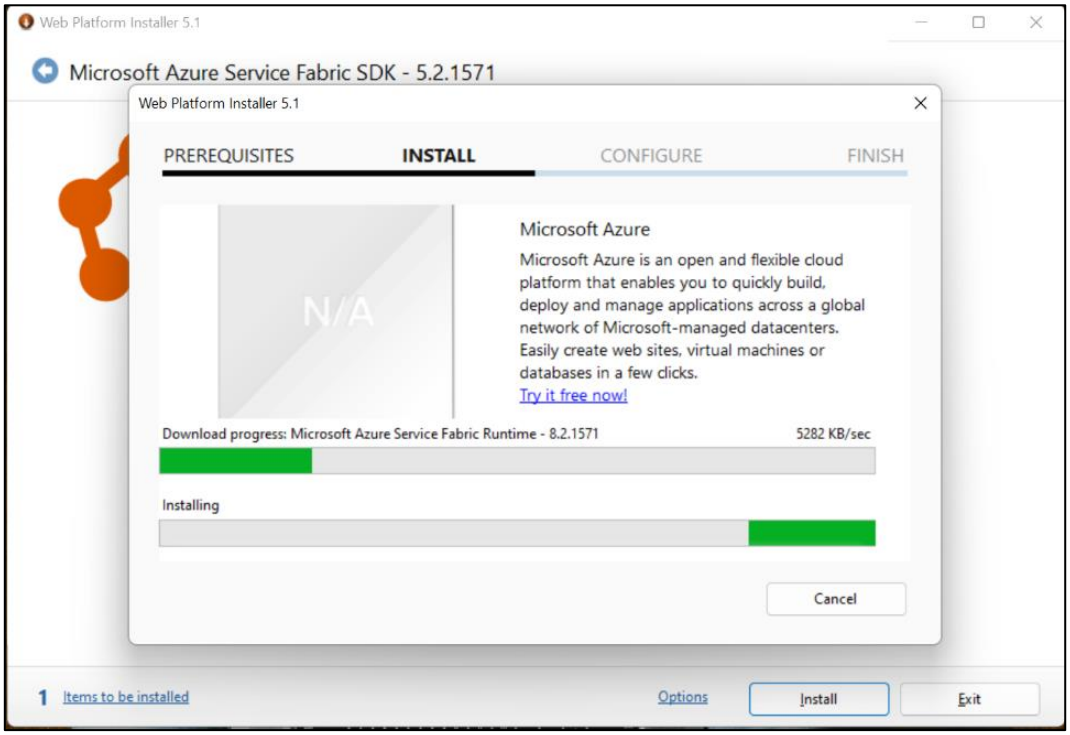
- It will download the setup file. Run it. It would ask you for permission, click on Yes. After that, the web installer window for Microsoft Azure Service Fabric SDK will open. Click on Install.



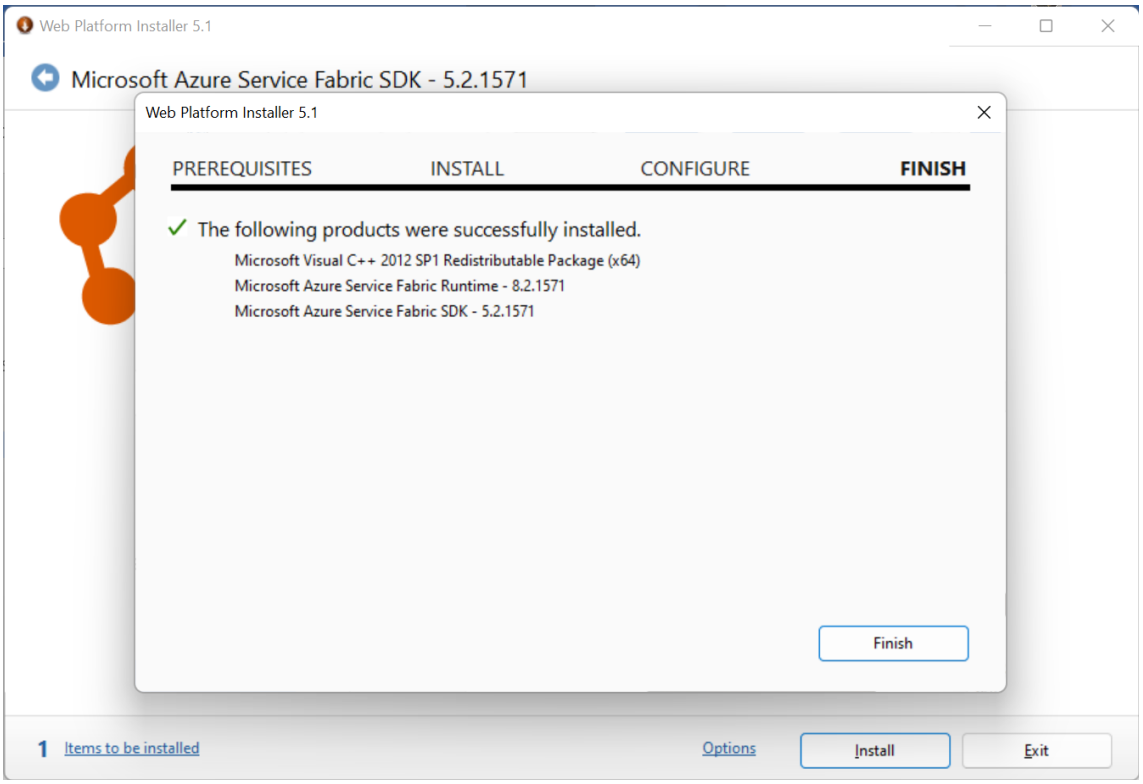
- Click on I Accept.



- The installation process will begin.

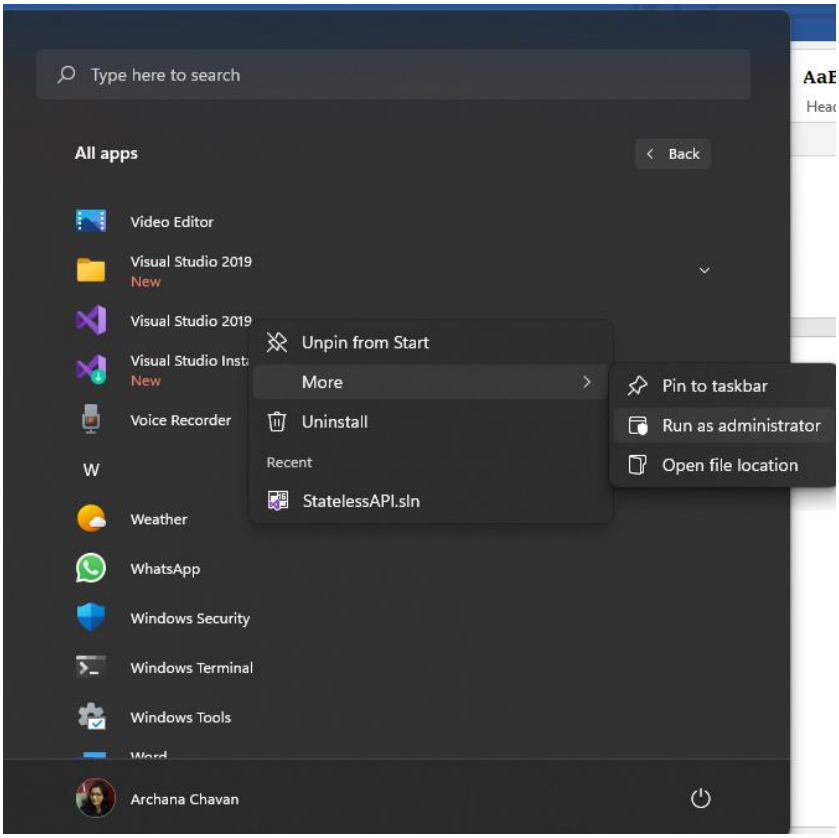


- Finally Click on Finish and Exit.

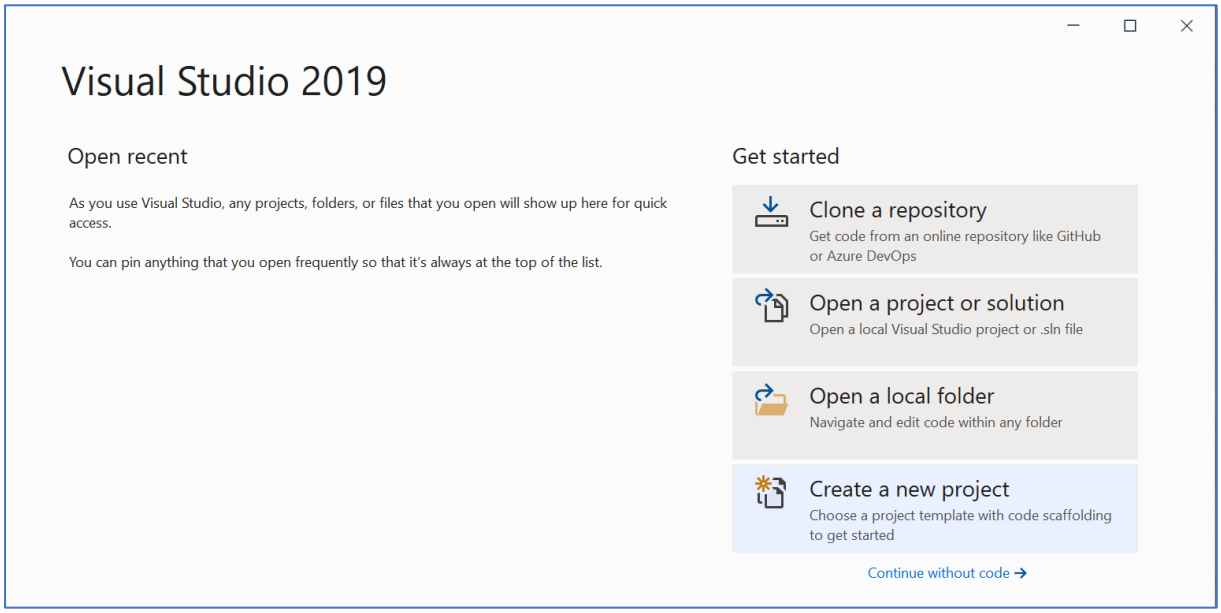


Steps:

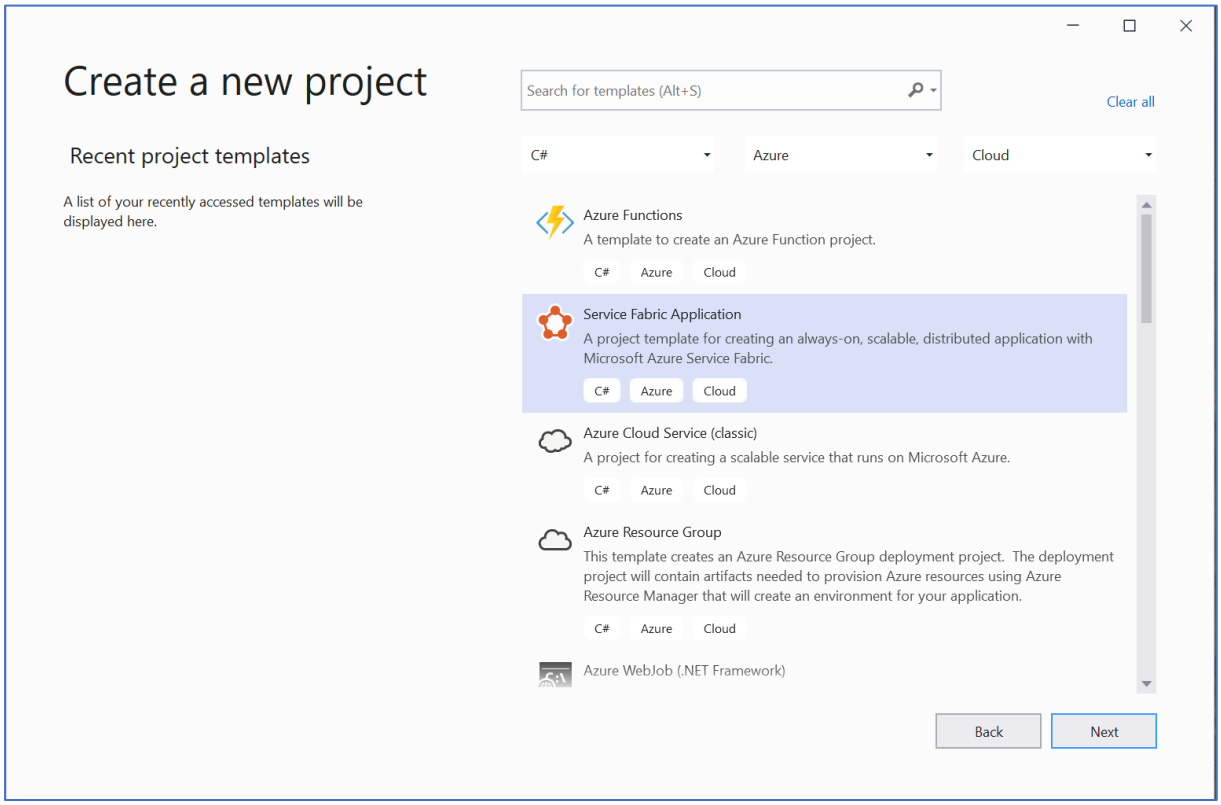
1. Create Project
   - Launch Visual Studio 2019 as an administrator.
   - Click on Windows icon > All Apps > Visual Studio 2019 > More > Run as Administrator



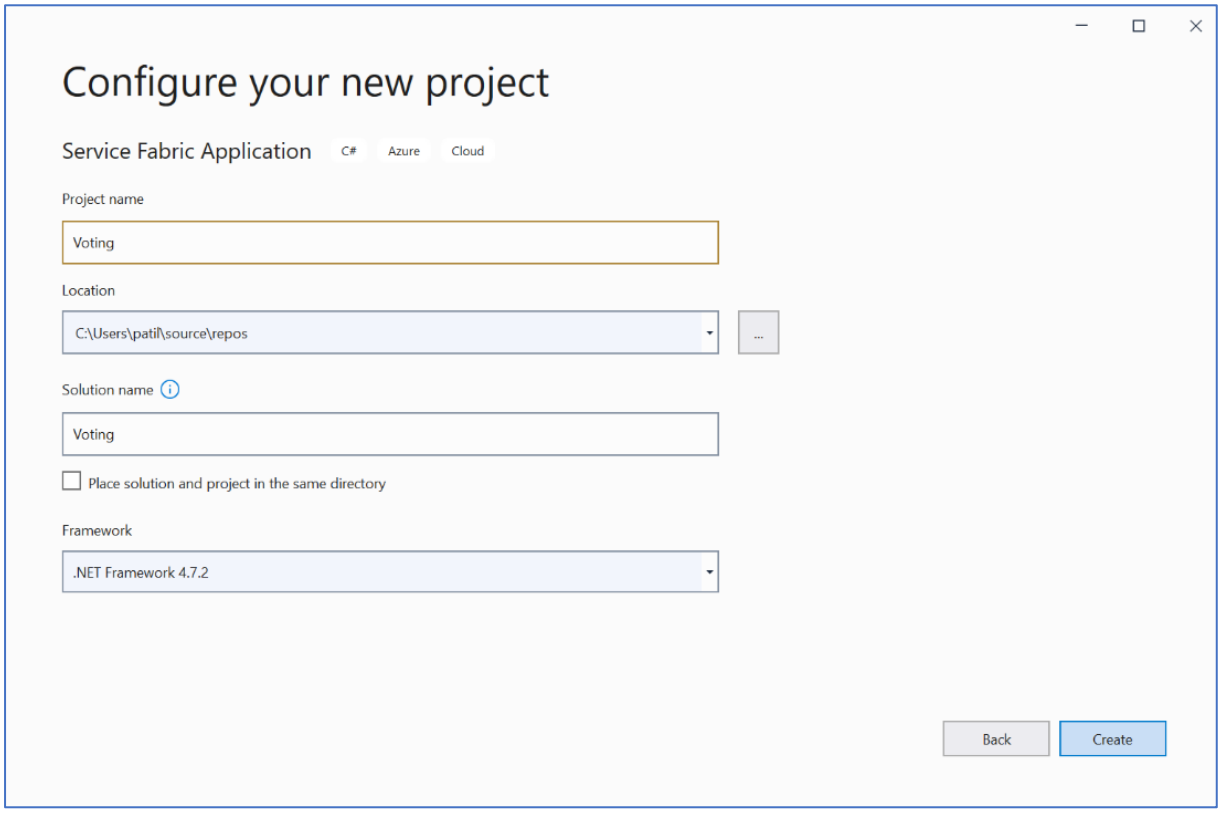
   - Now Click on *Create a new project*



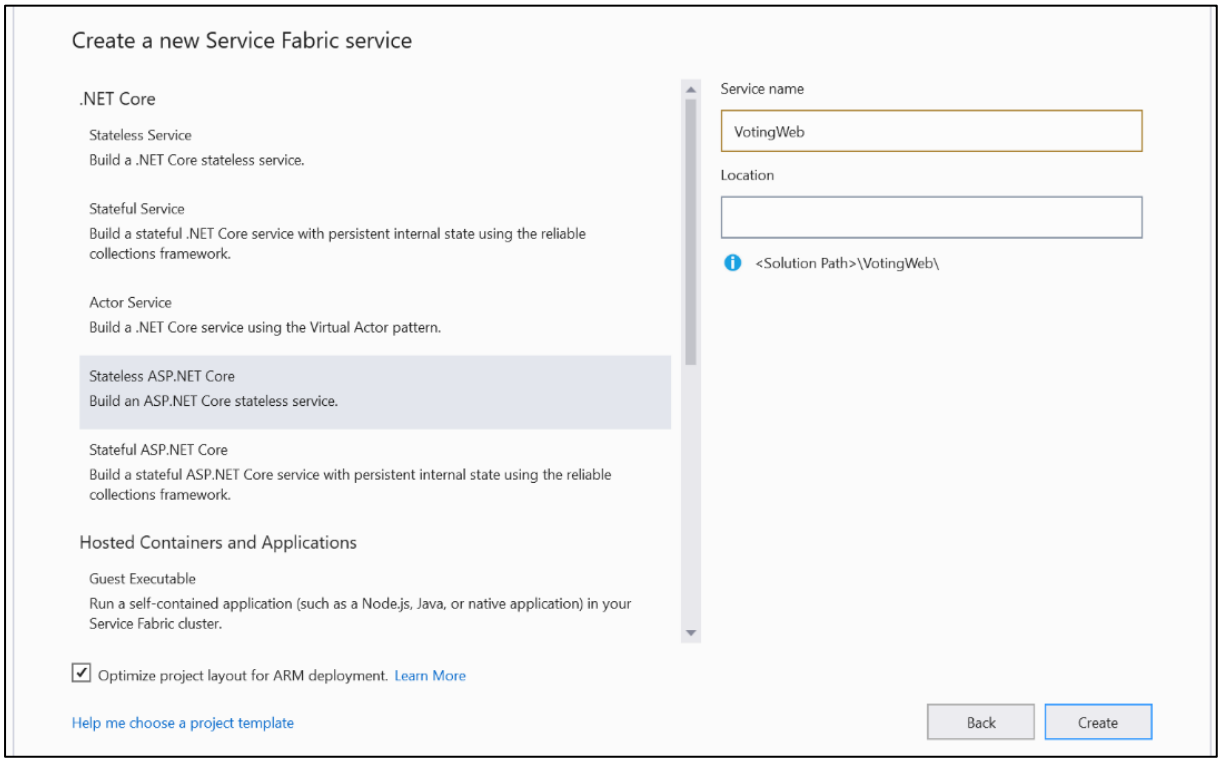
- Now select *Service Fabric Application* and click on *Next*



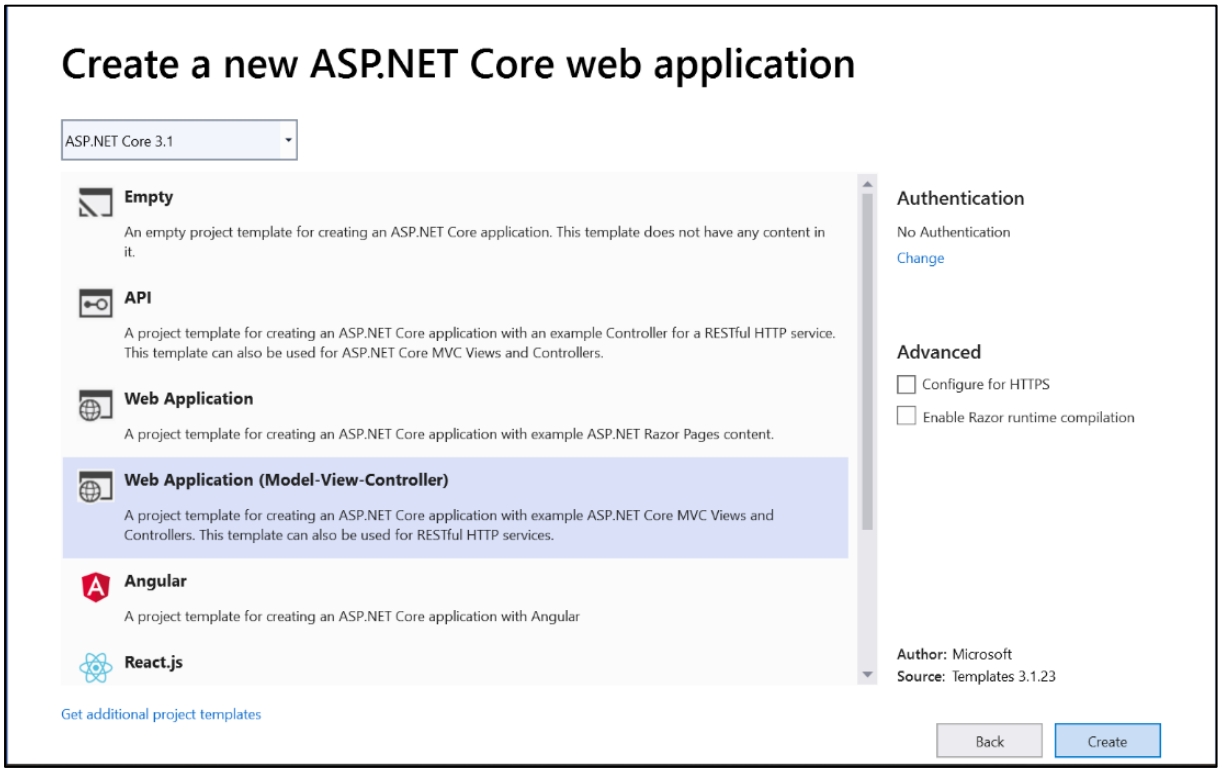
- Name the Project as Voting and click *Create*.



- On the New Service Fabric Service page, choose *Stateless ASP.NET Core*, name your service *VotingWeb*, then click Create.



- The next page provides a set of ASP.NET Core project templates. For this practical, choose Web Application (Model-View-Controller), then click Create.



- Visual Studio creates an application and a service project and displays them in Solution Explorer.

2. Create and Update the Files

- In this step we will create and will update file in the VotingWeb project

2.1 Update the site.js file

- Open VotingWeb/wwwroot/js/site.js. Replace its contents with the following JavaScript used by the Home views, then save your changes.

```
var app = angular.module('VotingApp', ['ui.bootstrap']);
app.run(function () { });

app.controller('VotingAppController', ['$rootScope', '$scope', '$http',
'$timeout', function ($rootScope, $scope, $http, $timeout) {

    $scope.refresh = function () {
        $http.get('api/Votes?c=' + new Date().getTime())
            .then(function (data, status) {
                $scope.votes = data;
            }, function (data, status) {
                $scope.votes = undefined;
            });
    };
```

```
    $scope.remove = function (item) {
        $http.delete('api/Votes/' + item)
            .then(function (data, status) {
                $scope.refresh();
            })
    };

    $scope.add = function (item) {
        var fd = new FormData();
        fd.append('item', item);
        $http.put('api/Votes/' + item, fd, {
            transformRequest: angular.identity,
            headers: { 'Content-Type': undefined }
        })
            .then(function (data, status) {
                $scope.refresh();
                $scope.item = undefined;
            })
    };
}]);
```

2.2 Update the Index.cshtml file

- Open VotingWeb/Views/Home/Index.cshtml, the view specific to the Home controller. Replace its contents with the following, then save your changes.

```
@{
    ViewData["Title"] = "Service Fabric Voting Sample";
}

<div ng-controller="VotingAppController" ng-init="refresh()">
    <div class="container-fluid">
        <div class="row">
            <div class="col-xs-8 col-xs-offset-2 text-center">
                <h2>Service Fabric Voting Sample</h2>
            </div>
        </div>

        <div class="row">
            <div class="col-xs-8 col-xs-offset-2">
                <form class="col-xs-12 center-block">
                    <div class="col-xs-6 form-group">
                        <input id="txtAdd" type="text" class="form-control"
placeholder="Add voting option" ng-model="item" />
                    </div>
                    <button id="btnAdd" class="btn btn-default" ng-
click="add(item)">
                        <span class="glyphicon glyphicon-plus" aria-
hidden="true"></span>
                        Add
                    </button>
```

```
                </form>
            </div>
        </div>

        <hr />

        <div class="row">
            <div class="col-xs-8 col-xs-offset-2">
                <div class="row">
                    <div class="col-xs-4">
                        Click to vote
                    </div>
                </div>
                <div class="row top-buffer" ng-repeat="vote in votes.data">
                    <div class="col-xs-8">
                        <button class="btn btn-success text-left btn-block" ng-
click="add(vote.key)">
                            <span class="pull-left">
                                {{vote.key}}
                            </span>
                            <span class="badge pull-right">
                                {{vote.value}} Votes
                            </span>
                        </button>
                    </div>
                    <div class="col-xs-4">
                        <button class="btn btn-danger pull-right btn-block" ng-
click="remove(vote.key)">
                            <span class="glyphicon glyphicon-remove" aria-
hidden="true"></span>
                            Remove
                        </button>
                    </div>
                </div>
            </div>
        </div>
    </div>
</div>
```

2.3 Update the _Layout.cshtml file

- Open VotingWeb/Views/Shared/_Layout.cshtml, the default layout for the ASP.NET app. Replace its contents with the following, then save your changes.

```
<!DOCTYPE html>
<html ng-app="VotingApp" xmlns:ng="https://angularjs.org">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"]</title>

    <link href="~/lib/bootstrap/dist/css/bootstrap.css" rel="stylesheet" />
    <link href="~/css/site.css" rel="stylesheet" />

</head>
```

```
<body>
    <div class="container body-content">
        @RenderBody()
    </div>

    <script src="~/lib/jquery/dist/jquery.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.7.2/angular.min.js"></scr
ipt>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/angular-ui-
bootstrap/2.5.0/ui-bootstrap-tpls.js"></script>
    <script src="~/js/site.js"></script>

    @RenderSection("Scripts", required: false)
</body>
</html>
```

2.4 Update the VotingWeb.cs file

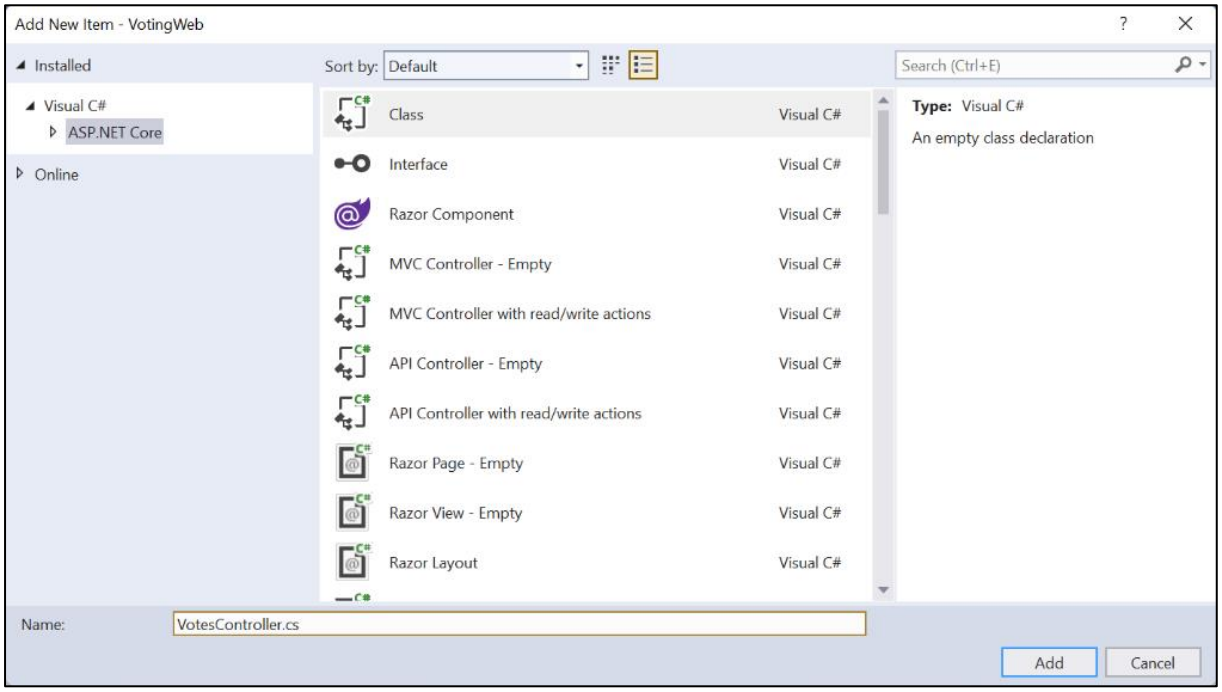
- Open the VotingWeb.cs file, which creates the ASP.NET Core WebHost inside the stateless service using the WebListener web server.
- Replace the content with the following code, then save your changes.

```
namespace VotingWeb
{
    using System;
    using System.Collections.Generic;
    using System.Fabric;
    using System.IO;
    using System.Net.Http;
    using Microsoft.AspNetCore.Hosting;
    using Microsoft.Extensions.Logging;
    using Microsoft.Extensions.DependencyInjection;
    using Microsoft.ServiceFabric.Services.Communication.AspNetCore;
    using Microsoft.ServiceFabric.Services.Communication.Runtime;
    using Microsoft.ServiceFabric.Services.Runtime;

    internal sealed class VotingWeb : StatelessService
    {
        public VotingWeb(StatelessServiceContext context)
            : base(context)
        {
        }

        protected override IEnumerable<ServiceInstanceListener>
CreateServiceInstanceListeners()
```

```
        {
            return new ServiceInstanceListener[]
            {
        new ServiceInstanceListener(
            serviceContext =>
                new KestrelCommunicationListener(
                    serviceContext,
                    "ServiceEndpoint",
                    (url, listener) =>
                    {
                        ServiceEventSource.Current.ServiceMessage(serviceContext,
$"Starting Kestrel on {url}");

                        return new WebHostBuilder()
                            .UseKestrel()
                            .ConfigureServices(
                                services => services
                                    .AddSingleton<HttpClient>(new HttpClient())
                                    .AddSingleton<FabricClient>(new
FabricClient())

.AddSingleton<StatelessServiceContext>(serviceContext))
                            .UseContentRoot(Directory.GetCurrentDirectory())
                            .UseStartup<Startup>()
                            .UseServiceFabricIntegration(listener,
ServiceFabricIntegrationOptions.None)
                            .UseUrls(url)
                            .Build();
                    }))
            };
        }

        internal static Uri GetVotingDataServiceName(ServiceContext context)
        {
            return new
Uri($"{context.CodePackageActivationContext.ApplicationName}/VotingData");
        }
    }
}
```

2.5 Add the VotesController.cs file

- Add a controller, which defines voting actions. Right-click on the Controllers folder, then select Add->New item->Visual C#->ASP.NET Core->Class. Name the file VotesController.cs, then click Add.

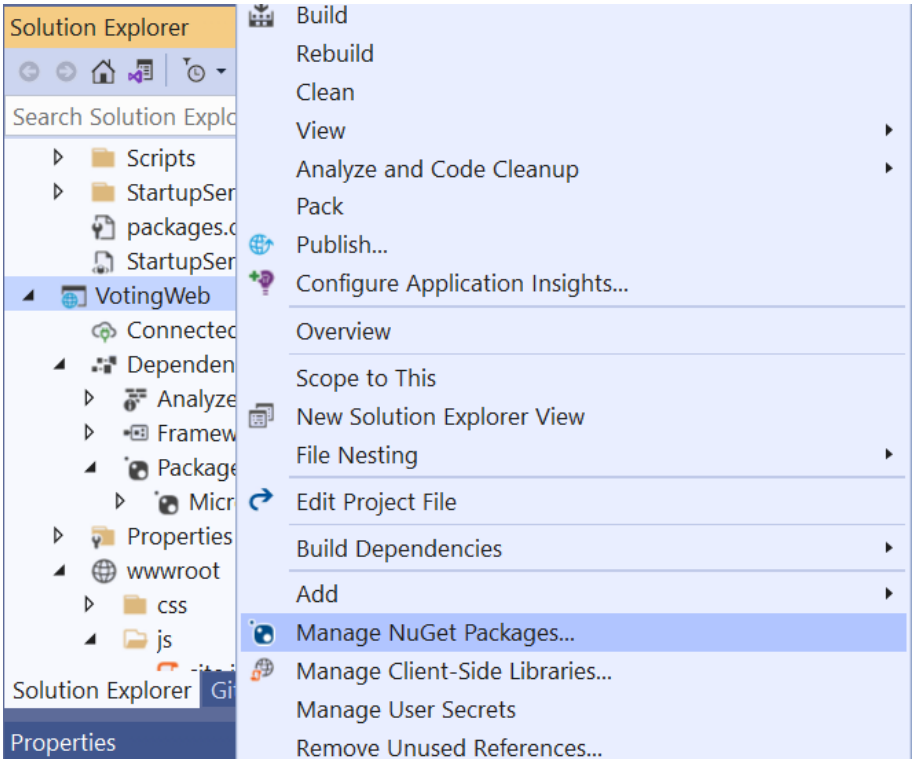


- Replace the VotesController.cs file contents with the following, then save your changes this file is modified to read and write voting data from the back-end service.

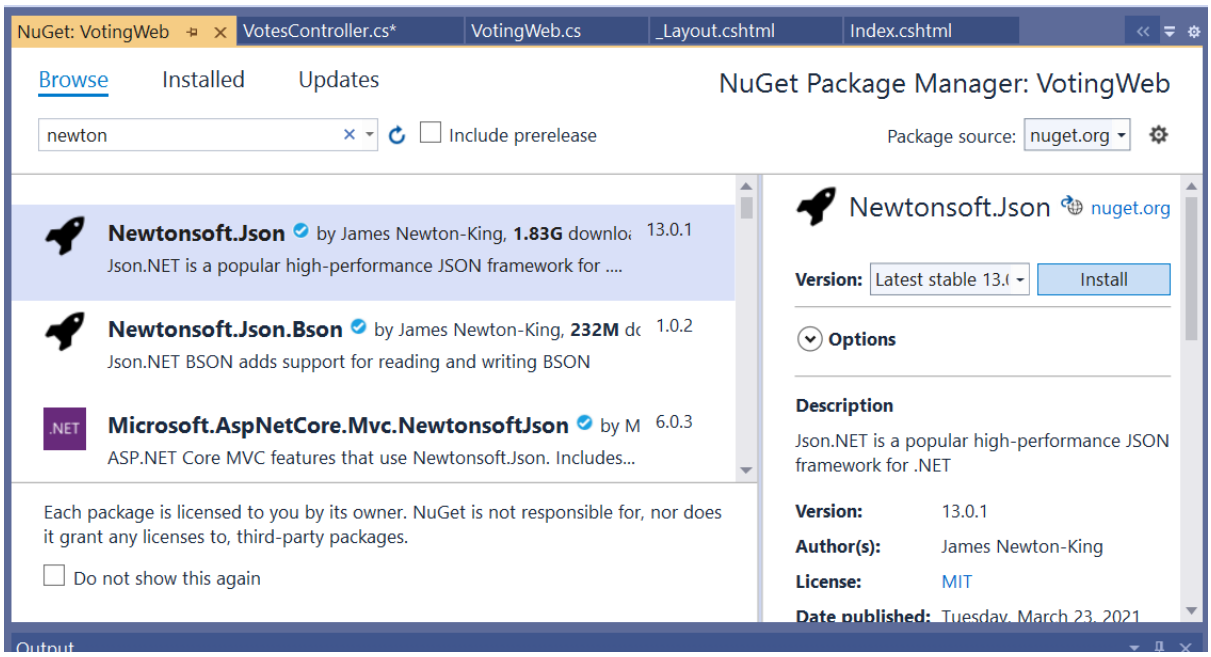
```
namespace VotingWeb.Controllers
{
    using System;
    using System.Collections.Generic;
    using System.Fabric;
    using System.Fabric.Query;
    using System.Linq;
    using System.Net.Http;
    using System.Net.Http.Headers;
    using System.Text;
    using System.Threading.Tasks;
    using Microsoft.AspNetCore.Mvc;
    using Newtonsoft.Json;

    [Produces("application/json")]
    [Route("api/Votes")]
    public class VotesController : Controller
    {
        private readonly HttpClient httpClient;

        public VotesController(HttpClient httpClient)
        {
            this.httpClient = httpClient;
        }

        // GET: api/Votes
        [HttpGet]
        public async Task<IActionResult> Get()
        {
```

```
            List<KeyValuePair<string, int>> votes = new List<KeyValuePair<string,
int>>();
            votes.Add(new KeyValuePair<string, int>("Pizza", 3));
            votes.Add(new KeyValuePair<string, int>("Ice cream", 4));
            return Json(votes);
        }
    }
}
```

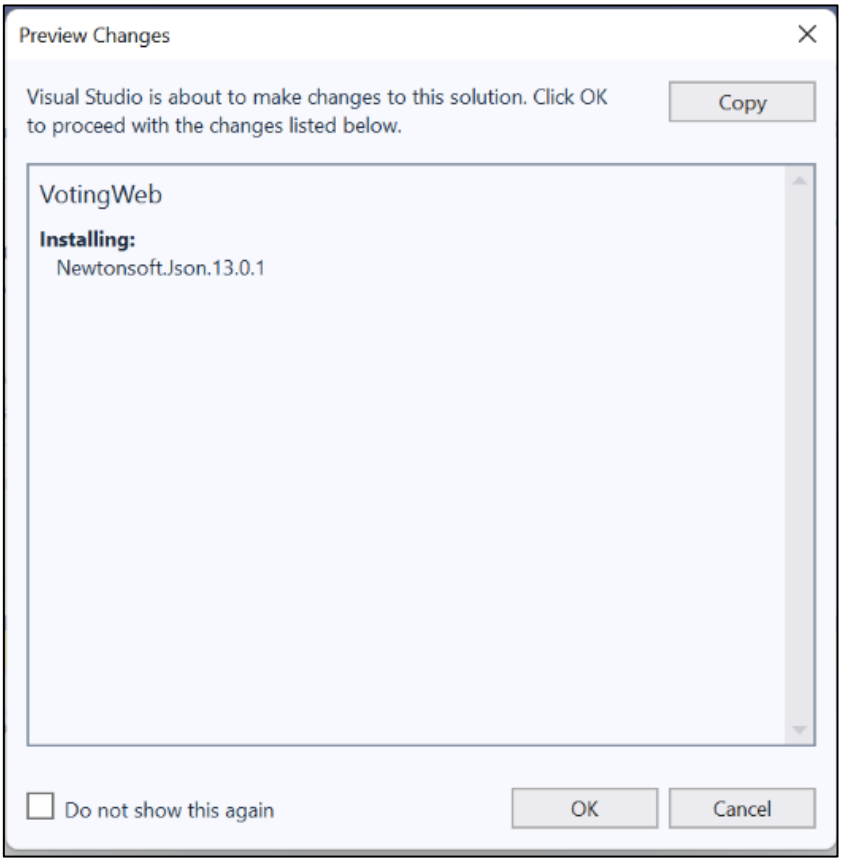
- Now to install the required *Newtonsoft.Json* NuGet package Right click on VotingWeb > Manage NuGet Packages



- Click Browse and search for Newtonsoft.Json Now select Newtonsoft.Json package and click install.



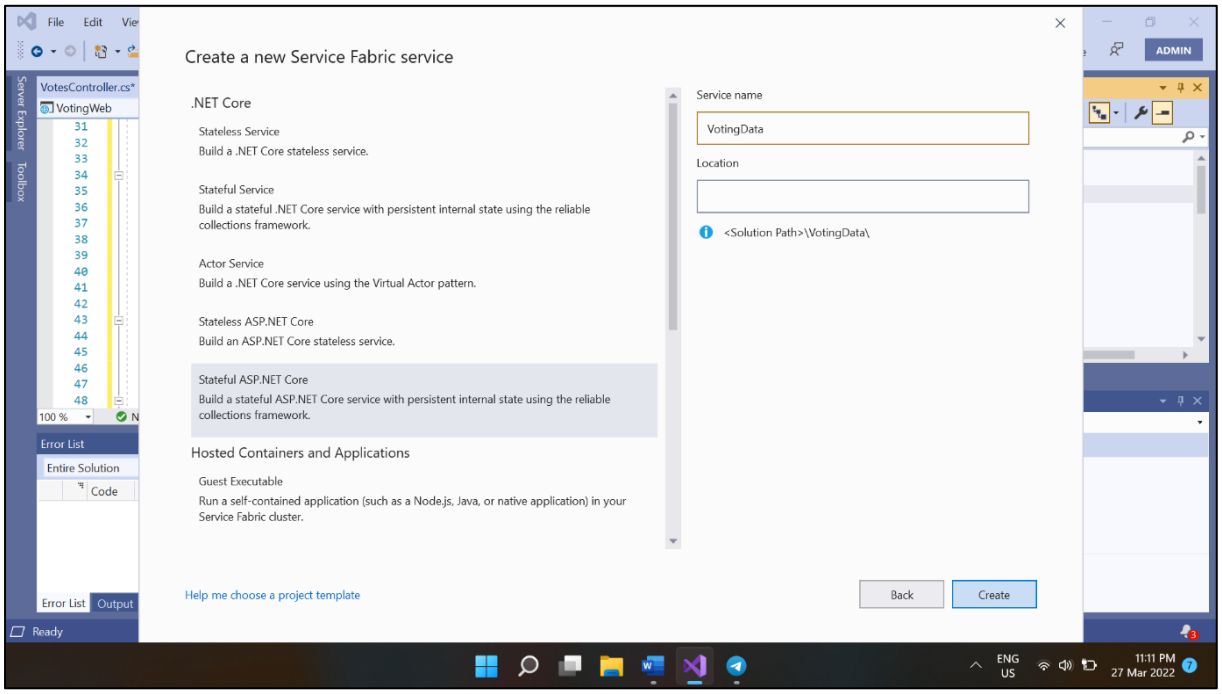
- Now It will ask you permission for solution changes. Click OK



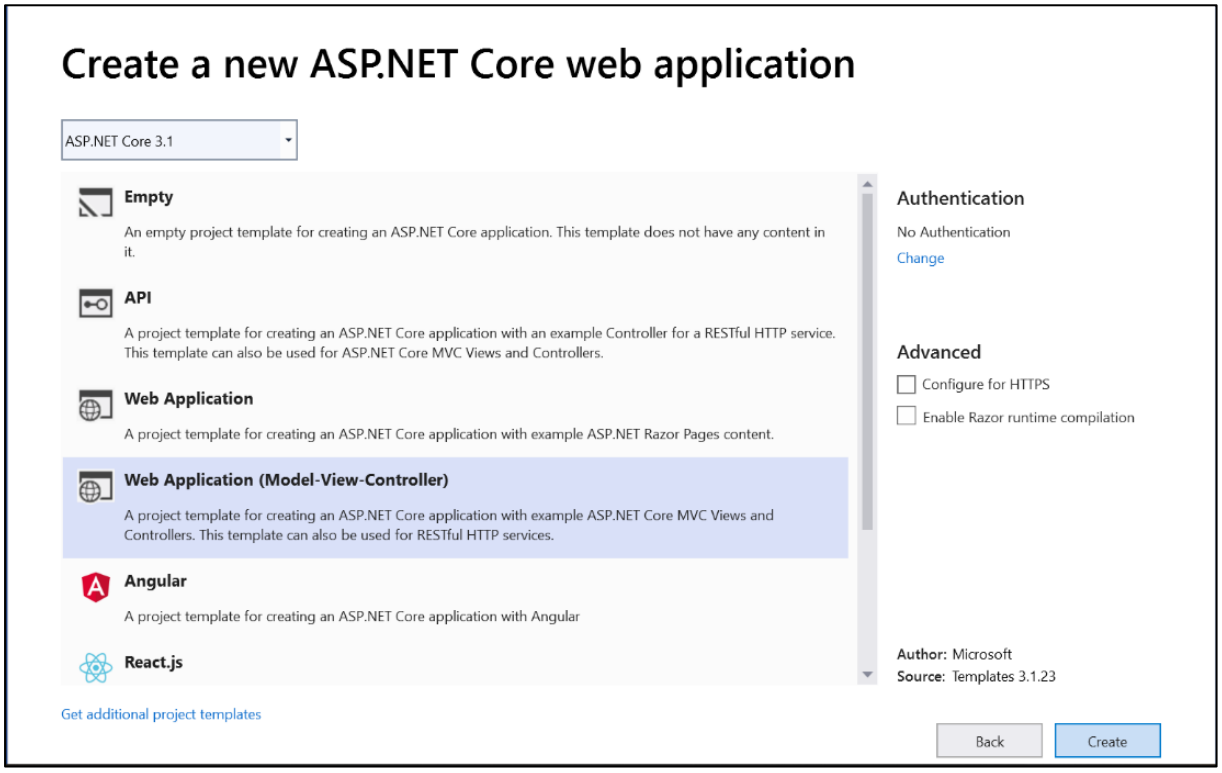
Step 3: Add a stateful back-end service to your application

- In Solution Explorer, right-click Services within the Voting application project and choose Add -> New Service Fabric Service...

- In the New Service Fabric Service dialog, choose Stateful ASP.NET Core, name the service VotingData, then press Create.
- Once your service project is created, you have two services in your application. As you continue to build your application, you can add more services in the same way. Each can be independently versioned and upgraded.
- The next page provides a set of ASP.NET Core project templates. choose API.
- Visual Studio creates the VotingData service project and displays it in Solution Explorer.

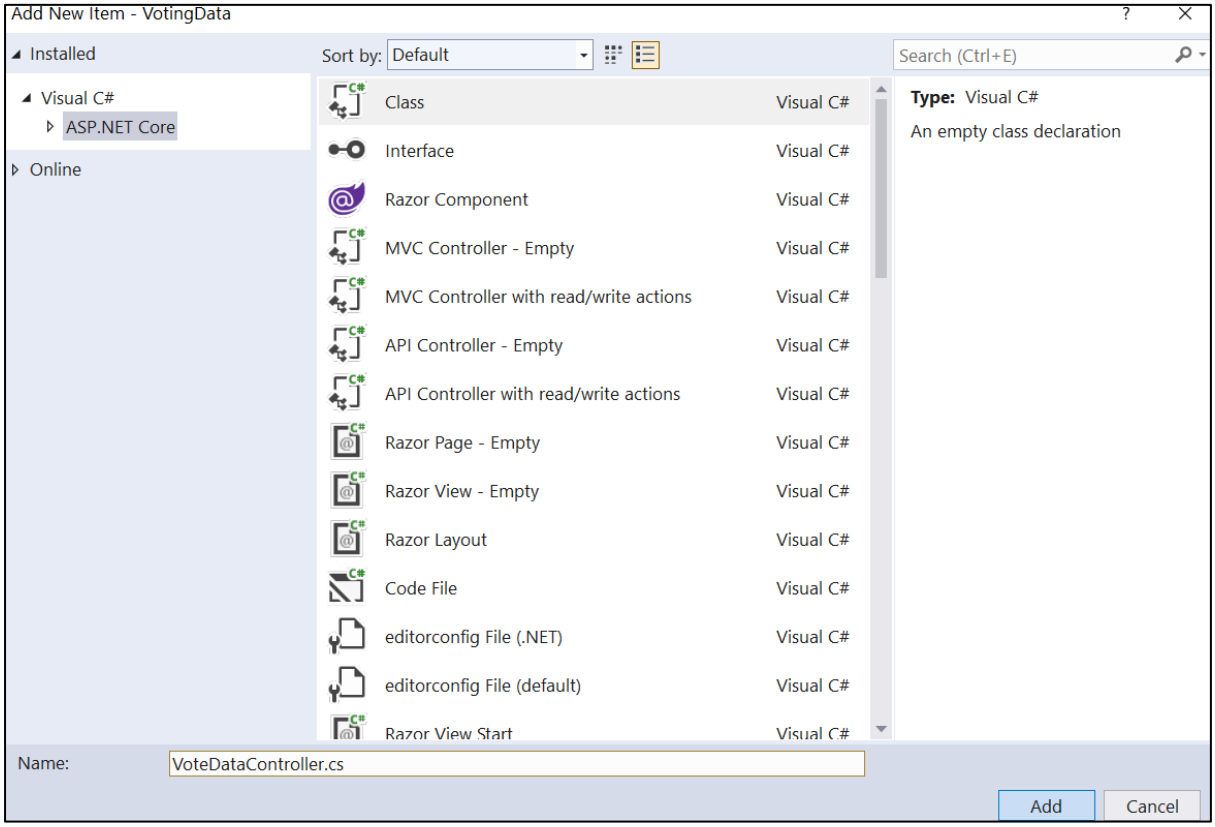


- The next page provides a set of ASP.NET Core project templates. For this practical, choose Web Application (Model-View-Controller), then click Create.



3.1 Add the VoteDataController.cs file

- In the VotingData project, right-click on the Controllers folder, then select Add->New item->Class. Name the file VoteDataController.cs and click Add.



- Replace the file contents with the following, then save your changes.

```
namespace VotingData.Controllers
{
    using System.Collections.Generic;
    using System.Threading;
    using System.Threading.Tasks;
    using Microsoft.AspNetCore.Mvc;
    using Microsoft.ServiceFabric.Data;
    using Microsoft.ServiceFabric.Data.Collections;

    [Route("api/[controller]")]
    public class VoteDataController : Controller
    {
        private readonly IReliableStateManager stateManager;

        public VoteDataController(IReliableStateManager stateManager)
        {
            this.stateManager = stateManager;
        }

        // GET api/VoteData
        [HttpGet]
        public async Task<IActionResult> Get()
        {
            CancellationToken ct = new CancellationToken();

            IReliableDictionary<string, int> votesDictionary =
                await this.stateManager.GetOrAddAsync<IReliableDictionary<string,
int>>("counts");

            using (ITransaction tx = this.stateManager.CreateTransaction())
            {
                var list = await votesDictionary.CreateEnumerableAsync(tx);

                var enumerator = list.GetAsyncEnumerator();

                List<KeyValuePair<string, int>> result = new
List<KeyValuePair<string, int>>();

                while (await enumerator.MoveNextAsync(ct))
                {
                    result.Add(enumerator.Current);
                }

                return this.Json(result);
            }
        }

        // PUT api/VoteData/name
        [HttpPut("{name}")]
        public async Task<IActionResult> Put(string name)
        {
            IReliableDictionary<string, int> votesDictionary = await
this.stateManager.GetOrAddAsync<IReliableDictionary<string, int>>("counts");

            using (ITransaction tx = this.stateManager.CreateTransaction())
            {
```

```
                    await votesDictionary.AddOrUpdateAsync(tx, name, 1, (key,
oldvalue) => oldvalue + 1);
                    await tx.CommitAsync();
                }

                return new OkResult();
            }

            // DELETE api/VoteData/name
            [HttpDelete("{name}")]
            public async Task<IActionResult> Delete(string name)
            {
                IReliableDictionary<string, int> votesDictionary = await
this.stateManager.GetOrAddAsync<IReliableDictionary<string, int>>("counts");

                using (ITransaction tx = this.stateManager.CreateTransaction())
                {
                    if (await votesDictionary.ContainsKeyAsync(tx, name))
                    {
                        await votesDictionary.TryRemoveAsync(tx, name);
                        await tx.CommitAsync();
                        return new OkResult();
                    }
                    else
                    {
                        return new NotFoundResult();
                    }
                }
            }
        }
}
```

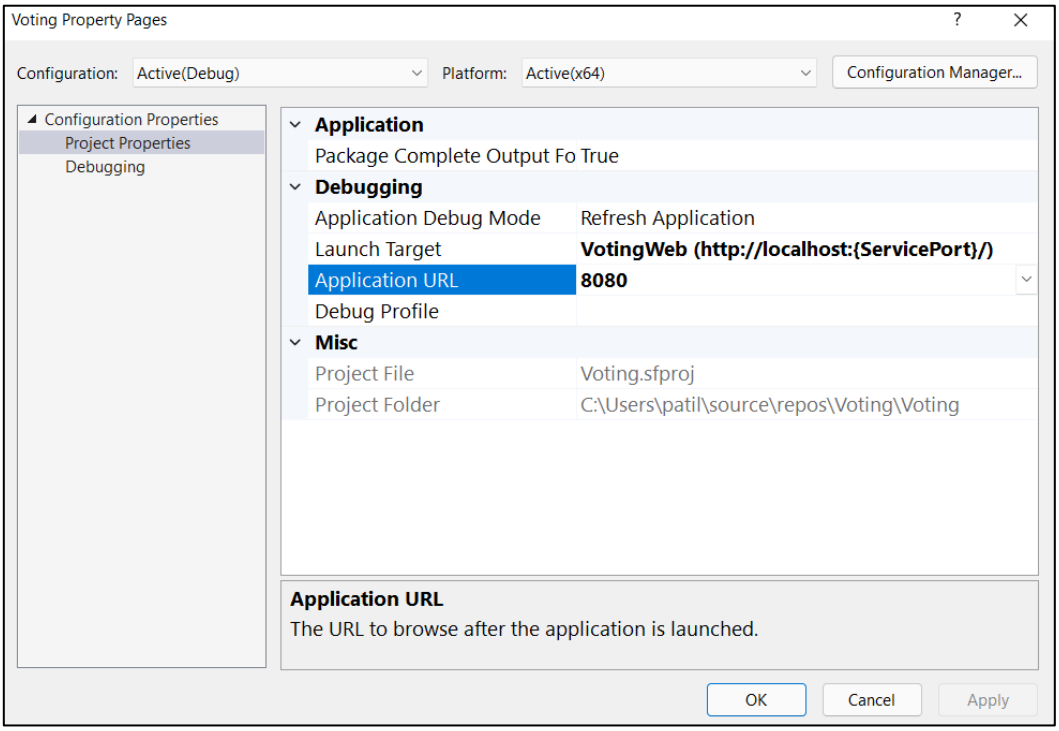
Step 4: Configure the listening port

- When the VotingWeb front-end service is created, Visual Studio randomly selects a port for the service to listen on. The VotingWeb service acts as the front-end for this application and accepts external traffic, so let's bind that service to a fixed and well-known port. The service manifest declares the service endpoints.

- In Solution Explorer, open VotingWeb/PackageRoot/ServiceManifest.xml. Find the Endpoint element in the Resources section and change the Port value to 8080. To deploy and run the application locally, the application listening port must be open and available on your computer.

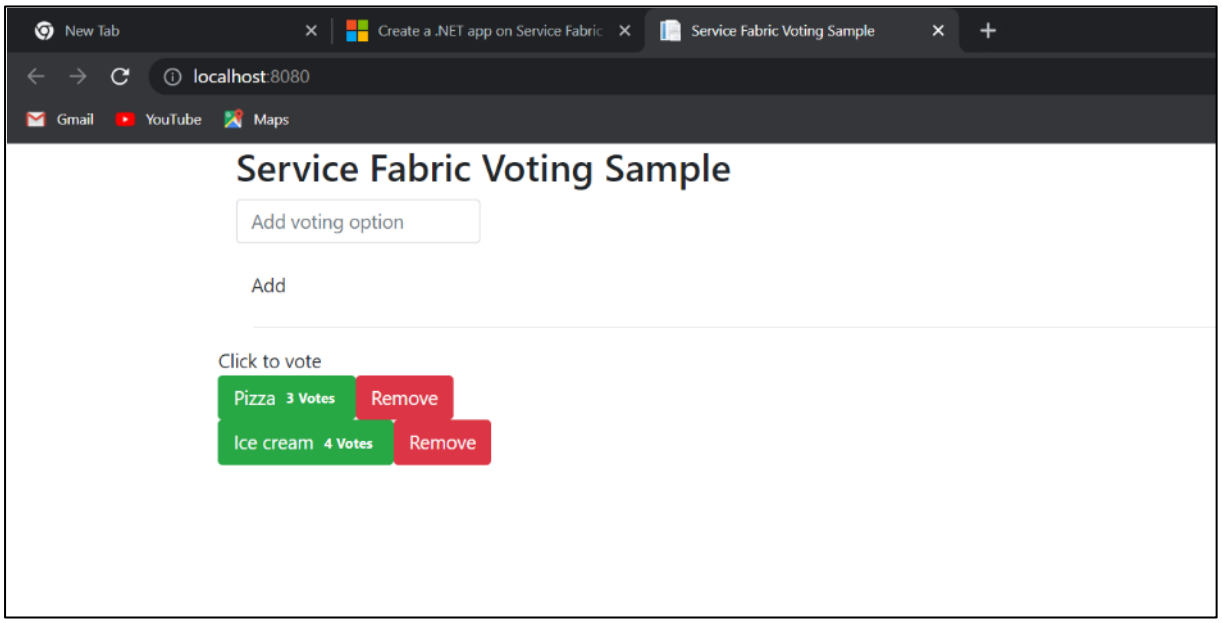
```
<Resources>
    <Endpoints>
      <!-- This endpoint is used by the communication listener to obtain the port
on which to
           listen. Please note that if your service is partitioned, this port is
shared with
           replicas of different partitions that are placed in your code. -->
      <Endpoint Protocol="http" Name="ServiceEndpoint" Type="Input" Port="8080" />
    </Endpoints>
  </Resources>
```

- Also, update the Application URL property value in the Voting project so a web browser opens to the correct port when you debug your application. In Solution Explorer, select the Voting project and update the Application URL property to 8080.



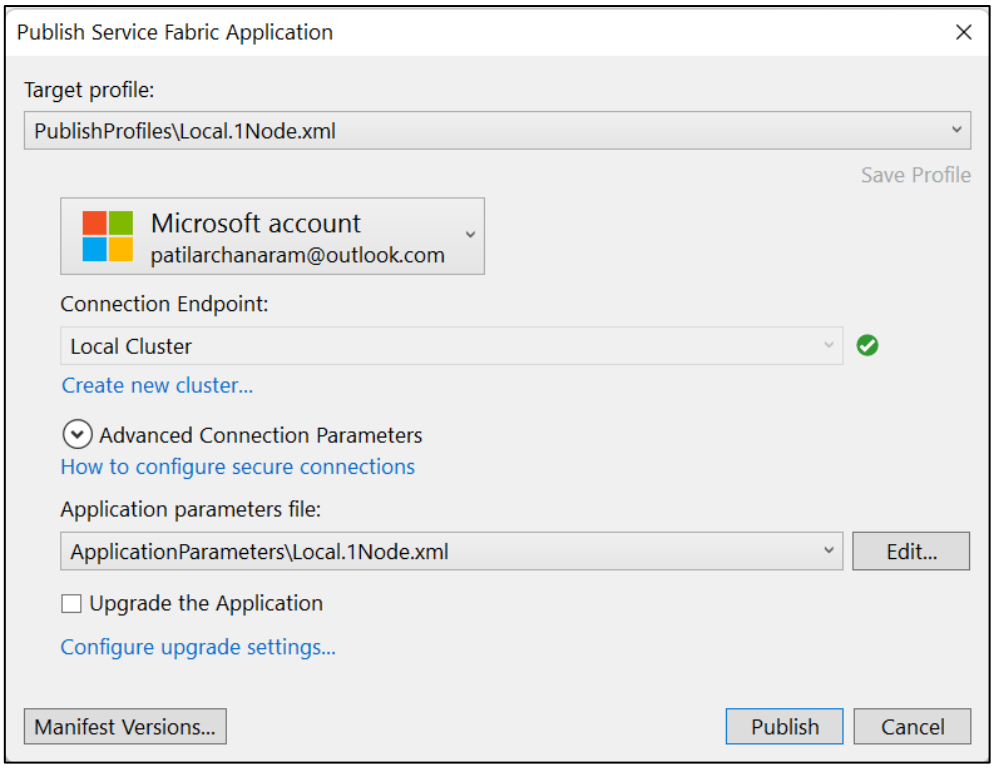
Deploy and run the Voting application locally

You can now go ahead and run the Voting application for debugging. In Visual Studio, press F5 to deploy the application to your local Service Fabric cluster in debug mode. The application will fail if you didn't previously open Visual Studio as an administrator.



Step 5: Publish Service Fabric application

- In the solution Explorer Right-click on the Voting and select Publish. The Publish dialog box appears.



- To open the service fabric explorer, open the Browser and paste the following IP address http://localhost:19080/