Ajay Kumar Kosuru

R Systems International

DEVOPS Training [SEZ Office]

[Microsoft AZURE]

**Azure DevOps:**

* Knowledge Transfer on build & deploy of GMAC application on Azure DevOps
* Knowledge Transfer in Site Core to publish GMAC in Azure DevOps test environment
* Knowledge Transfer on OCTOPUS CI/CD tool.
* Involved in test deployment of GMAC on site core web server.
* Started exploring on Octopus & Dynatrace DevOps tools.

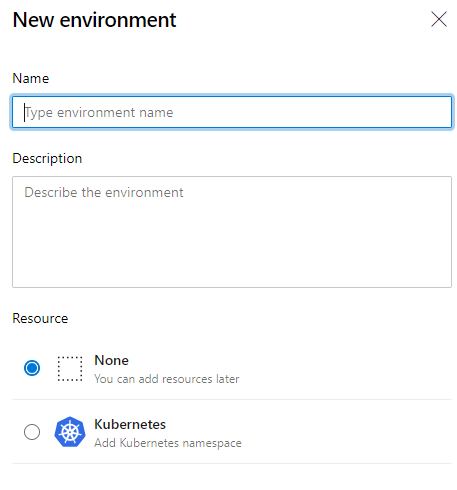
**.Net Core:**

* Created Web API in ASP.Net Core and hosted on IIS Local Server.
* Started publishing ASP.Net Core API on Azure Service Fabric.
* Started publishing ASP.NET Core API on OCTOPUS environment.
* Prerequisites:
* Install Azure Service fabric SDK with all dependencies for Visual Studio.
* Web Platform Installer 5.1 is expected to be installed in the VS Framework.

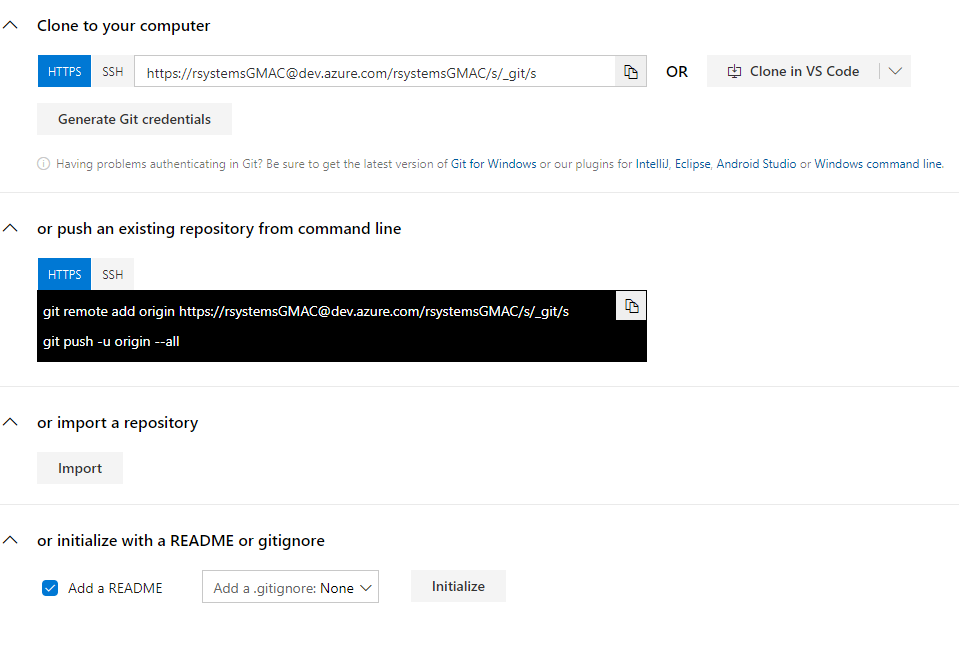
**Azure Service Fabric with Azure DevOps:**

1. Build & Release:

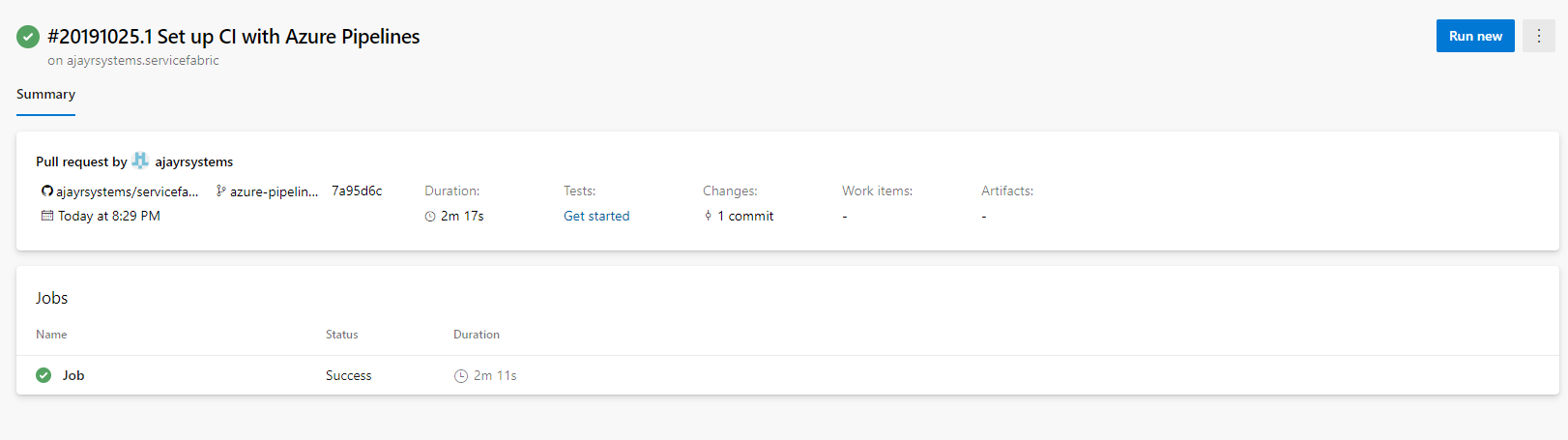
* Create Azure Service Fabric sample application in ASP.NET Core.
* Connect to Azure DevOps & create new environment to deploy application.



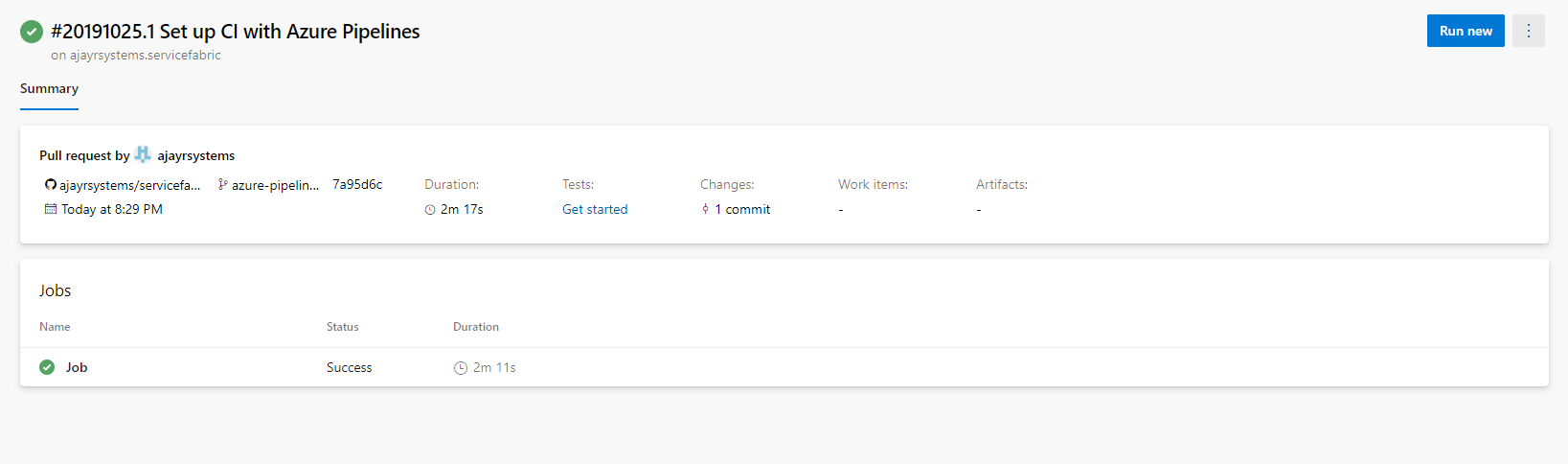
* Push sample code to GIT repository.
* Now start pushing the GIT source/master repository to Azure DevOps environment.



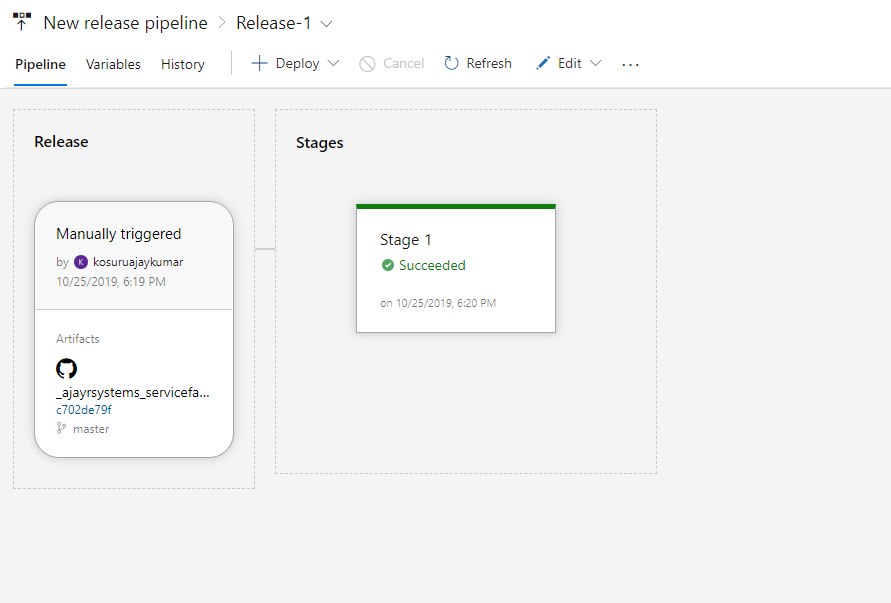
* Build the source code in Azure DevOps.
* After build went success, need to create a release pipeline for the code to push into production server or web server.



Build Success State:



* So, create a release pipeline for respective code in Azure DevOps.



* Once release pipeline created we need to configure the pipeline with the latest build.
* Then, start deployment using release pipeline for latest code in Azure DevOps.

1. Testing:

Just started exploring.

1. Artifacts:

Need to explore

**OCTOPUS:**

* Started exploring OCTOPUS CI/CD tool in automating deployments with Azure.
* Created Environment for test deployments.
* Choose the deployment target depending on the application on source environment like
* Windows
* Linux
* MAC
* AZURE
* KUBERNETES CLUSTER

**Azure DevOps to Octopus Deployment:**

* Create new Organization as “rsystems” and environments as Test, Dev & Prod in Azure DevOps
* Choose Dev environment first and create a new project as “Octopus Deployment”
* Open Repos and add the code by using clone GIT repository.
* Once code is available it’s time to build the code before pushing the code to octopus environment.
* There will be multiple ways to build the code

1. Azure Repos GIT (uses Azure pipeline)
2. Bitbucket Cloud
3. GitHub
4. GitHub Enterprise Server
5. Other GIT
6. Subversion
7. Use Classic Editor (Best way to go for)

* Let’s discuss on Classic Editor process,
* Choose classic editor, then select a source – Git repository which was already added to Azure DevOps then choose the branch as master.
* Choose Empty Job (Customized way to select the items)
* Add tasks to the job now one by one which were used to create Octopus Package and as well as helps in pushing the final package from Azure DevOps to Octopus Environment.

1. Task name: Use Nuget

* Task version: 1
* Display Name: Use Nuget

1. Task name: Nuget restore

* Display Name: Nuget restore
* Command: restore
* Path to solution: \*\*/\*.sln
* Feeds to use: Feed(s) | select here: Enabled
* Check Use packages from Nuget.org

1. Task name: Build solution

* Display name: Build solution
* Solution: GMAC.sln
* Visual Studio Version: Latest
* MSBuild Arguments:

/p:DeployOnBuild=true /p:WebPublishMethod=Package /p:PackageAsSingleFile=true /p:SkipInvalidConfigurations=true  /p:DesktopBuildPackageLocation="$(build.artifactstagingdirectory)\drop.$(productversionnumber).$(build.buildid).zip"  /p:DeployIisAppPath="Default Web Site"

* Clean: Enabled

1. Task name: Publish Artifact

* Task Version: 1
* Display name: Publish Artifact
* Path to publish: GMAC.sln (choose project solution file)
* Artifact name: $(Parameters.ArtifactName)
* Artifact publish location; Azure pipelines

1. Task name: Package application for octopus

* Task Version: 4
* Display name: Octopack
* Package ID: drop
* Package Format: ZIP
* Package Version: $(productversionnumber)-$(packversion)
* Source Path: $(build.artifactstagingdirectory)
* Output Path: $(build.artifactstagingdirectory)

1. Push package to octopus

* Task Version: 4
* Display name: Octopus package push
* Octopus Deploy Server: choose from dropdown
* Space; Choose from dropdown
* Package: $(build.artifactstagingdirectory)\drop.$(productversionnumber)-$(packversion).zip
* Overwrite Mode: Ignore if exists
* Now save & run the pipeline.
* If everything went fine your build will generate octopus package and that package will be automatically pushed to your octopus environment.

**Azure Learning:**

1. Virtual Networks:

* Started learning how to create virtual networks and their configuration with respect to network gateways with respect to on premises and off premises environments.
* Steps to be done in achieving above task
* Create a new virtual network as “RsystemsOnPremises”
* Create a virtual network gateway for the above VPN as “On-premises”
* Assign a new public IP address for the gateway as “OnPremisesPublicIP”
* Create a new virtual network as “RsystemsOffPremises”
* Create a virtual network gateway for the above VPN as “Off Premises”
* Assign a new public IP address for the gateway as “OffPremisesPublicIP”
* Now it’s time to establish a connection between On-premises VPN and Off Premises VPN using Azure Connection Vnet-To-Vnet (a network access between two virtual networks).

**Sonar Cloud:**

* Publishing a Web Application from Azure DevOps to Sonar Cloud:
* Login to your Sonar Cloud Account
* Create new Organization as “rsystems”
* Create new project as “Octopus”
* Login to Azure DevOps account
* Create new project as “OcctopusDeploy”
* Now open project, go to project setting option in the bottom of the screen and create a new service connection by providing the following details.
* Connection Name: Unique name
* Sonar Cloud Token: \*\*\*\*\*\*\*\*\*\*\*
* For Sonar cloud token, go to Sonar Cloud portal open My Account, go to Security and use Generate Tokens option to generate a Sonar Cloud Token.
* Now, create a new pipeline as discussed below,
* Create an empty job in pipeline
* Add Use Nuget 4.4.1
* Display Name: Use NuGet 4.4.1
* Use NuGet 4.4.4
* Version of NuGet.exe to install
* 4.4.1
* Add NuGet Restore
* Display Name: NuGet restore
* Use NuGet 4.4.4
* Command
* Restore
* Path to solution, package.config. or project.json
* \*\*\\*.sln
* Use packages from NuGet.org
* Enable
* Add Prepare Analysis on Sonar Cloud
* Display Name: Prepare Analysis on Sonar Cloud
* SonarCloud Service Endpoint
* SonarCloudConnection
* Organization
* Select your organization
* Choose the way to run the analysis
* Integrate with MSBuild
* Project Key
* Use your Project Key
* Project Name
* User your Project name
* Project Version
* 1.0
* Add Build Solution
* Display Name: Build Solution
* Solution
* \*\*\\*.sln
* Visual Studio Version
* Latest
* MSBuild Arguments
* /p:DeployOnBuild=true /p:WebPublishMethod=Package /p:PackageAsSingleFile=true /p:SkipInvalidConfigurations=true /p:DesktopBuildPackageLocation="$(build.artifactstagingdirectory)\WebApp.zip" /p:DeployIisAppPath="Default Web Site"
* Platform
* $(BuildPlatform)
* Configuration
* $(BuildConfiguration)
* Add Run Code Analysis
* Display Name: Run Code Analysis
* Add Publish Quality Gate Result
* Display name: Publish Quality Gate Result
* Timeout: 300
* Now, save and run the pipeline and check the build in Sonar Cloud.
* As of the process the deployment from Azure DevOps to Sonar Cloud will gives a success result.