Agenda: Docker Registry

- Azure Container Registry
- Azure Container Instance

Azure Container Registry

- **Azure Container Registry allows you to build, store, and manage docker container images**
- Azure Container Registry is a managed Docker registry service based on the open-source Docker Registry 2.0. Create and maintain Azure container registries to store and manage your private Docker container images.
- Use container registries in Azure with your existing container development and deployment pipelines. Use Azure Container Registry Build (ACR Build) to build container images in Azure. Build on demand, or fully automate builds with source code commit and base image update build triggers.
- Pull images from an Azure container registry to various deployment targets:
  - **Scalable orchestration systems** that manage containerized applications across clusters of hosts, including Kubernetes, DC/OS, and Docker Swarm.
  - **Azure services** that support building and running applications at scale, including Azure Kubernetes Service (AKS), App Service, Batch, Service Fabric, and others.

Container Registry SKUs

- **Basic:** A cost-optimized entry point for developers learning about Container Registry. Basic registries have the same programmatic capabilities as Standard and Premium (Azure Active Directory authentication integration, image deletion, and web hooks), however, there are size and usage constraints.
- **Standard:** The Standard registry offers the same capabilities as Basic, but with increased storage limits and image throughput. Standard registries should satisfy the needs of most production scenarios.
- **Premium:** Premium registries have higher limits on constraints, such as storage and concurrent operations, including enhanced storage capabilities to support high-volume scenarios. In addition to higher image throughput capacity, Premium adds features like geo-replication for managing a single registry across multiple regions, maintaining a network-close registry to each deployment.

Step 1: Update the Code

1. Create a New Project - HelloWebApp

   dotnet new mvc -n HelloWebApp

   cd HelloWebApp

2. Add **Dockerfile** as below Dockerfile

```
FROM mcr.microsoft.com/dotnet/core/sdk:3.1 AS build
WORKDIR /app

# copy csproj and restore as distinct layers
COPY *.csproj ./
RUN dotnet restore

# copy everything else and build app
COPY . ./
WORKDIR /app
RUN dotnet publish -c Release -o out

FROM mcr.microsoft.com/dotnet/core/aspnet:3.1 AS runtime
WORKDIR /app
COPY --from=build /app/out ./
ENTRYPOINT ["dotnet", "HelloWebApp.dll"]
```

3. Build images

   docker build -t sandeepsoni/hellowebapp .

4. Run the docker image locally

   docker run --rm -p 8080:80 sandeepsoni/hellowebapp

Step2: Create Container Registry using Portal

5. Create a resource → Containers → **Azure Container Registry.**

6. Under **Admin user**, select **Enable**. Take note of the following values:
   - Login server
   - Username
   - password

7. Login to ACR

   docker login --**username** dssdemo --**password** X3bI/uVbrNJ8lgfLXqjDV4zQVWRJgOI1 **dssdemo.azurecr.io**

8. Tag Local Images

   docker image **tag** sandeepsoni/hellowebapp **dssdemo1.azurecr.io**/hellowebapp:v1

9. Push images to ACR

   docker **push** dssdemo.azurecr.io/hellowebapp:v1

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

10. From another machine we can login to Azure Docker Registry (Step 7)

    docker login --**username** dssdemo --**password** 9vGPunhg4B0ZHDL20ifV=TuxrOUKM9RE **dssdemo.azurecr.io**

11. You can now download and execute the application locally

    docker **pull** dssdemo.azurecr.io/hellowebapp:v1

    docker **run** --rm -p 8080:80 sandeepsoni/hellowebapp:v1

Azure Container Instances

- Azure Container Instances enables exposing your containers directly to the internet with an IP address and a fully qualified domain name (FQDN). When you create a container instance, you can specify a custom DNS name label so your application is reachable at <**customlabel**>.<**azureregion**>.azurecontainer.io.
- Azure Container Instances guarantees your application is as isolated in a container as it would be in a VM.
- Azure Container Instances enables deployment of container instances into an Azure virtual network. By deploying container instances into a subnet within your virtual network, they can communicate securely with other resources in the virtual network, including those that are on premises (through VPN gateway or ExpressRoute).

Container Instance Features:

- **Fast startup times:** Containers offer significant startup benefits over virtual machines. Container Instances can start containers in Azure in seconds, without the need to provision and manage VMs.
- **Public IP connectivity and DNS name:** Container Instances enables exposing your containers directly to the internet with an IP address and a fully qualified domain name (FQDN). When you create a container instance, you can specify a custom DNS name label so your application is reachable at customlabel.azureregion.azurecontainer.io.
- **Hypervisor-level security:** Historically, containers have offered application dependency isolation and resource governance but have not been considered sufficiently hardened for hostile multi-tenant usage. Container Instances guarantees your application is as isolated in a container as it would be in a VM.
- **Custom sizes:** As demand for resources increases, the nodes of an AKS cluster can be scaled out to match. If resource demand drops, nodes can be removed by scaling in the cluster.
- For compute-intensive jobs such as machine learning, Container Instances can schedule Linux containers to use **NVIDIA Tesla GPU resources**.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

- **Persistent storage:** To retrieve and persist state with Container Instances, we offer direct mounting of Azure Files shares.
- **Linux and Windows containers:** Container Instances can schedule both Windows and Linux containers with the same API. Simply specify the OS type when you create your container groups. Some features are currently restricted to Linux containers.
- Container Instances supports Windows images based on Long-Term Servicing Channel (LTSC) versions. Windows Semi-Annual Channel (SAC) releases like 1709 and 1803 are unsupported.
- **Co-scheduled groups:** Container Instances supports scheduling of multi-container groups that share a host machine, local network, storage, and lifecycle. This enables you to combine your main application container with other supporting role containers, such as logging sidecars.
- **Virtual network deployment:** Container Instances enables deployment of container instances into an Azure virtual network. By deploying container instances into a subnet within your virtual network, they can communicate securely with other resources in the virtual network, including those that are on premises (through VPN gateway or ExpressRoute).

Step 4: Create Container Instance for HelloWebApp:

12. Select Create a Resource → Containers → Container Instance → + Add
13. In Basic blade

    Container name = "HelloWebAppInstance",

    Container Image type="Private",

    Container image="dssdemo.azurecr.io/hellowebapp:v2"

    Copy Image registry login server, username and password from Azure Container Registry created before.
14. In Configuration blade

    OS Type= Windows, . . ., DNS name label = "hellowebapp"
15. OK

16. Copy and paste the address http://hellowebapp.eastus.azurecontainer.io in browser.

Using CLI:

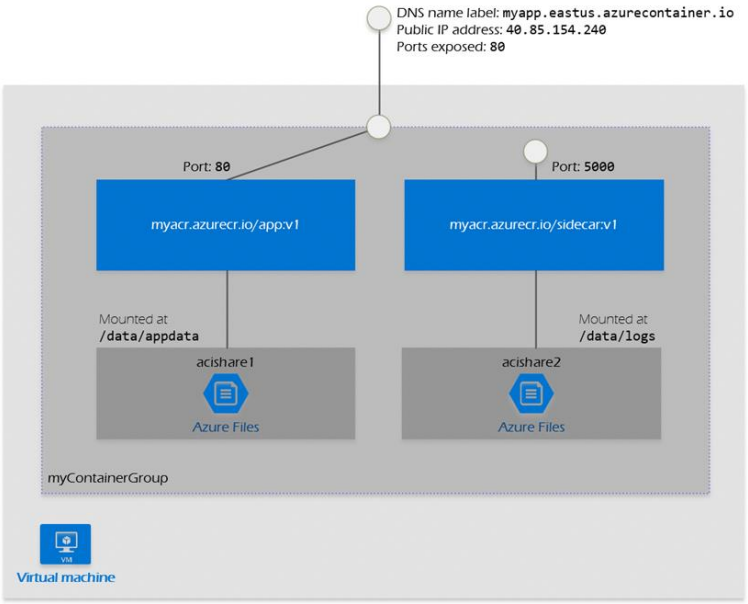
```
# Get name of container registry login server
az acr show --name <acrName> --query loginServer


# Get container registry password
az acr credential show --name <acrName> --query "passwords[0].value"
```

```
# Deploy container
az container create --resource-group myResourceGroup --name aci-tutorial-app --image <acrLoginServer>/aci-tutorial-app:v1 --cpu 1 --memory 1 --registry-login-server <acrLoginServer> --registry-username <acrName> --registry-password <acrPassword> --dns-name-label <aciDnsLabel> --ports 80
```

Azure Container Groups

- A container group is a collection of containers that get scheduled on the same host machine. The containers in a container group share a lifecycle, resources, local network, and storage volumes. It's similar in concept to a *pod* in Kubernetes.
- A collection of containers that get scheduled on the same host.
- The containers in the group share a lifecycle, resources, local network, and storage volumes.



This example container group:

- Is scheduled on a single host machine.
- Is assigned a DNS name label.
- Exposes a single public IP address, with one exposed port.
- Consists of two containers. One container listens on port 80, while the other listens on port 5000.
- Includes two Azure file shares as volume mounts, and each container mounts one of the shares locally.

There are two common ways to deploy a multi-container group: use a Resource Manager template or a YAML file. A Resource Manager template is recommended when you need to deploy additional Azure service resources (for

example, an Azure Files share) when you deploy the container instances. Due to the YAML format's more concise nature, a YAML file is recommended when your deployment includes only container instances.

Common scenarios

Multi-container groups are useful in cases where you want to divide a single functional task into a small number of container images. These images can then be delivered by different teams and have separate resource requirements.

Example usage could include:

- A container serving a web application and a container pulling the latest content from source control.
- An application container and a logging container. The logging container collects the logs and metrics output by the main application and writes them to long-term storage.
- An application container and a monitoring container. The monitoring container periodically makes a request to the application to ensure that it's running and responding correctly, and raises an alert if it's not.
- A front-end container and a back-end container. The front end might serve a web application, with the back end running a service to retrieve data.