AZ-203 Developer Training

Deploying and Managing Containers

Agenda

- What are Docker containers?
- How can we deploy them to Azure?
- How can we run containers locally?
- How can we create and publish container images?
- What are our options for hosting in Azure?
- Installing Docker for Windows
- Using Docker Commands
- Dockerfile basics
- Building images
- Tagging images
- Pushing images to a registry
- Create an Azure Container Registry using Azure CLI
- Azure Container Instances (ACI)
- Azure Web App for Containers
- Azure Service Fabric
- Orchestrators
- Kubernetes basics
- Azure Kubernetes Service (AKS)

Docker Basics, Upgrades, Networks and More

Docker Basics

Images

Image = application + dependencies Layered

Built from a dockerfile

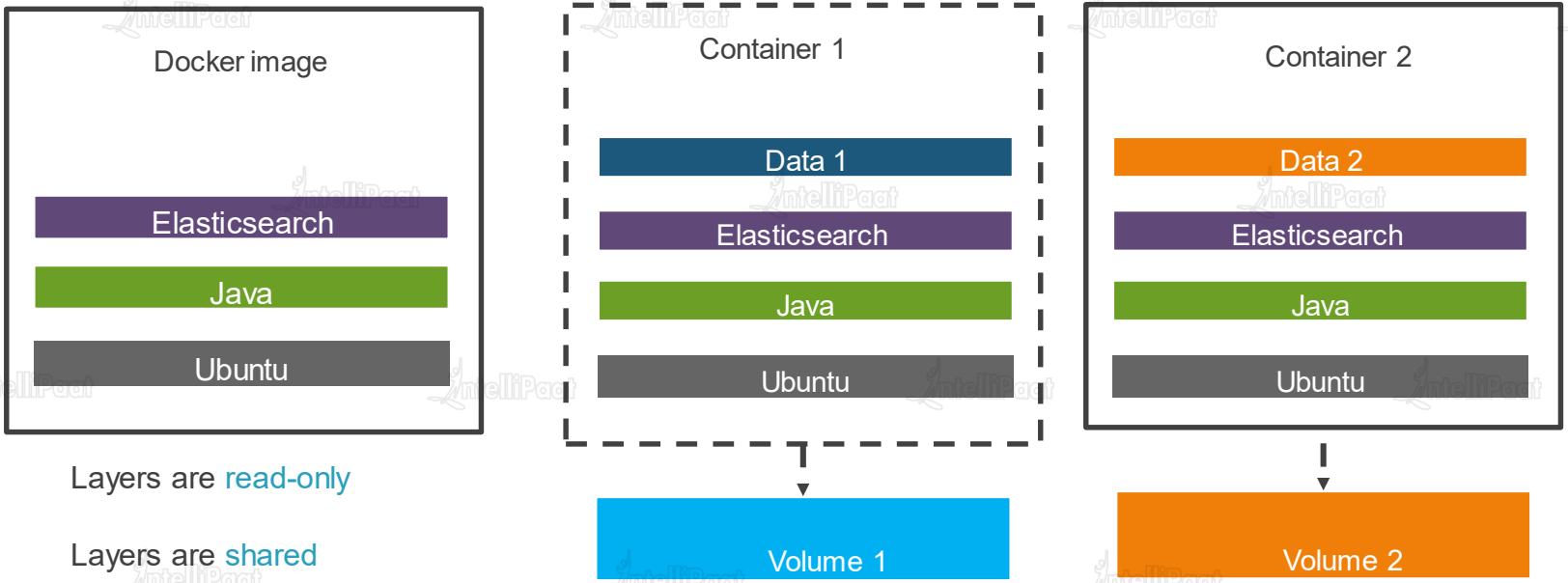
Tagged - e.g. markheath/myapp:1.4 Publish to a "container registry"

- Docker Hub
- Azure Container Registry

Containers

- An instance based on an image
- Run on a "Docker host"
  - Works the same everywhere
  - Provides memory and CPU
  - Publish ports
  - Disk access (image only)
- Can be stopped and restarted
- Multiple containers from a single image

Data Storage



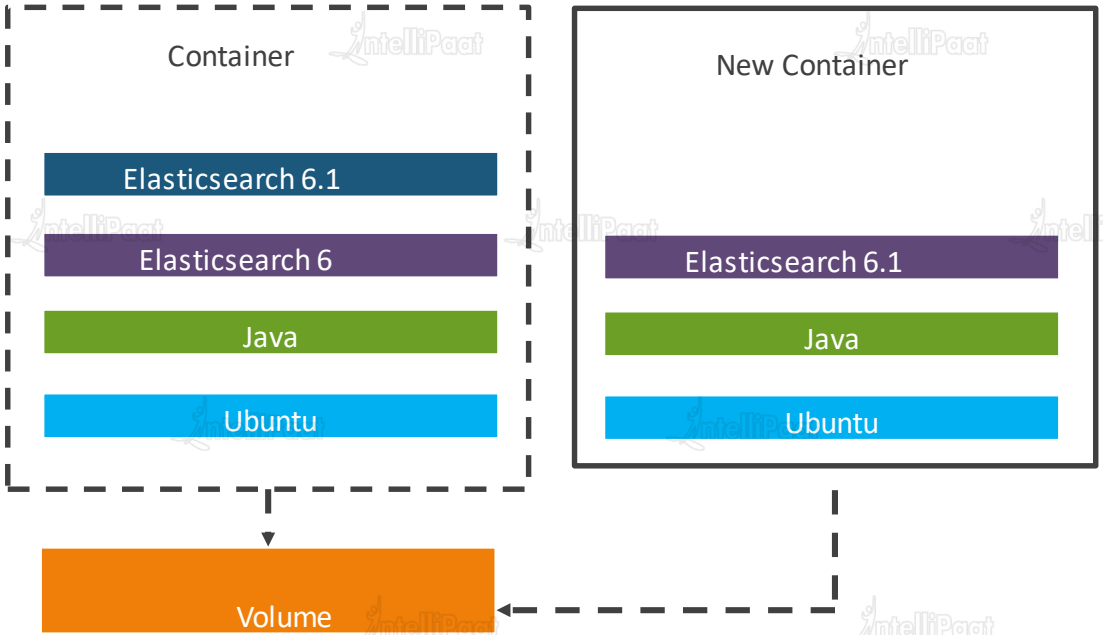
Layers are read-only

Layers are shared

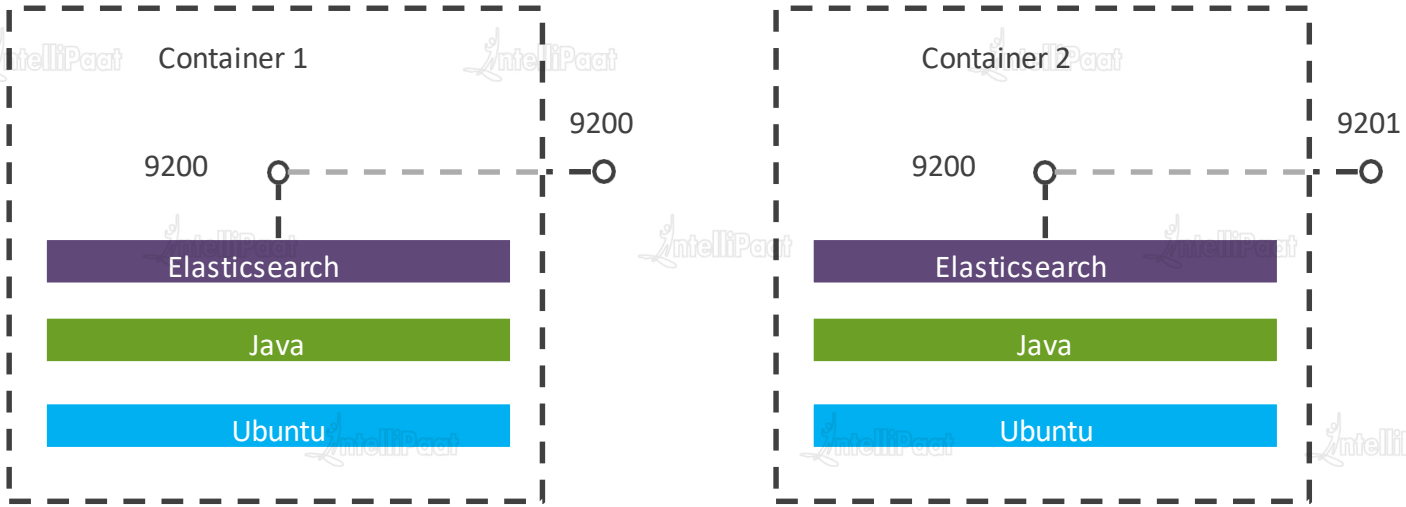
Use volumes to persist data

Mount volumes to a path in the container – e.g. /var/lib/mysql

Upgrades



Network



Containers can publish a port

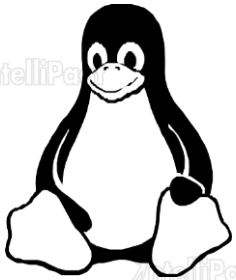
Docker hosts can map a port

Environment Variables

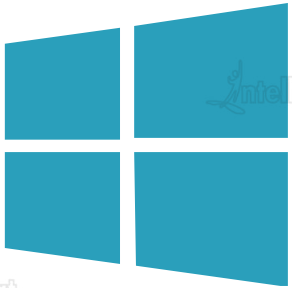
Containers have their own environment variables

Linux/Windows

Linux



Windows



- Windows Server 2016 supports Windows containers
- Linux containers on Windows (LCOW)

Docker Benefits

Docker Benefits

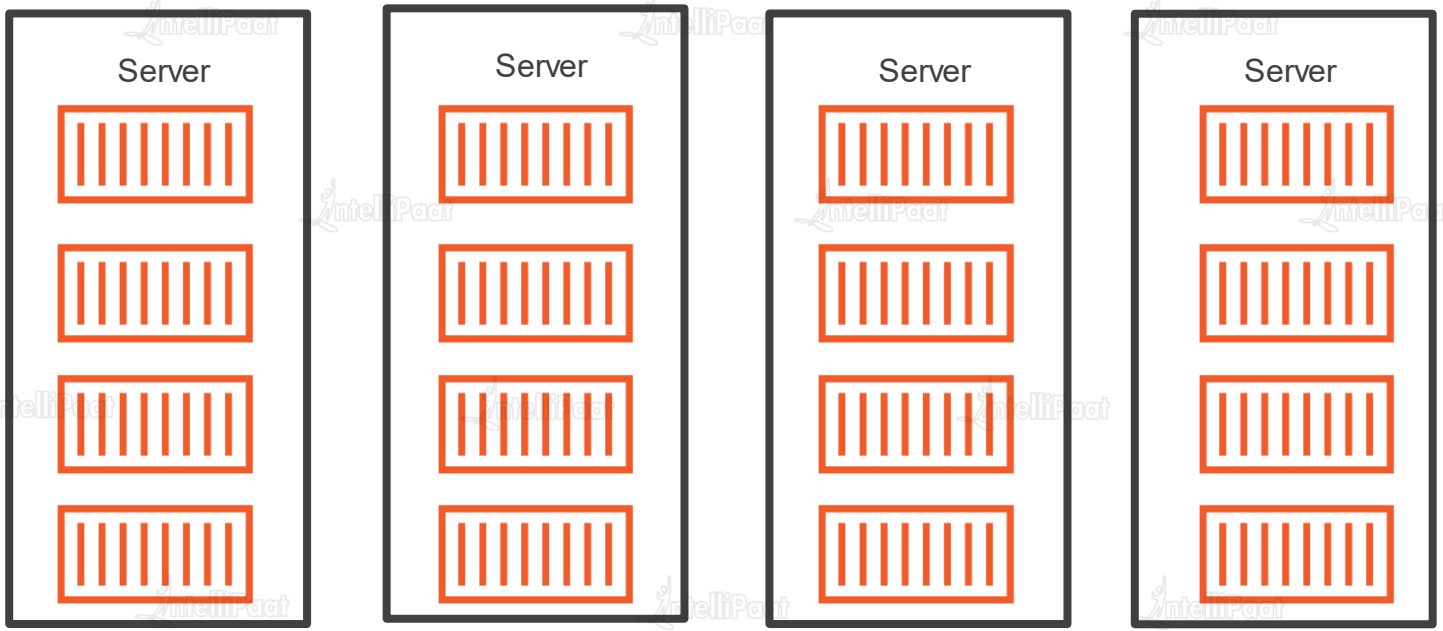
Containers	VMs
✓ Isolation	✓ Strong isolation
✓ Portability	✓ Portability
✓ Efficient	✗ Resource hungry
✓ Fast start	✗ OS boot times
✓ Disposable	✗ Require patching
✓ Minimal attack surface area	✗ OS needs hardening

Security

- Isolation
- Minimal attack surface area
- Vulnerability scanning
- Image signing

Multiple Containers & Orchestrators

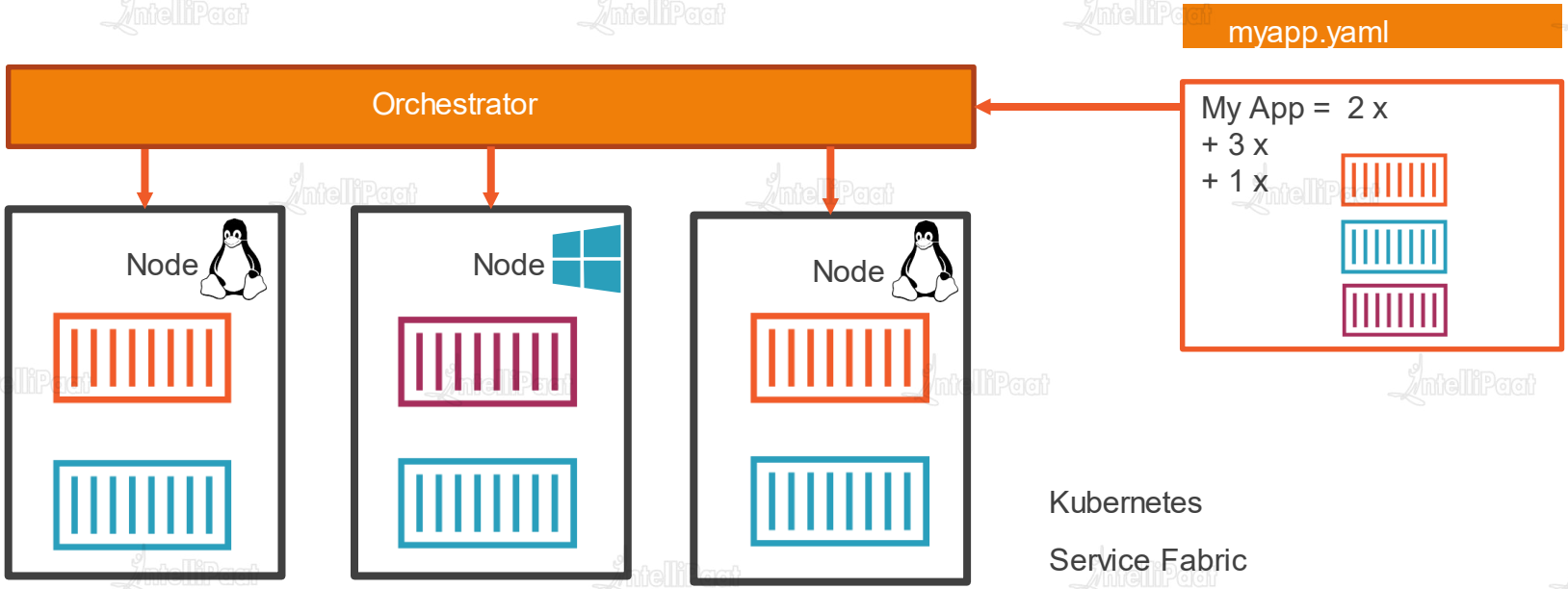
Managing Multiple Containers



Challenges:

- Deploying
- Scaling
- Monitoring
- Networking
- Upgrading
- Repairing

Orchestrators



Kubernetes

Service Fabric

Hosting Containers in Azure

Hosting Containers in Azure

Virtual Machines (IaaS)
Linux
Windows Server
Kubernetes

Azure Container Instances (ACI)
Serverless
Fast and easy
Per-second billing

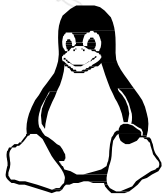
Azure Web Apps for Containers
Great for web apps
Custom domains
Auto-scaling

Azure Service Fabric "Mesh"
Scalable orchestration platform
Powers key Azure services
Multiple programming models

Azure Kubernetes Service (AKS)
Managed Kubernetes cluster
Just specify node count
Open source tooling

Installing Docker in Production

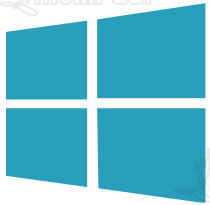
Installing Docker in Production



Alpine Linux

Ubuntu Core

Container Linux



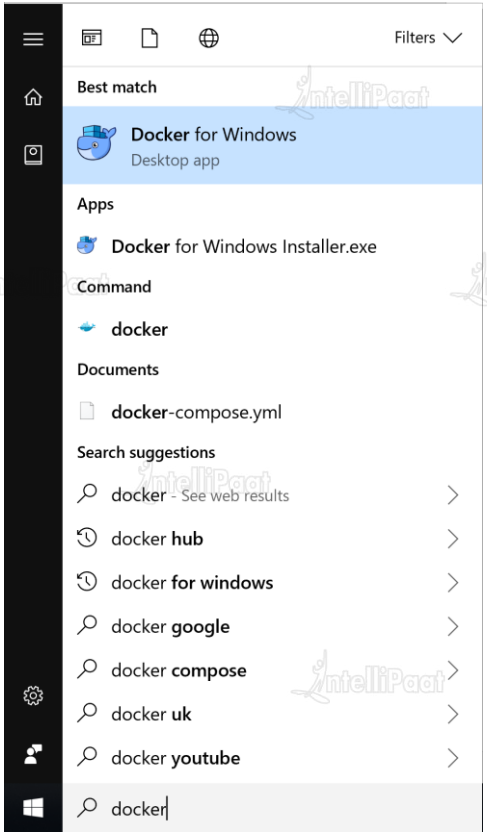
Windows Server 2016

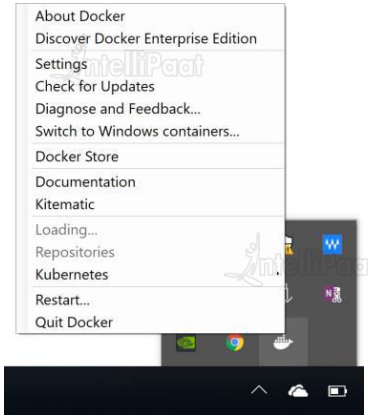
Windows Server 2019

Installing Docker Locally

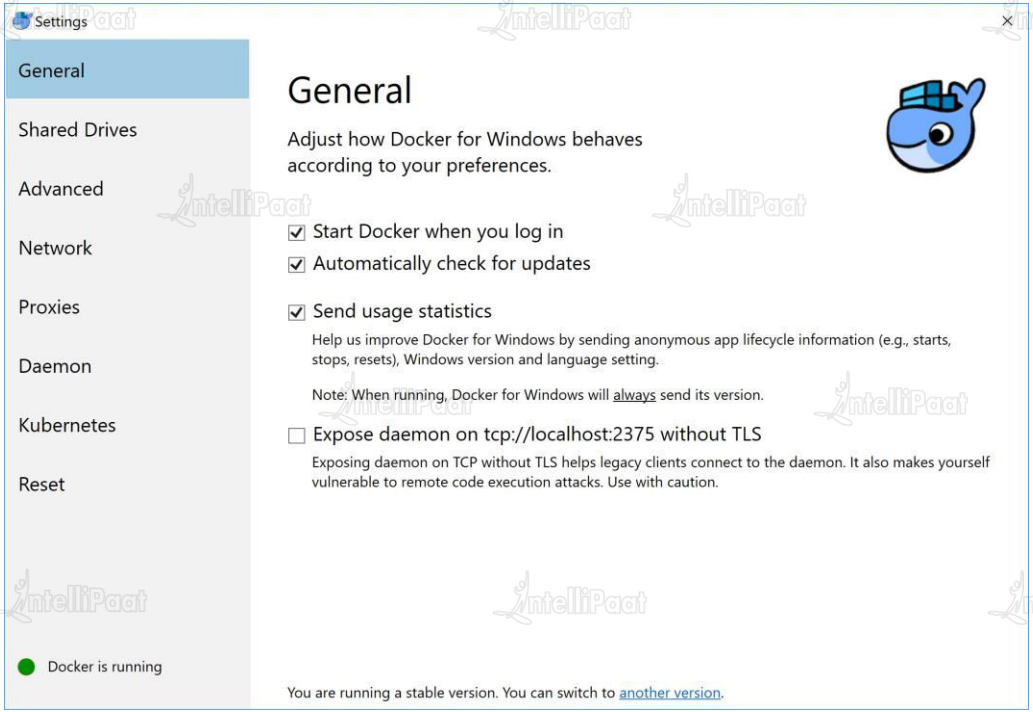
https://docs.docker.com/install/

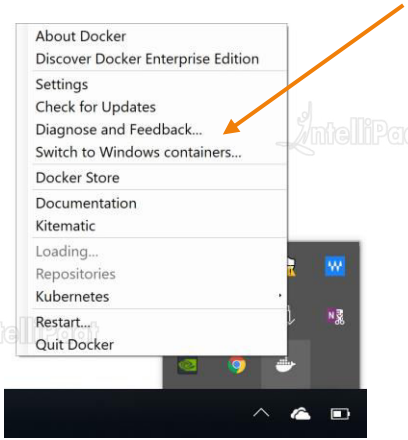
Installing Docker for Windows



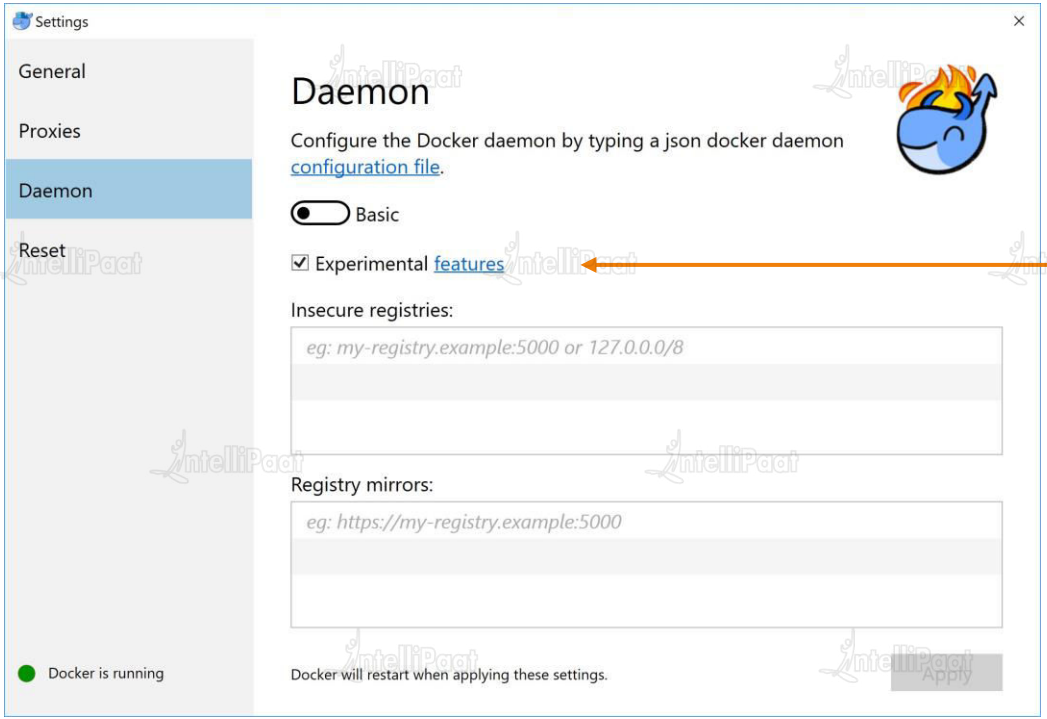


Continued….





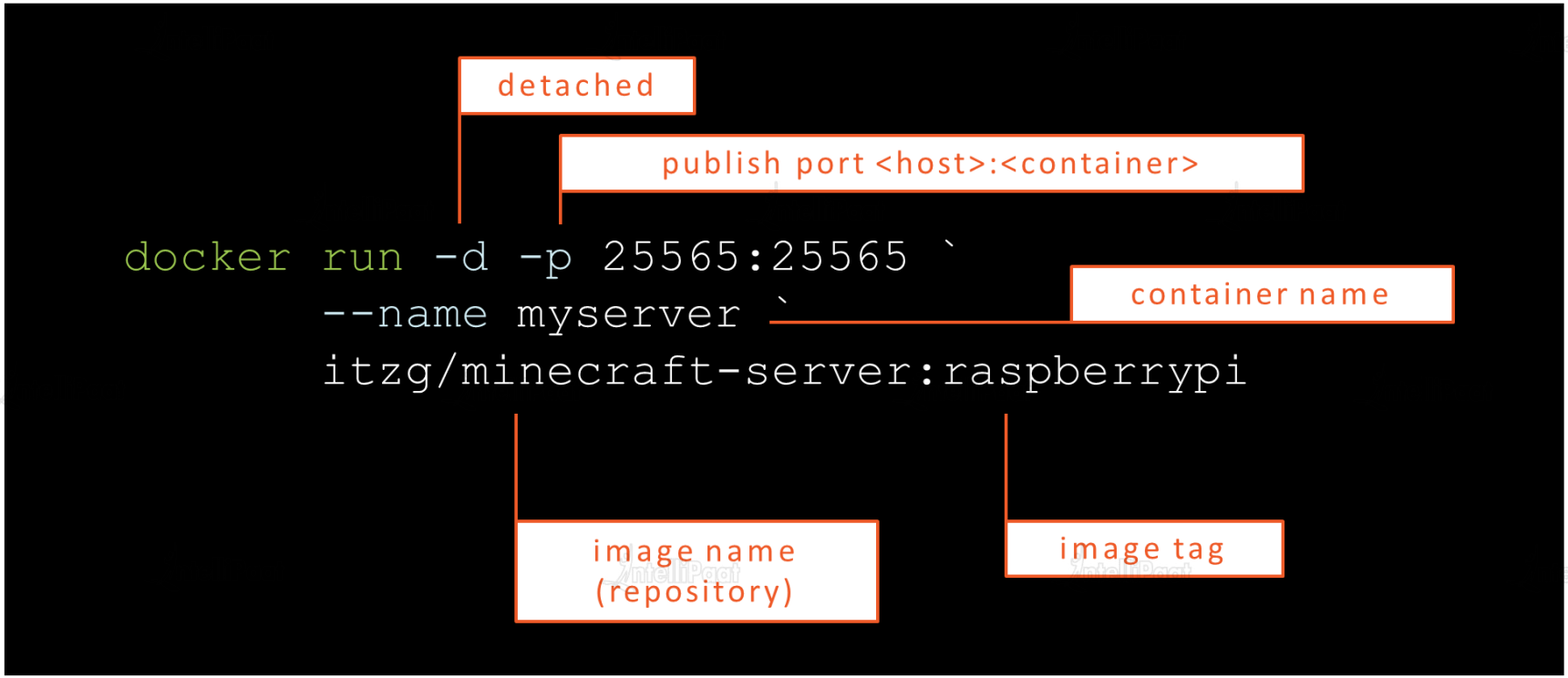
Continued….



--platform=linux

Docker Run

Docker Run



Hands-On

Hands-On

- **Hands-On 1**
  - Running Redis with Docker
- **Hands-On 2**
  - Running commands inside containers
  - Connect on localhost port 6379
  - Run the Redis CLI in a container
  - docker exec
- **Hands-On 3**
  - Storing container data
  - Containers are disposable
  - Keep data in Docker volumes
  - Mount a volume

Creating Docker Images with Dockerfiles

Creating Docker Images with Dockerfiles

Dockerfile

Contains instructions for how to create a Docker image

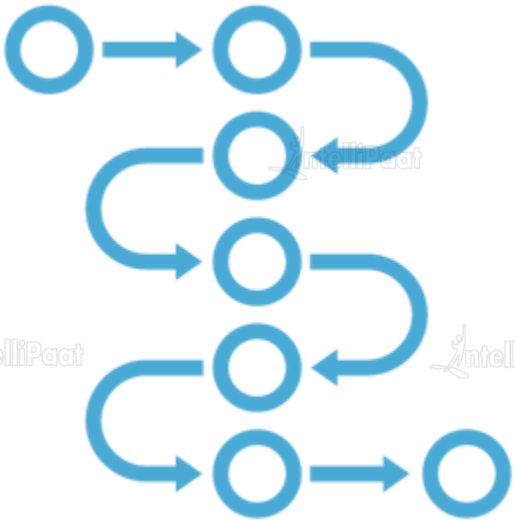
```
FROM microsoft/dotnet:aspnetcore-runtime
WORKDIR /app
COPY ./out .
ENTRYPOINT ["dotnet", "samplewebapp.dll"]
```

```
docker build -t samplewebapp.
```

Creating Dockerfiles

- Copy example dockerfiles
  - e.g. ASP.NET Core
- Visual Studio 2017
  - Right-click project > Add Docker support
- Yeoman
  - yo docker
- Azure Functions CLI
  - func init --docker

Multi-Stage Dockerfiles



- Single stage dockerfile
- Copies in a pre-built application
- Use Docker to build your application
- No developer SDKs needed locally
- Multi-stage dockerfile
- Stage 1: build the application in a container with the SDKs
- Stage 2: copy the published application into a container with the runtime

Commands for Build-Env

Commands for Build-Env

```
FROM microsoft/dotnet:sdk AS build-env   WORKDIR /app

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

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

# Build runtime  image
FROM microsoft/dotnet:aspnetcore-runtime   WORKDIR /app

COPY --from=build-env  /app/out .  ENTRYPOINT  ["dotnet", "samplewebapp.dll" ]
```

Hands-On

- Build an image with a multi-stage dockerfile

Azure Container Registry

What is Azure Container Registry?

Azure Container Registry allows you to **build**, **store**, and **manage** images for all types of container deployments. 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 Cases:

- Pull images from an Azure container registry to various deployment targets:
  - Scalable orchestration systems to manage containerized applications across clusters of hosts
  - Azure services that support building and running applications at scale.
- Developers can also **push** to a container registry as part of a **container development workflow**.
  - For example, target a container registry from a **continuous integration** and **deployment** tool such as **Azure DevOps Services** or **Jenkins**.

Azure Container Registry: Key Concepts

- **Registry**: Create one or more container registries in your Azure subscription
- **Repository**: A registry contains one or more repositories, which store groups of container images.
- **Image**: Stored in a repository, each image is a read-only snapshot of a Docker-compatible container
- **Container**: A container defines a software application and its dependencies wrapped in a complete filesystem including code, runtime, system tools, and libraries.

Create a private container registry using Azure CLI

- **Create a resource group**
- Create a resource group with the *az group* create command.
- An Azure resource group is a logical container into which Azure resources are deployed and managed.
- The following example creates a resource group named *myResourceGroup* in the *eastus location*.

```
az group create --name myResourceGroup  --location eastus
```

- **Create a container registry**
- Create an ACR instance using the az acr create command. The registry name must be unique within Azure, and contain 5-50 alphanumeric characters.
- In the following example, myContainerRegistry007 is used. Update this to a unique value.

```
az acr create --resource-group myResourceGroup  --name myContainerRegistry007  --sku Basic
```

Create a private container registry using Azure CLI

Continued……

- …When the registry is created, the output is similar to the following:


```
{  "adminUserEnabled": false,
"creationDate": "2019-01-08T22:32:13.175925+00:00",
"id": "/subscriptions/00000000-0000-0000-0000-
000000000000/resourceGroups/myResourceGroup/providers/Microsoft.ContainerRegistry/registri
es/myContainerRegistry007",
"location": "eastus",
"loginServer": "mycontainerregistry007.azurecr.io",
"name": "myContainerRegistry007",
"provisioningState": "Succeeded",
"resourceGroup": "myResourceGroup",
"sku": {
"name": "Basic",
"tier": "Basic"
},
"status": null,
"storageAccount": null,
"tags": {},
"type": "Microsoft.ContainerRegistry/registries"}
```

Create a private container registry using Azure CLI

Continued……

- **Log in to registry**
- Before pushing and pulling container images, you must log in to the registry. To do so, use the az acr login command.

```
az acr login --name <acrName>
```

- **Push Image to the registry**
- To push an image to an Azure Container registry, you must first have an image. If you don't yet have any local container images, run the following docker pull command to pull an existing image from Docker Hub. For this example, pull the *hello-world* image.

```
docker pull hello-world
```

Create a private container registry using Azure CLI

Continued……

- Before you can push an image to your registry, you must tag it with the fully qualified name of your ACR login server. The login server name is in the format *<registry-name>.azurecr.io* (all lowercase), for example, *mycontainerregistry007.azurecr.io*.
- Tag the image using the docker tag command. Replace *<acrLoginServer>* with the login server name of your ACR instance.

```
docker tag hello-world <acrLoginServer>/hello-world:v1
```

- Finally, use docker push to push the image to the ACR instance.

```
docker push <acrLoginServer>/hello-world:v1
```

- After pushing the image to your container registry, remove the *hello-world:v1* image from your local Docker environment.

```
docker rmi <acrLoginServer>/hello-world:v1
```

Create a private container registry using Azure CLI

Continued……

List Container Images

- The following example lists the repositories in your registry:

```
az acr repository list --name <acrName> --output table
```

Output:

```
Result
----------------
busybox
```

Create a private container registry using Azure CLI

Continued……

Run image from registry

- Now, you can pull and run the *hello-world:v1* container image from your container registry by using docker run:

```
docker run <acrLoginServer>/hello-world:v1
```

Output:

```
Unable to find image 'mycontainerregistry007.azurecr.io/hello-world:v1' locally
v1: Pulling from hello-world
Digest: sha256:662dd8e65ef7ccf13f417962c2f77567d3b132f12c95909de6c85ac3c326a345
Status: Downloaded newer image for mycontainerregistry007.azurecr.io/hello-world:v1

Hello from Docker!
This message shows that your installation appears to be working correctly.

[...]
```

- After this, clean up the resources using:
  - ***az group delete --name myResourceGroup***

Container Registry Security

Container Registry Security

- Images not accessible to the public
- Can restrict access rights Only allow trusted images
- Automatically update to use latest security updates

Hands-On

Hands-On

- Push a local Docker image to an Azure Container Registry

Hands-On

- **Create a private container registry using the Azure portal**
  - Create a container registry
  - Log in to registry
  - Push image to registry
  - List container images
  - Run image from registry
  - Clean up resources

Hint: You may perform the similar steps for Azure Portal which are used in CLI.

Azure Container Instances

What is Azure Container Instances?

Containers are becoming the preferred way to package, deploy, and manage cloud applications. Azure Container Instances offers the fastest and simplest way to run a container in Azure, without having to manage any virtual machines and without having to adopt a higher-level service.

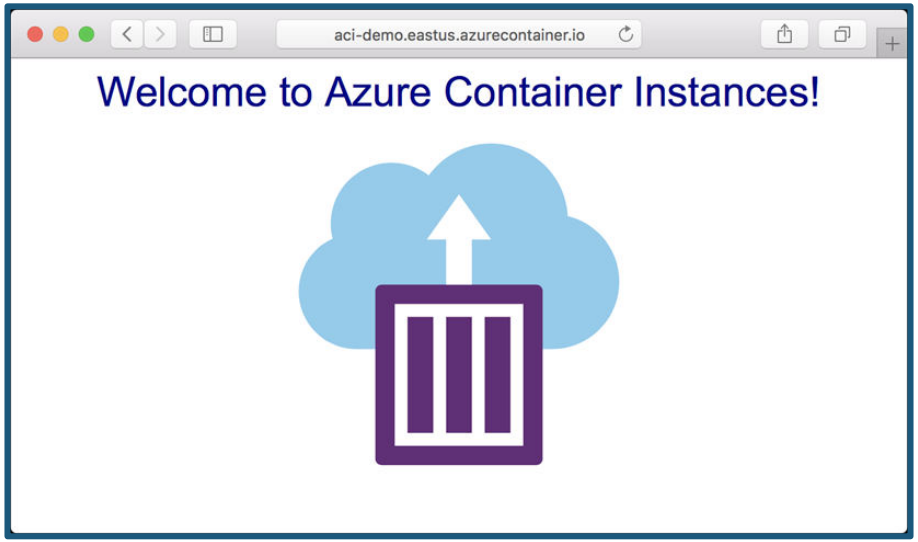
- **Benefits of Azure Container Instances**
  - Fast startup times
  - Public IP connectivity and DNS name
  - Hypervisor-level security
  - Custom sizes
  - Persistent storage
  - Linux and Windows containers
  - Co-scheduled groups
  - Virtual network deployment (preview)

Run a container application in

Azure Container Instances with Azure CLI

Use Azure Container Instances to run Docker containers in Azure with simplicity and speed. You don't need to deploy virtual machines or use a full container orchestration platform like Kubernetes.

Use the Azure CLI to create a container in Azure and make its application available with a fully qualified domain name (FQDN). A few seconds after you execute a single deployment command, you can browse to the running application



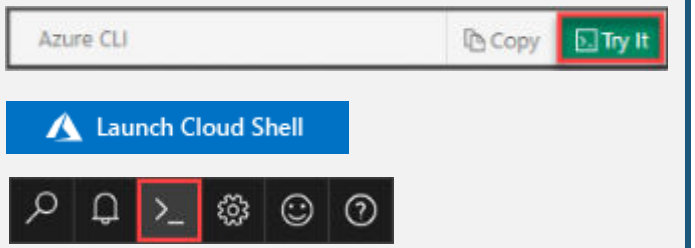
Run a container application in

Azure Container Instances with Azure CLI

Open the **Azure Cloud Shell**.

Azure Cloud Shell is a free, interactive shell that you can use to run the steps in this article. Common Azure tools are preinstalled and configured in Cloud Shell for you to use with your account. Just select the Copy button to copy the code, paste it in Cloud Shell, and then press Enter to run it. There are a few ways to open Cloud Shell:

- Select Try It in the upper-right corner of a code block.
- Open Cloud Shell in your browser.
- Select the Cloud Shell button on the menu in the upper-right corner of the Azure portal.



Run a container application in Azure Container Instances with Azure CLI

Create a resource group named *myResourceGroup* in the *eastus* location with the following az group create command:

```
az group create --name myResourceGroup --location eastus
```

Now create the **Container**.

Deploy a container with a DNS name label so that the web app is publicly reachable. Execute the following command to start a container instance.

```
az container create --resource-group myResourceGroup --name mycontainer --image microsoft/aci-helloworld --dns-name-label aci-demo --ports 80
```

Within a few seconds, you should get a response from the Azure CLI indicating that the deployment has completed. Check its status with the az container show command:

```
az container show --resource-group myResourceGroup --name mycontainer --query "{FQDN:ipAddress.fqdn,ProvisioningState:provisioningState}" --out table
```

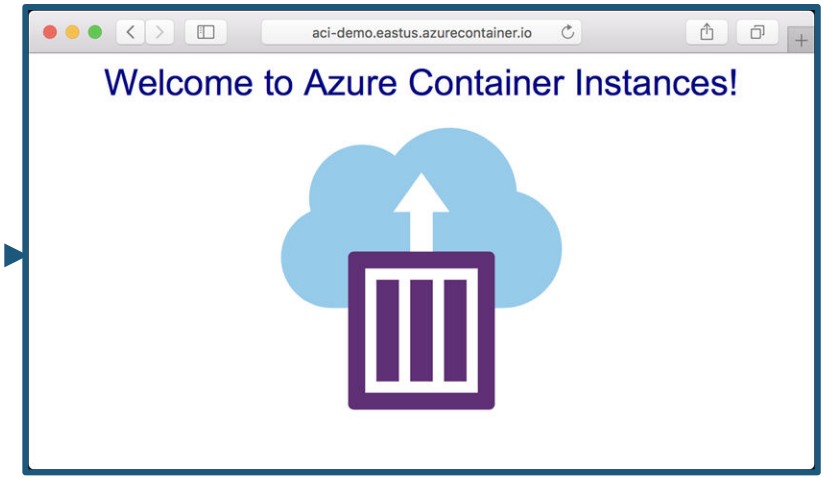
Run a container application in Azure Container Instances with Azure CLI

Continued……

- **Output**:

```
--------------------------------  ------------------
aci-demo.eastus.azurecontainer.io  Succeeded
```

- If the container's *ProvisioningState* is Succeeded, navigate to its FQDN in your browser.
- If you see a web page similar to the following, congratulations! You've successfully deployed an application running in a Docker container to Azure.



Run a container application in Azure Container Instances with Azure CLI

Continued……

Pull the container logs:

When you need to troubleshoot a container or the application it runs (or just see its output), start by viewing the container instance's logs.

Pull the container instance logs with the az container logs command:

```
az container logs --resource-group myResourceGroup --name mycontainer
```

Attach output streams:

In addition to viewing the logs, you can attach your local standard out and standard error streams to that of the container. First, execute the az container attach command to attach your local console to the container's output streams:

```
az container attach --resource-group myResourceGroup -n mycontainer
```

Run a container application in Azure Container Instances with Azure CLI

Continued……

- **Clean-Up the resources**:

When you're done with the container, remove it using the az container delete command:

```
az container delete --resource-group myResourceGroup --name mycontainer
```

To verify that the container has been deleted, execute the az container list command:

```
az container list --resource-group myResourceGroup --output table
```

The **mycontainer container should not appear in the command's output.*

If you're done with the myResourceGroup resource group and all the resources it contains, delete it with the az group delete command:

```
az group delete --name myResourceGroup
```

Creating Container Groups with the Azure CLI

```
az container create
-n mycontainergroup –g myresourcegroup
--image someimage:sometag
--ip-address public
--dns-name-label mysite # mysite.eastus.azurecontainer.io
--ports 80
--os-type Windows # default is Linux
--cpu  --memory 1.5
-e name=value
--restart-policy never # always, onfailure
--azure-file-volume…  # credentials, mount path, share-name
```

Hands-On

Hands-On

- **Run a container application in Azure Container Instances in the Azure portal**Create a container registry
  - Create a container instance
  - View container logs
  - Clean up resources

Hint: You may perform the similar steps for Azure Portal which are used in CLI.

Hands-On

- **Hands-On 1**
  - Create an ACI container group
    - Ghost blog
    - az group create
    - az container create
    - az container logs
    - az group delete
- **Hands-On 2**
  - Create a container group
    - Docker image from ACR
    - Mount an Azure file share
    - az container exec

App Service Web App for Containers

Web App for Containers

Now, you can easily deploy and run the container-based application on your Windows and Linux which scale according to your business.

Benefits of Using Web App for Containers

- A fully managed platform to perform infrastructure maintenance

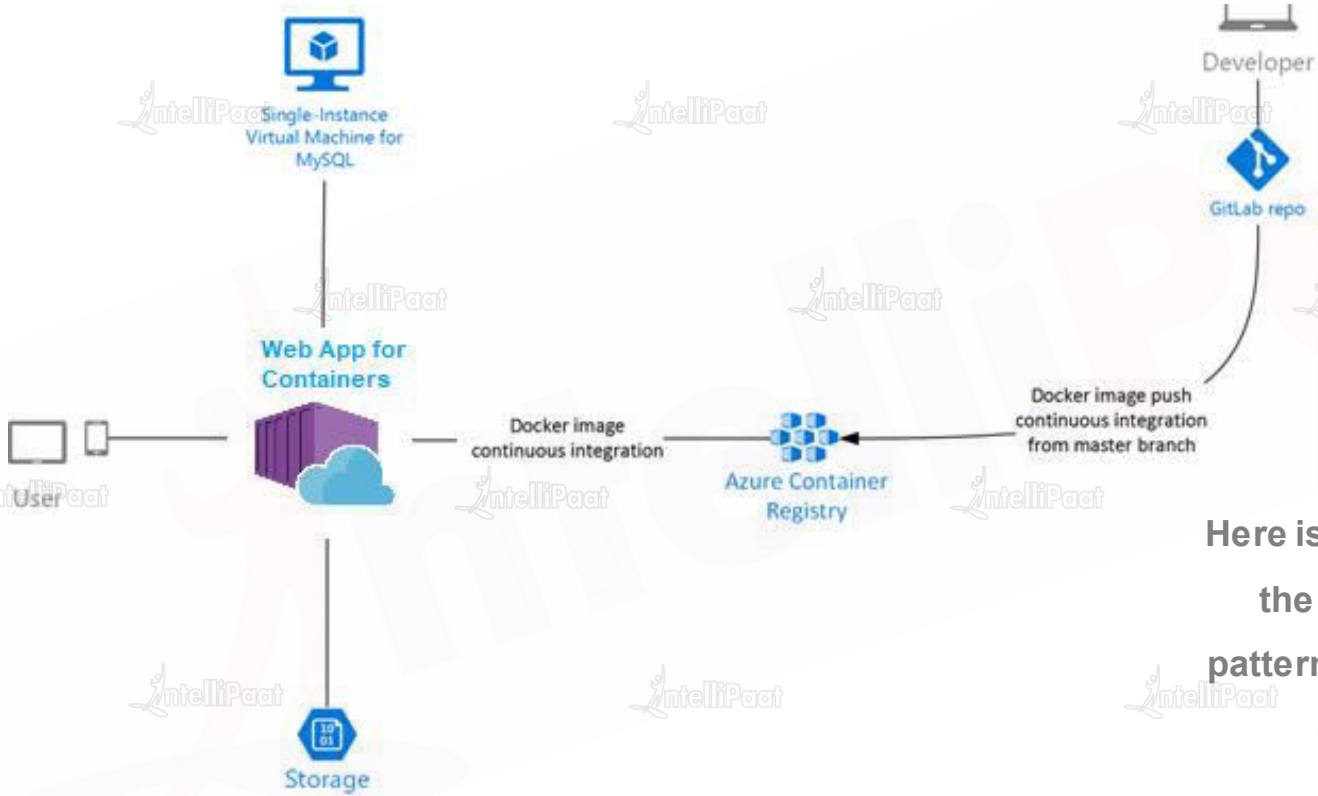
- Built-in auto scaling and load balancing

- Easily streamline CI/CD with Docker Hub, Azure Container Registry, and GitHub

Web App for Containers

- Web App for containers is catered more towards developers who want to have more control over, not just the code, but also the different packages, runtime framework, tooling, etc. that are installed on their containers.
- This offering is essentially bringing years worth of Azure App Service PaaS innovations to the community, by allowing developers to just focus on composing their containers without worrying about managing and maintaining an underlying container orchestrator. Customers of this offering prefer to package their code and dependencies into containers using various CI/CD systems like Jenkins, Maven, Travis CI, or Azure DevOps, alongside setting up continuous deployment webhooks with App Service.

Web App for Containers



Here is an example of one of the many architecture patterns used by customers of this offering.

Deploying a Web App for Containers

Opening Azure Cloud Shell

- Configure a deployment user, for which use the below command:

```
az webapp deployment user set --user-name <username> --password <password>
```

By this, you will get a JSON output with the password shown as null.

- Now, create a resource group using the below command:

```
az group create --name myResourceGroup --location "West Europe"
```

- You generally create your resource group and the resources in a region near to you.
- When the command finishes, a JSON output shows you the resource group properties.

Deploying a Web App for Containers

Creating an Azure App Service Plan

- In the Cloud Shell, create an App Service plan in the resource group with this command:

```
az appservice plan create --name myAppServicePlan  --resource-group myResourceGroup --sku B1 --is-linux
```

- When the App Service plan has been created, the Azure CLI shows information similar to the below example:

```
{ "adminSiteName": null,
"appServicePlanName": "myAppServicePlan",
"geoRegion": "West Europe",
"hostingEnvironmentProfile": null,
"id": "/subscriptions/0000-
0000/resourceGroups/myResourceGroup/providers/Microsoft.Web/serverfarms/myAppServicePlan",
"kind": "linux",
"location": "West Europe",
"maximumNumberOfWorkers": 1,
"name": "myAppServicePlan",
< JSON data removed for brevity. >
"targetWorkerSizeId": 0,
"type":  "Microsoft.Web/serverfarms",
"workerTierName": null }
```

Deploying a Web App for Containers

Creating a Web App

- Create a web app using this command:

```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app name> --deployment-container-image-name microsoft/azure-appservices-go-quickstart
```

- When the web app has been created, the Azure CLI shows output similar to the below example:

```
{
"availabilityState": "Normal",
"clientAffinityEnabled": true,
"clientCertEnabled": false,
"cloningInfo": null,
"containerSize": 0,
"dailyMemoryTimeQuota": 0,
"defaultHostName": "<app name>.azurewebsites.net",
"deploymentLocalGitUrl": "https://<username>@<app name>.scm.azurewebsites.net/<app name>.git",
"enabled": true,
< JSON data removed for brevity. >
}
```

Deploying a Web App for Containers

- Browse to the app:

http://<app_name>.azurewebsites.net/hello

- Congratulations! You've deployed a custom Docker image running an application to Web App for Containers.

Hands-on

Hands-on

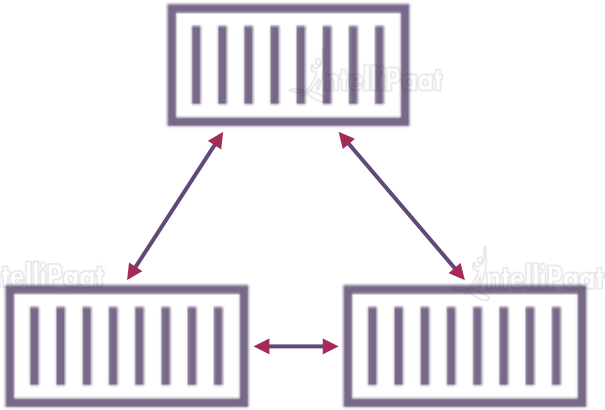
- Deploy a custom Docker image to Azure
- Configure environment variables to run the container
- Update the Docker image and redeploy it
- Connect to the container using SSH
- Deploy a private Docker image to Azure

Hint: In a terminal window, run the following command to clone the sample app repository to your local machine, and then change to the directory that contains the sample code.

>> git clone https://github.com/Azure-Samples/docker-django-webapp-linux.git --config core.autocrlf=input
>> cd docker-django-webapp-linux

Challenges of Microservices

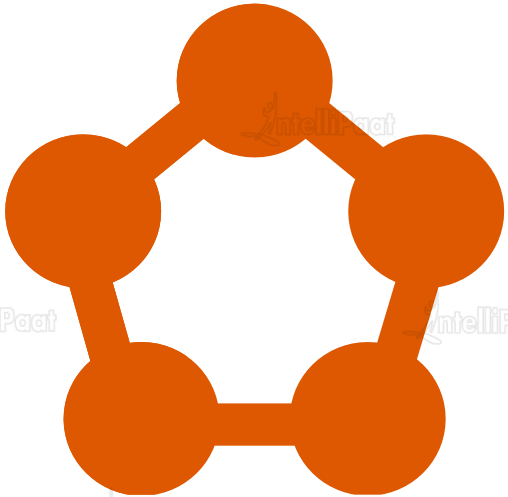
Challenges of Microservices



- Deployment
- Health monitoring
- Scaling out to multiple instances Service to service communication Upgrades
- Recover from hardware failures Orchestrators can help us
  - Azure Service Fabric

Azure Service Fabric

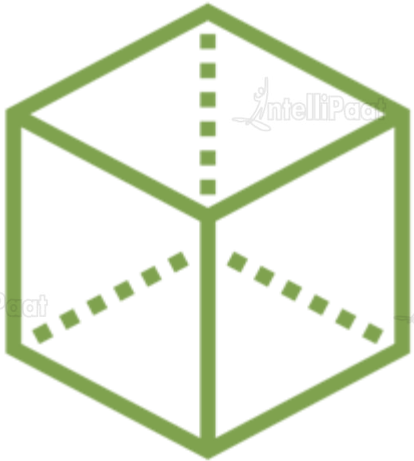
Azure Service Fabric



- An “application platform”
- Scalable and reliable micro services
- Hosting options
  - On-premises or other cloud providers
  - Development laptop
  - Azure
- Cluster
  - Monitors service health

Programming Models

Programming Models



- Stateful services
  - Co-locate compute and data
  - Reliable collections
- Stateless services
  - Web APIs or executables
  - Containers
- Linux and Windows containers
  - Constrain RAM and CPU allocation
  - Docker Compose YAML support

Service Fabric Benefits

Service Fabric Benefits

- Powers many key Azure services
  - e.g. Cortana, Skype, Cosmos DB & Power BI
- Why choose Service Fabric?
  - Microservices applications
  - Windows containers
  - Ability to deploy outside Azure
  - Orchestration features

Hands-On

Hands-On 1

- Setting up an Azure Service Fabric development environment
  - Service Fabric Tools

Hands-On 2

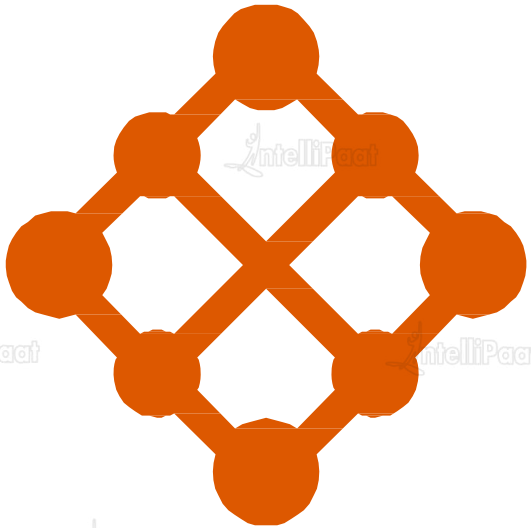
- Creating a Service Fabric cluster in the Azure portal

Hands-On 3

- Deploy an application to a Service Fabric cluster

Service Fabric Mesh

Service Fabric Mesh



- Based on Service Fabric
- Simplified deployment model
  - Container focused
  - Serverless – no need to pre-provision infrastructure
  - Just specify resources required per service
  - Deployment model based on ARM
  - YAML format also available

Preview Alert

Preview Alert

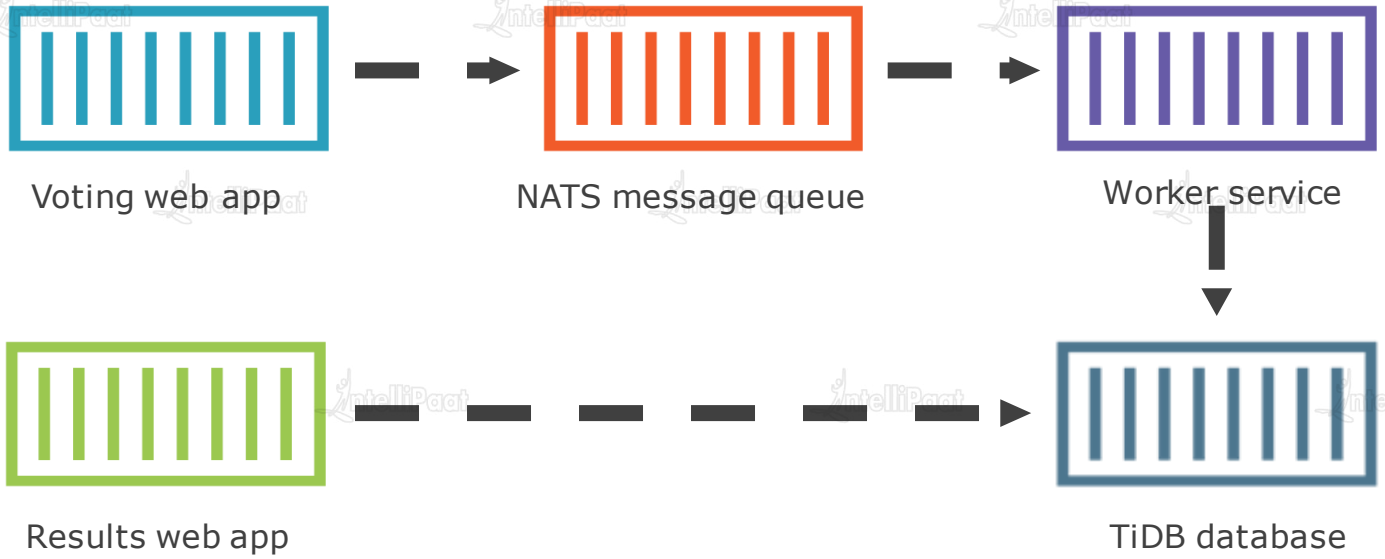
- Deployment file formats may change
- Command line syntax may change
- Some features not yet available

Hands-On

- Deploy a containerized Windows application to Service Fabric Mesh
  - Azure CLI “mesh” extension
  - Deploy an ARM template
  - Scale a service to multiple instances

Demo Voting Application

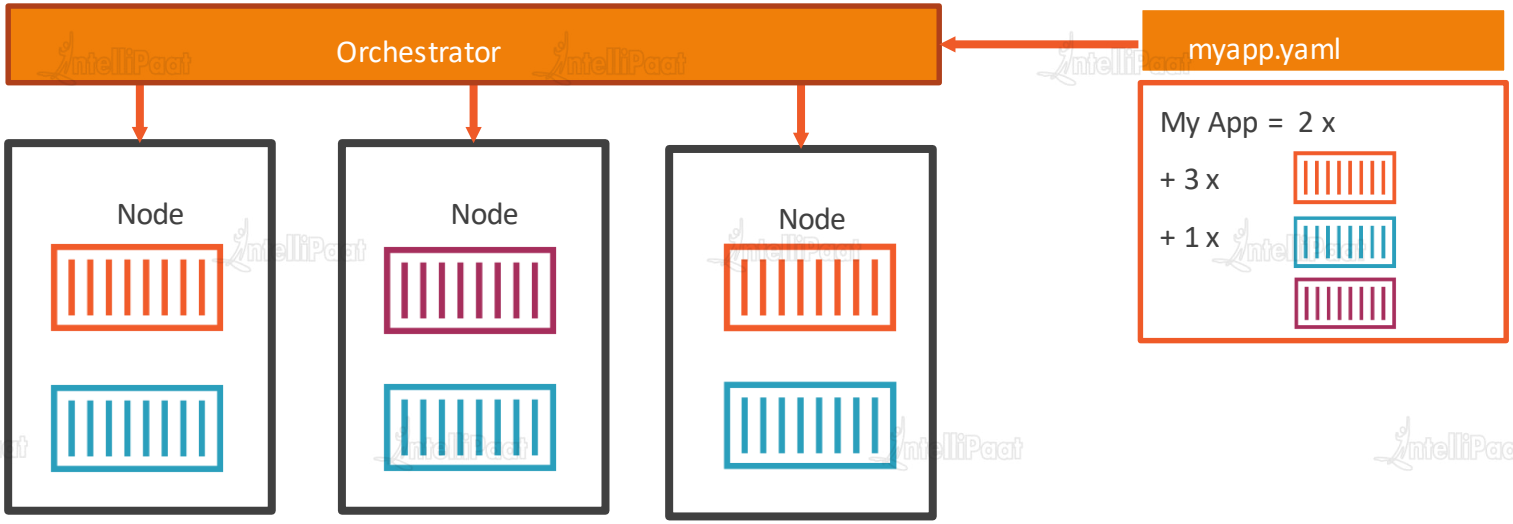
Demo Voting Application Using TiDB



https://github.com/dockersamples/example-voting-app

Orchestrators

Orchestrators



Health monitoring | Self-healing| Upgrades | Scaling

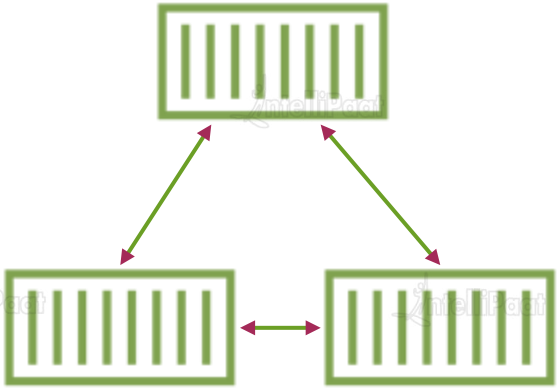
Resource constraints | Networking| Service discovery | Ingress

Kubernetes Basics

Kubernetes Basics

- A “production grade container orchestration system”
- Cluster
  - Master nodes schedule containers
  - Worker nodes run containers
- Kubectl
  - Command line tooling

Kubernetes Concepts

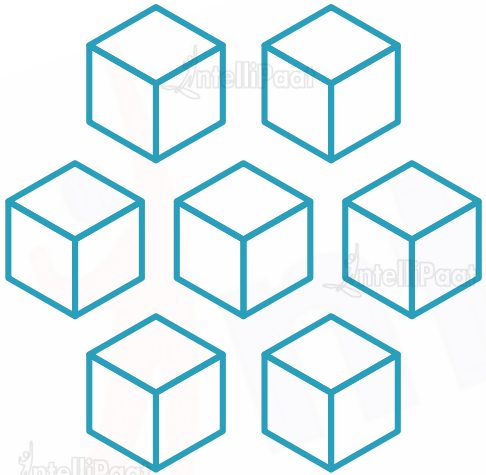


- Pod – one or more containers
- ReplicaSet
  - multiple instances of a pod Deployment
  - running code on Kubernetes Service
  - load balancing
- Namespaces – isolation
- YAML– declarative deployments
- Helm – package manager for Kubernetes

Azure Kubernetes Service (AKS)

Azure Kubernetes Service

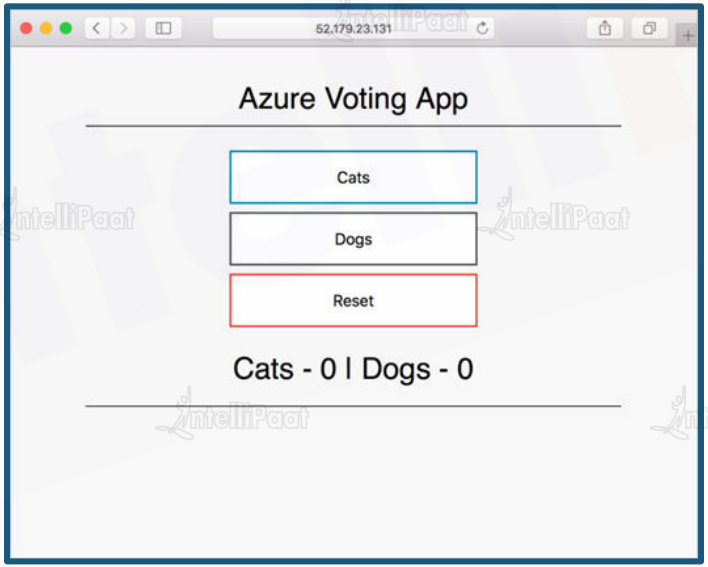
Azure Kubernetes Service (AKS) makes it simple to deploy a managed Kubernetes cluster in Azure. Basically, it reduces the complexity and operational overhead of managing Kubernetes by offloading much of that responsibility to Azure.



- As a hosted Kubernetes service, Azure handles critical tasks like health monitoring and maintenance for you.
- Kubernetes masters are managed by Azure.
- You only manage and maintain agent nodes.
- As a managed Kubernetes service, AKS is free—you only need to pay for the agent nodes within your clusters, not for the masters.

Deploying an AKS Cluster Using Azure CLI

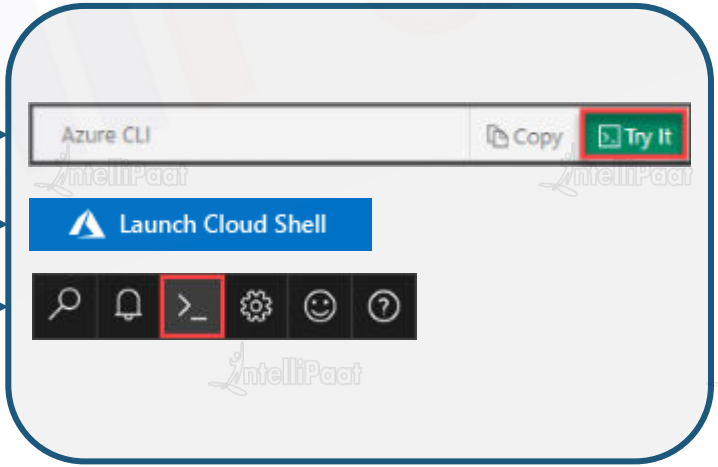
AKS is a managed Kubernetes service that lets you quickly deploy and manage clusters. In this quick start, you deploy an AKS cluster using the Azure CLI. A multi-container application that includes a web front end and a Redis instance is run in the cluster. You can then see how to monitor the health of the cluster and pods that run your application.



Deploying an AKS Cluster Using Azure CLI

Azure Cloud Shell is a free, interactive shell that you can use to run the steps in this article. Common Azure tools are preinstalled and configured in Cloud Shell for you to use with your account. Just select the Copy button to copy the code, paste it in Cloud Shell, and then press Enter to run it. There are a few ways to open Cloud Shell:

- Select the Try It option in the upper-right corner of a code block.
- Open Cloud Shell in your browser.
- Select the Cloud Shell button on the menu in the upper-right corner of the Azure portal.



Deploying an AKS Cluster Using Azure CLI

Create a **Resource Group** using the *az group create* command

The following example creates a resource group named *myResourceGroup* in the *eastus* location.

In Azure CLI, use the following command:

```
az group create --name myResourceGroup --location eastus
```

The following example output shows that the resource group has been created successfully.

```
{"id": "/subscriptions/<guid>/resourceGroups/myResourceGroup",
"location": "eastus",
"managedBy": null,
"name": "myResourceGroup",
"properties": {
"provisioningState": "Succeeded"
},
"tags": null
}
```

Deploying an AKS Cluster Using Azure CLI

Creating the AKS Cluster

Use the *az aks create* command to create an AKS cluster. Following example creates a cluster named *myAKSCluster* with one node. Azure Monitor for containers is also enabled using the --enable-addons monitoring parameter.

In Azure CLI, use the following command:

```
az aks create \
--resource-group myResourceGroup \
--name myAKSCluster \
--node-count 1 \
--enable-addons monitoring \
--generate-ssh-keys
```

After a few minutes, the command completes and returns JSON-formatted information about the cluster.

Deploying an AKS Cluster Using Azure CLI

Connecting to the Cluster

To manage a Kubernetes cluster, you use kubectl, the Kubernetes command-line client. If you use Azure Cloud Shell, kubectl is already installed.

To install kubectl locally, in Azure CLI, use the following command:

az aks install-cli

*az aks get-credentials --resource-group myResourceGroup --name myAKSCluster ll**command downloads credentials and configures the Kubernetes CLI *

*kubectl get nodes ll** to return a list of the cluster nodes*

Deploying an AKS Cluster Using Azure CLI

The following example output shows the single node that has been created in the previous steps. Make sure that the status of the node is **Ready**

```
NAME                       STATUS   ROLES   AGE     VERSION
aks-nodepool1-31718369-0   Ready    agent   6m44s   v1.9.11
```

Next step is to run the application.

Create a file named ***azure-vote.yaml*** and copy in the following YAML definition. If you use the Azure Cloud Shell, this file can be created by using ***vi*** or ***nano*** just like the way you create on virtual or physical systems. You can find the YAML Commands in the next slides.

Deploying an AKS Cluster Using Azure CLI

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-back
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-back
  template:
    metadata:
      labels:
        app: azure-vote-back
```

```
spec:
      containers:
      - name: azure-vote-back
        image: redis
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 250m
            memory: 256Mi
        ports:
        - containerPort: 6379
```

```
name: redis
---
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-back
spec:
  ports:
  - port: 6379
  selector:
    app: azure-vote-back
---
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
name: azure-vote-front
spec: replicas: 1
  selector:
    matchLabels:
      app: azure-vote-front
  template:
    metadata:
      labels:
        app: azure-vote-front
    spec:
      containers:
      - name: azure-vote-front
```

Deploying an AKS Cluster Using Azure CLI

```
 image: microsoft/azure-vote-front:v1
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 250m
            memory: 256Mi
        ports:
        - containerPort: 80
        env:
        - name: REDIS
          value: "azure-vote-back"
```

```
---
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-front
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: azure-vote-front
```

(Use the commands in the sequence)

Deploying an AKS Cluster Using Azure CLI

Deploy the application using the *kubectl apply* command and specify the name of your YAML manifest

```
kubectl apply -f azure-vote.yaml
```

The following example output shows that the Deployments and Services have been created successfully.

```
deployment "azure-vote-back" created
service "azure-vote-back" created
deployment "azure-vote-front" created
service "azure-vote-front" created
```

Test the application: To monitor the progress, use the ***kubectl get service*** command with the ***--watch*** argument.

```
kubectl get service azure-vote-front --watch
```

Deploying an AKS Cluster Using Azure CLI

Initially, the EXTERNAL-IP for the azure-vote-front service is shown as pending.

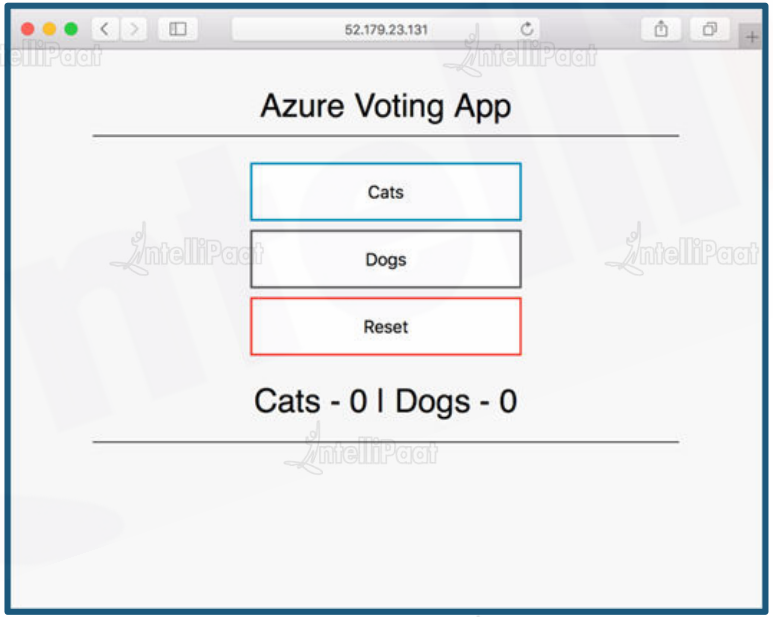
```
NAME               TYPE           CLUSTER-IP    EXTERNAL-IP   PORT(S)        AGE
azure-vote-front   LoadBalancer   10.0.37.27    <pending>     80:30572/TCP   6s
```

When the EXTERNAL-IP address changes from pending to an actual public IP address, use CTRL-C to stop the kubectl watch process. The following example output shows that a valid public IP address is assigned to the service.

```
azure-vote-front   LoadBalancer   10.0.37.27   52.179.23.131   80:30572/TCP   2m
```

Deploying an AKS Cluster Using Azure CLI

To see the Azure Vote app in action, open a web browser to the external IP address of your service.



Deploying an AKS Cluster Using Azure CLI

Monitoring Health and Logs

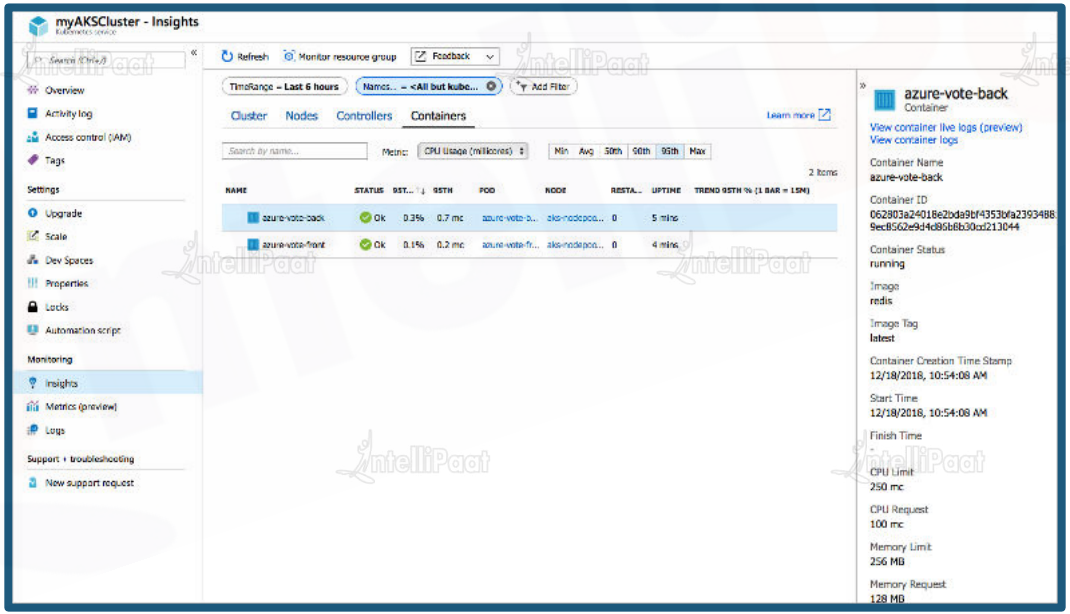
When the AKS cluster was created, Azure Monitor for containers was enabled to capture health metrics for both cluster nodes and pods. These health metrics are available in the Azure portal.

To see current status, uptime, and resource usage for the Azure Vote pods, complete the following steps:

- Open a web browser to the Azure portal https://portal.azure.com
- Select your resource group, such as *myResourceGroup*, and then select your AKS cluster, such as *myAKSCluster*
- Under Monitoring on the left-hand side, choose **Insights**
- Across the top, choose **+ Add Filter**
- Select Namespace as the property, then choose ***<All but kube-system>***
- Choose View the Containers

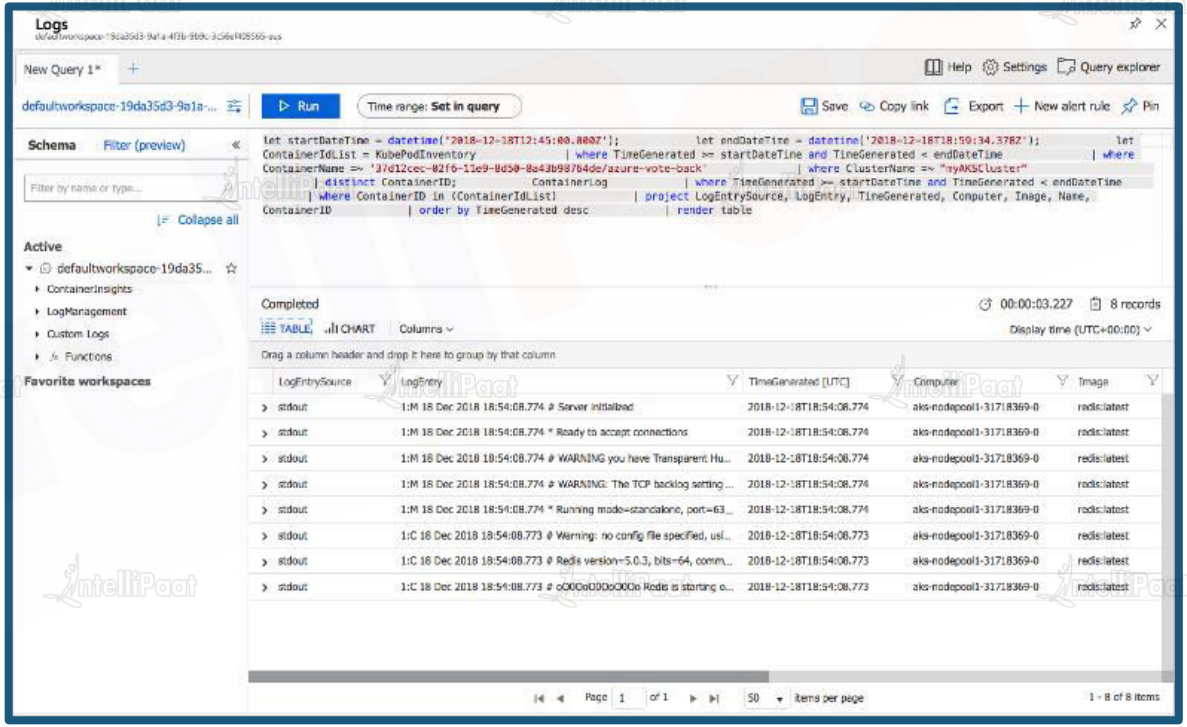
Deploying an AKS Cluster Using Azure CLI

The *azure-vote-back* and *azure-vote-front* containers are displayed, as shown in the following example:



Deploying an AKS Cluster Using Azure CLI

To see logs for the ***azure-vote-front*** pod, select the View container logs link on the right-hand side of the containers list. These logs include the ***stdout*** and ***stderr*** streams from the container.



Deploying an AKS Cluster Using Azure CLI

Deleting the Cluster

When the cluster is no longer needed, use the ***az group delete*** command to remove the resource group, container service, and all related resources.

```
az group delete --name myResourceGroup --yes --no-wait
```

Getting the Code for This Demo

The related application code, Dockerfile, and the Kubernetes manifest file are available on GitHub.

https://github.com/Azure-Samples/azure-voting-app-redis.git

Hands-on

- Deploy an AKS cluster using the Azure portal
  - Create an AKS cluster
  - Connect to the cluster
  - Run the application
  - Test the application
  - Monitor health and logs
  - Delete the cluster

The related application code, Dockerfile, and the Kubernetes manifest file are available on GitHub.
https://github.com/Azure-Samples/azure-voting-app-redis

Hint: You may perform similar steps for the Azure portal which are used in CLI.

Integration with Azure Services

Integration with Azure Services

- Azure Monitor
- Mount Azure file shares or disks Secure with RBAC and AD Virtual network integration Elastic scale with ACI
- Develop and debug with Dev Spaces

Hands-On

- Scaling AKS
  - Scale cluster
  - Scale a microservice
  - View the Kubernetes dashboard

Choosing an Orchestrator

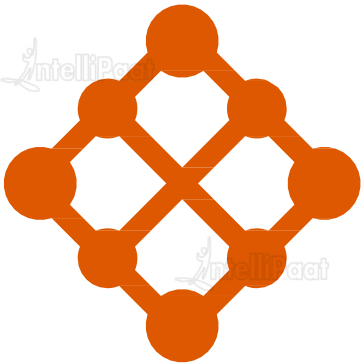
Choosing an Orchestrator

Azure Service Fabric

Azure Kubernetes Service

Both are excellent orchestrators!

- Scheduling
- Upgrades
- Health Monitoring
- Service Discovery



- Windows
- Stateful services
- Serverless (Mesh)

- Open source tooling ecosystem
- Other clouds
- Virtual Kubelet
- Dev Spaces

India: +91-7847955955

US: 1-800-216-8930 (TOLL FREE)

sales@intellipaat.com

24/7 Chat with Our Course Advisor