**Microsoft Ready 2020**

**SE360T**

**Modernize .NET core app with Azure Functions on Serverless Kubernetes and Azure Kubernetes Service**

Akash Khandelwal (akashkh)

Finbar Ryan (Finbarr)

CONTENTS

* INTRODUCTION
* TASK 1
  + Create the Azure Functions locally and test with Queue Trigger
* TASK 2
  + Build the Docker Image and deploy it to Kubernetes with the Virtual Nodes
* TASK 3
  + Load Test the Storage Queue and watch the Pod Scale out.
* TASK 4
  + Look at the Functions logs and information in Insights to help Debug Functions issue.
* EPILOGUE

INTRODUCTION

**Estimated time**

40 mins

**Objective**

After completing this lab, you will be able to

* Develop and Deploy Azure Functions app on Kubernetes with KEDA

**Prerequisites**

This document is designed to walk you through the whole lab. However, to take the most out of it you are expected to have

* Basic knowledge of Azure functions.
* Basic understanding of Docker and Containers.
* Basic understanding of Kubernetes

**Overview of the lab**

The lab consists of 4 steps. This document provides a complete walkthrough of Develop and Deployment of Azure functions on Kubernetes.

GETTING THINGS READY

First, we need to setup environment which is very simple.

To run this lab, we need:

* An Azure Subscription (one provided by Ready) which have already resources created for the purpose of this lab:
  + Azure Kubernetes Clusters
  + Azure storage account
  + Azure container registry
  + Azure VM

TASK 1: Create the Azure Functions locally and test with Queue Trigger

**1. Open CMD on the Virtual machine**

**2. Create a new directory for the function app**

|  |
| --- |
| mkdir hello-keda  cd hello-keda |

**3. Initialize the directory for functions**

|  |
| --- |
| func init . --docker |

Select dotnet

**4. Add a new queue triggered function**

|  |
| --- |
| func new |

Select Azure Queue Storage Trigger

Leave the default of QueueTrigger1 for the name

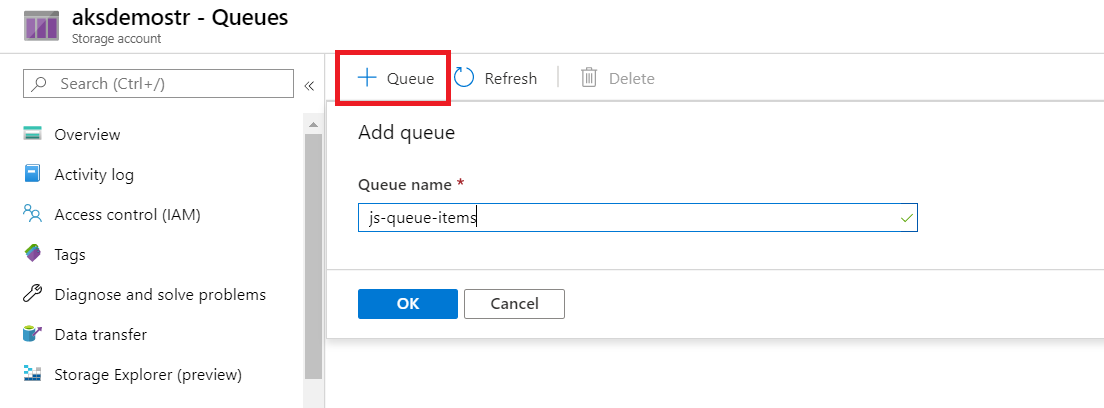
**5. Create an Azure storage queue**

Azure storage account is already created in given subscription.

Create an Azure storage queue and use name **js-queue-items**

A screenshot of a cell phone

Description automatically generated



**6. Update the function metadata with the storage account info**

Open the hello-keda directory in VSCODE by typing below command in CMD

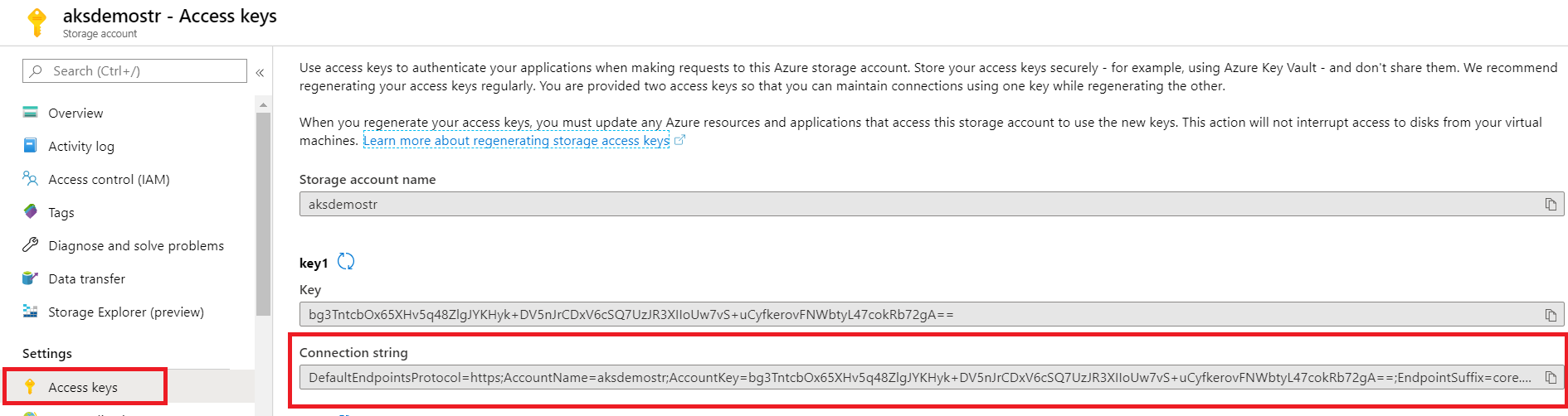
|  |
| --- |
| code . |

A picture containing black, indoor, photo

Description automatically generated

We'll need to update the connection string info for the queue trigger, and make sure the queue trigger capabilities are installed.

Copy the current storage account connection string (HINT: don't include the ")



Open **local.settings.json** which has the local debug connection string settings. Replace the {AzureWebJobsStorage} with the connection string value:

**local.settings.json**

{

"IsEncrypted": false,

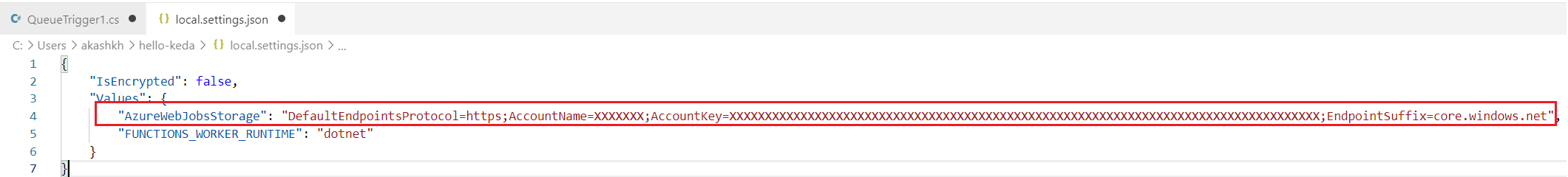
"Values": {

"FUNCTIONS\_WORKER\_RUNTIME": "node",

"AzureWebJobsStorage": “DefaultEndpointsProtocol=https;EndpointSuffix=core.windows.net;AccountName=mystorageaccount;AccountKey=shhhh==="

}

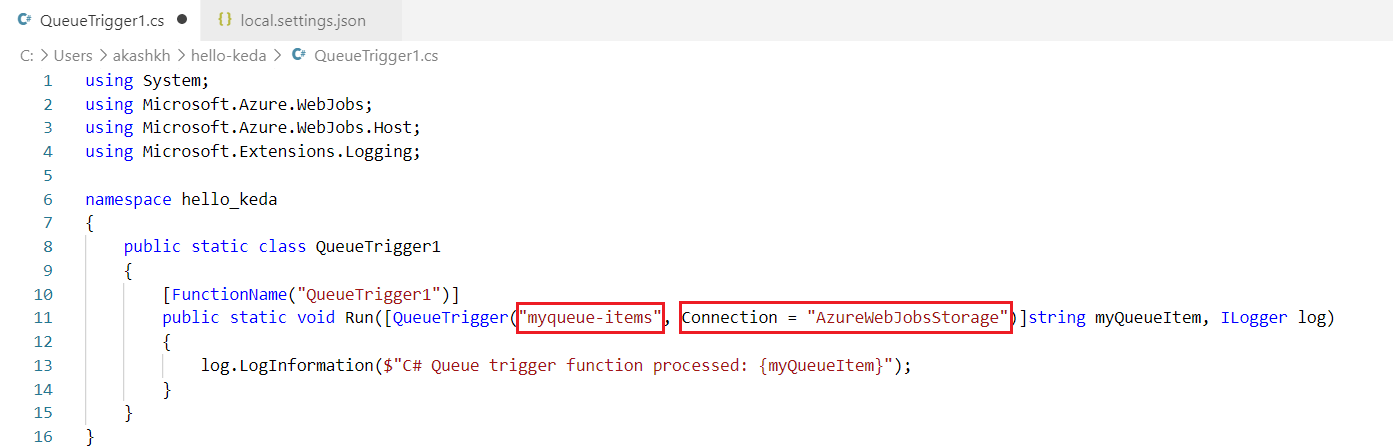
}



Open **QueueTrigger1.cs** which has Queue name and connection settings. Replace the queue name to

**“js-queue-items**” and **Connection = "AzureWebJobsStorage"**

**QueueTrigger1.cs**



**7. Enable the storage queue bundle on the function runtime**

Replace the **host.json** content with the following. This [pulls in the extensions to the function runtime](https://docs.microsoft.com/azure/azure-functions/functions-bindings-register#local-development-with-azure-functions-core-tools-and-extension-bundles) like Azure Storage Queues support.

**host.json**

{

"version": "2.0",

"extensionBundle": {

"id": "Microsoft.Azure.Functions.ExtensionBundle",

"version": "[1.\*, 2.0.0)"

} ,

    "extensions": {

        "queues": {

            "batchSize": 1

        }

    }

}

**8. Debug and test the function locally**

Start the function locally

func start

Go to your Azure Storage account in the [Azure Portal](https://portal.azure.com/) and open the **queues**. Select the js-queue-items queue and add a message to send to the function.

A screenshot of a cell phone

Description automatically generated

You should see your function running locally fired correctly immediately

A close up of a screen

Description automatically generated

TASK 2: Build the Docker Image and deploy it to Kubernetes with the Virtual Nodes

**1. Install KEDA**

1. Add Helm repo

helm repo add kedacore <https://kedacore.github.io/charts>

1. Update Helm repo

helm repo update

1. Install keda Helm chart

kubectl create namespace keda

helm install keda kedacore/keda --namespace keda

To confirm that KEDA has successfully installed you can run the following command and should see the following CRD.

kubectl get customresourcedefinition

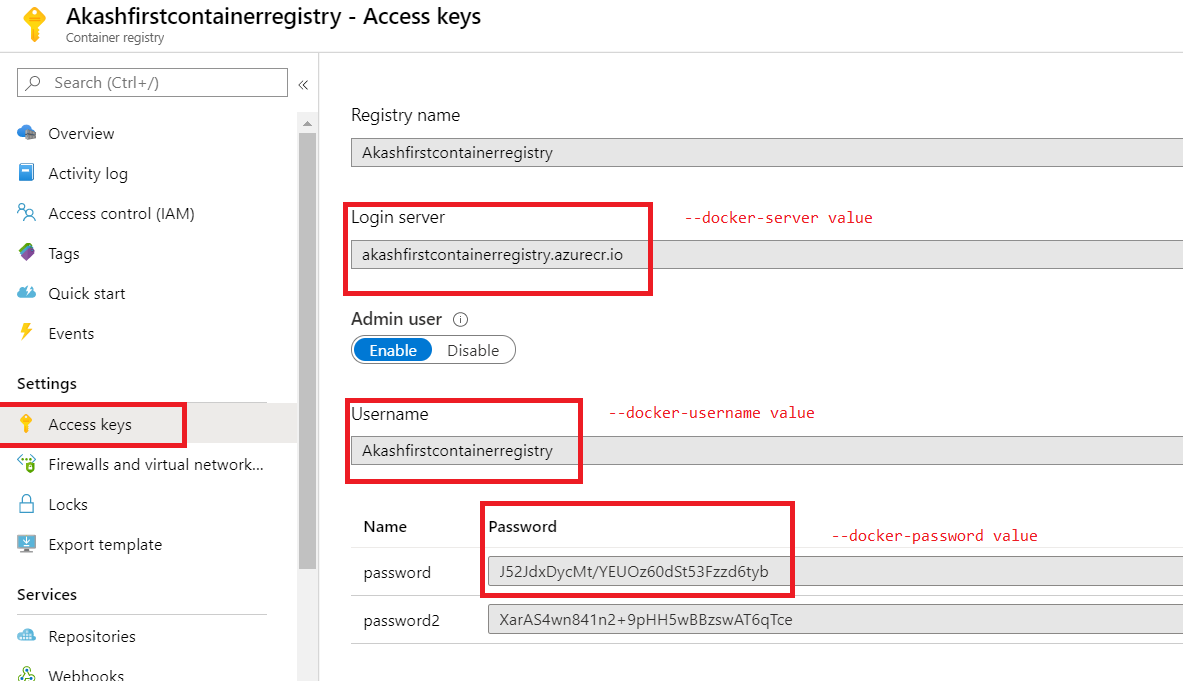
NAME AGE

scaledobjects.keda.k8s.io 2h

**2. We need to create Kubernetes secret to fetch image from ACR using secrets.**

kubectl create secret docker-registry acr-auth --docker-server=<acr\_name>.azurecr.io --docker-username=<acr username> --docker-password=<ACR password> --docker-email [xxxxxx](mailto:odl_user_142946@cloudlabsaioutlook.onmicrosoft.com)

--docker-email is email address from which Azure portal is logged in like xxxx@cloudlabsaioutlook.onmicrosoft.com



**3. Deploy Function App to KEDA (Virtual Nodes)**

To deploy your function Kubernetes with Azure Virtual Nodes, you need to modify the details of the deployment to allow the selection of virtual nodes.

Generate a deployment yaml for the function.

In below command add ACR name as <acrname>.azurecr.io

func kubernetes deploy --name hello-keda --registry <acr\_name>.azurecr.io --dotnet --pull-secret acr-auth --dry-run > deploy.yaml

Open and modify the created deploy.yaml to tolerate scheduling onto any nodes, including virtual.

spec:

      containers:

      - name: hello-keda

        image: Akashfirstcontainerregistry.azurecr.io/hello-keda

        env:

        - name: AzureFunctionsJobHost\_\_functions\_\_0

          value: QueueTrigger1

        envFrom:

        - secretRef:

            name: hello-keda

      imagePullSecrets:

      - name: acr-auth

      tolerations:

      - operator: Exists

Your deploy.yaml should look like below screenshot with alignment as shown.

A screenshot of a social media post

Description automatically generated

Login into ACR

az acr login –name <acr name>

Build and deploy the container image and apply the deployment to your cluster.

Replace < your-acr-id> with <ACRname>.azurecr.io in below commands.

docker build -t <your-acr-id>/hello-keda .

docker push <your-acr-id>/hello-keda

kubectl apply -f deploy.yaml

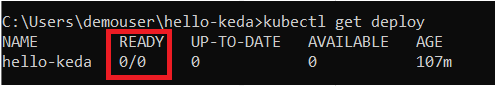
*Note: in docker build command above .(dot) in the last is required*

TASK 3: Load Test the Storage Queue and watch the Pod Scale out.

**1. Add a queue message and validate the function app scales with KEDA**

Initially after deploying and with an empty queue you should see 0 pods.

kubectl get deploy



Add a queue message to the queue (using portal in step 8 of TASK 1 above). KEDA will detect the event and add a pod. By default, the polling interval set is 30 seconds on the ScaledObject resource, so it may take up to 30 seconds for the queue message to be detected and activate your function. This can be [adjusted on the ScaledObject resource](https://github.com/kedacore/keda/wiki/ScaledObject-spec).

2. Load Test the Storage Queue and watch the Pod Scale out.

Open new CMD window and type in below command

C:\Labfiles\Sendmessagestoqueue>SendMessageToQueue.exe

Provide the connection string, queue name (**js-queue-items**) and no. of message you need to send to queue (like 1000 messages)

A screenshot of a social media post

Description automatically generated

A screen shot of a computer

Description automatically generated

3. Watch the pod scale out

kubectl get deploy

kubectl get pods -w

The queue message will be consumed. You can validate the message was consumed by using kubectl logs on the activated pod. New queue messages will be consumed and if enough queue messages are added the function will autoscale. After all messages are consumed and the cooldown period has elapsed (default 300 seconds), the last pod should scale back down to zero.

TASK 4: Look at the Functions logs and information in Insights to help Debug Functions issue.

**1. Function logs:**

We can view function logs in AKS Insight logs

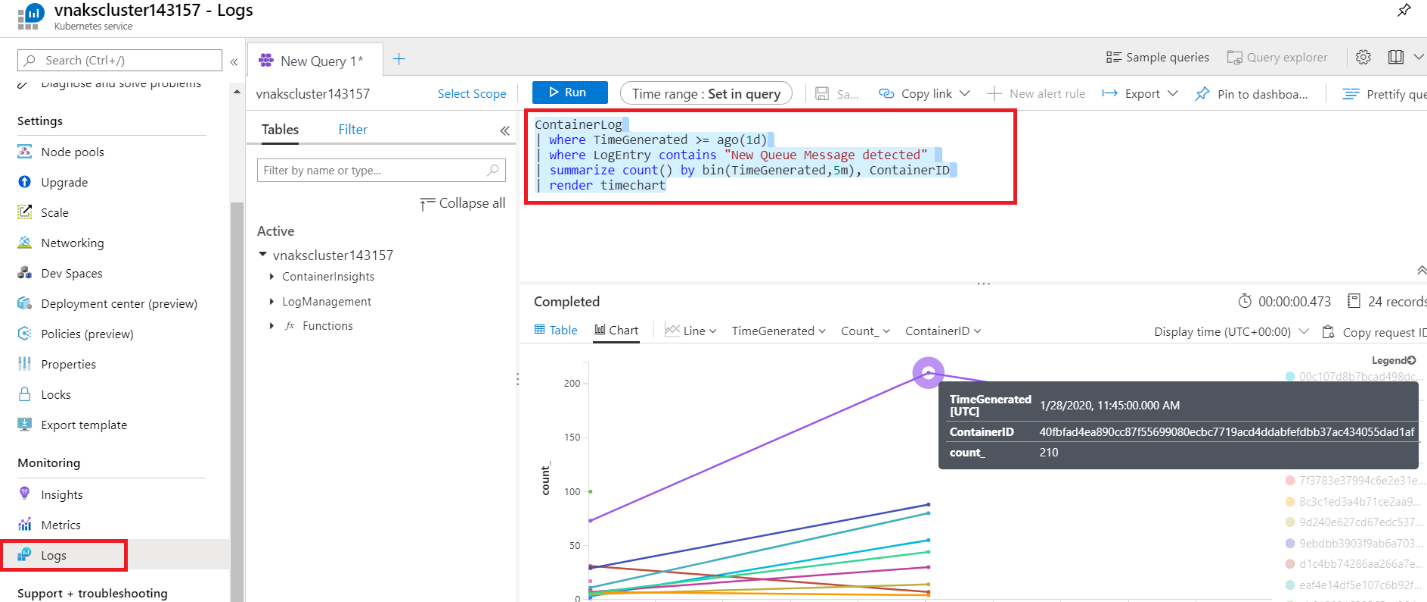
ContainerLog

| where TimeGenerated >= ago(1d)

| where LogEntry contains "New Queue Message detected"

| summarize count() by bin(TimeGenerated,5m), ContainerID

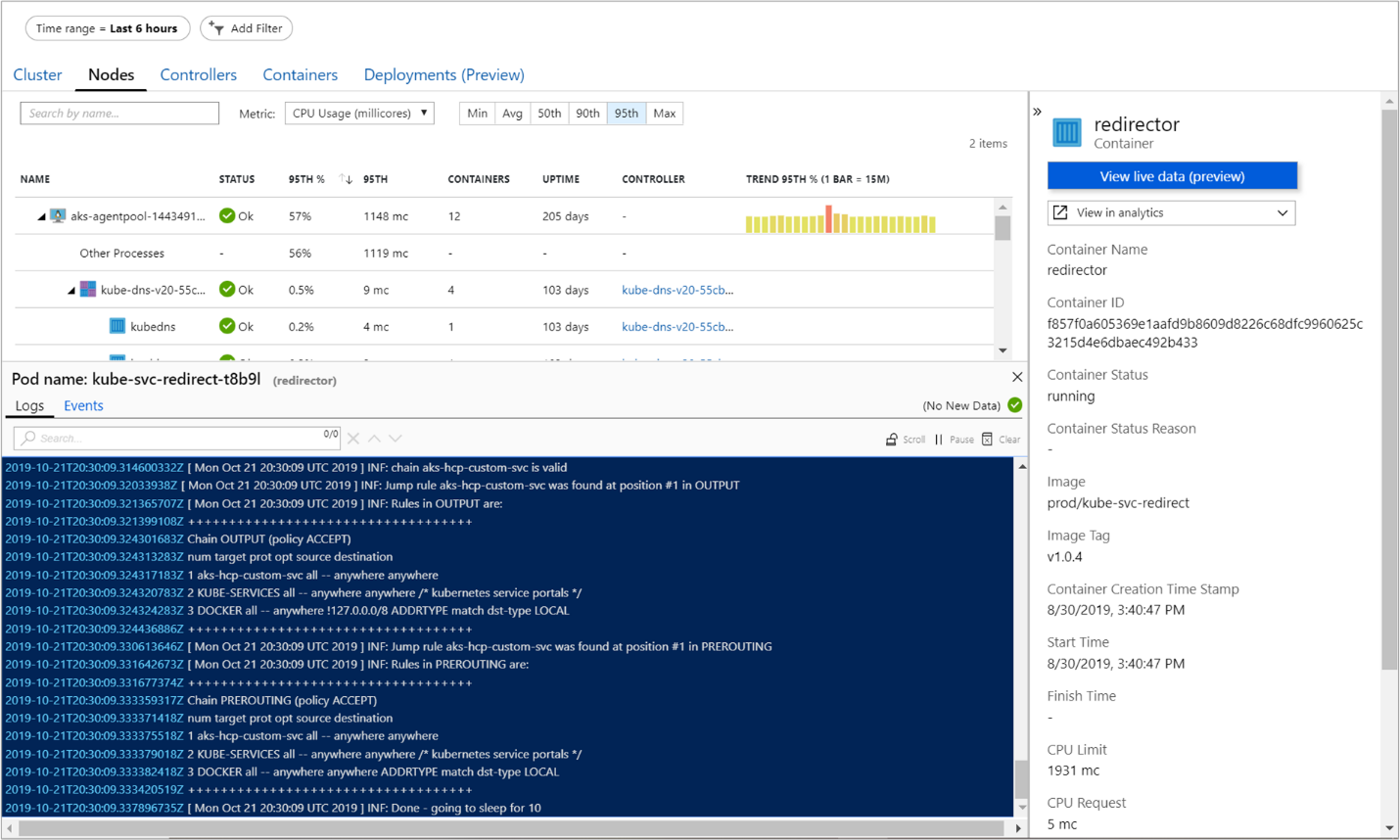
| render timechart



**2. View logs**

You can view real-time log data as they are generated by the container engine from the Nodes, Controllers, and Containers view. To view log data, perform the following steps.

1. In the Azure portal, browse to the AKS cluster resource group and select your AKS resource.
2. On the AKS cluster dashboard, under Monitoring on the left-hand side, choose Insights.
3. Select either the Nodes, Controllers, or Containers tab.
4. Select an object from the performance grid, and on the properties pane found on the right side, select View live data (preview) option. If the AKS cluster is configured with single sign-on using Azure AD, you are prompted to authenticate on first use during that browser session. Select your account and complete authentication with Azure.

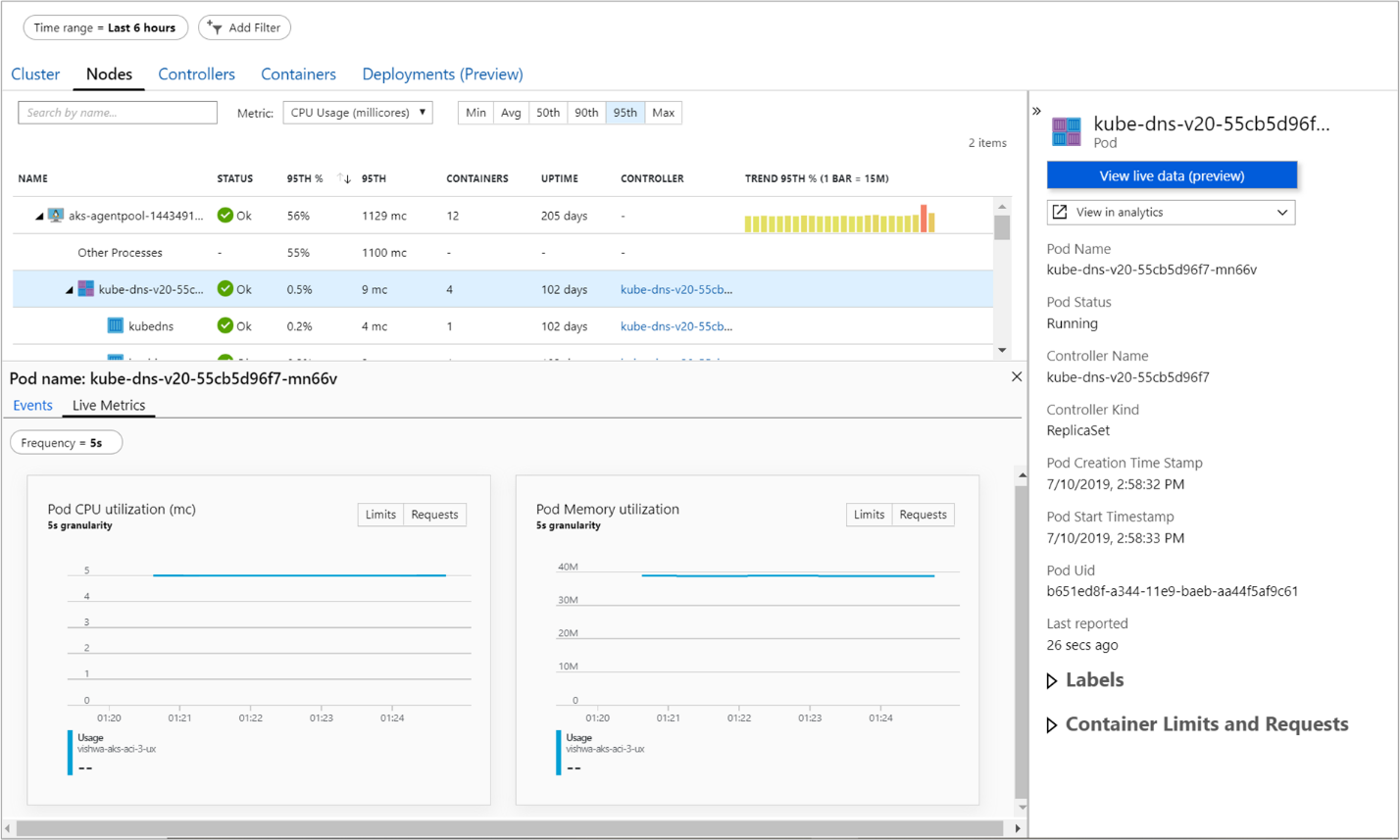


The pane title shows the name of the pod the container is grouped with.

**3. View metrics**

You can view real-time metric data as they are generated by the container engine from the **Nodes** or **Controllers** view only when a **Pod** is selected. To view metrics, perform the following steps.

1. In the Azure portal, browse to the AKS cluster resource group and select your AKS resource.
2. On the AKS cluster dashboard, under **Monitoring** on the left-hand side, choose **Insights**.
3. Select either the **Nodes** or **Controllers** tab.
4. Select a **Pod** object from the performance grid, and on the properties pane found on the right side, select **View live data (preview)** option. If the AKS cluster is configured with single sign-on using Azure AD, you are prompted to authenticate on first use during that browser session. Select your account and complete authentication with Azure.



EPILOGUE

**THANK YOU**

Congratulations! You made it through the end!

Thank you for taking this lab!

We enjoyed developing the lab and we sincerely hope it met your expectations!

The exercise in the lab were meant to touch base with some new trending technologies in the Azure function app developer’s world.

Many of us have a strong background in developing and deploying Azure Functions app, built upon several years. Today, with all the new exciting scenarios and technologies fostered by the cloud, it is easy to feel left behind. However, even in this brand-new world, we can still rely on our own skills!

It is the approach we sue towards developing and deploying that makes the difference!

**LAB EVALUATION**

Please take a minute to let us know what you think!

This is super important! All your comments, recommendations, suggestions or complaints are extremely important to us! You satisfaction is the ultimate driver that motivates us to work harder and better for the next time!