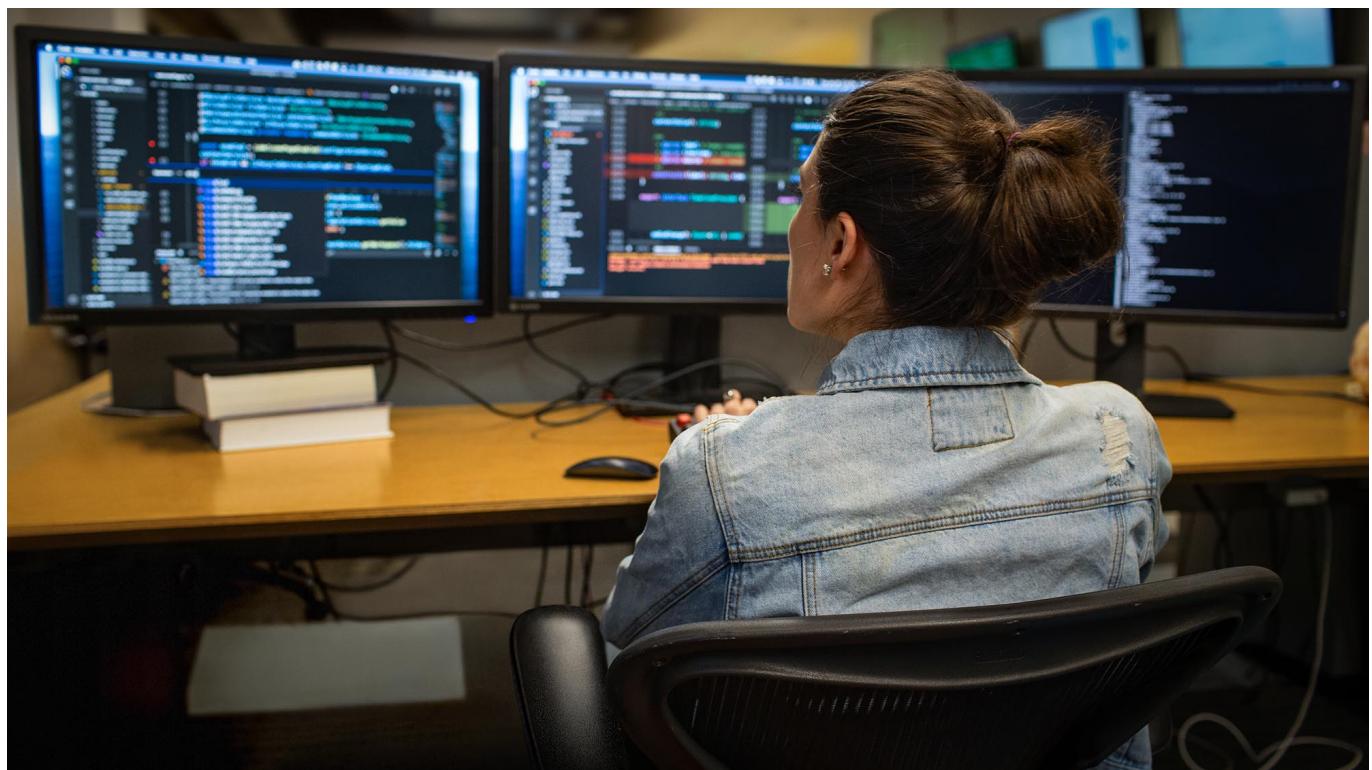


# Lab Module 4: Application Development with Kubernetes



Estimated Duration: 60 minutes

- Lab Module 4: Application Development with Kubernetes
  - Create a Basic Development Cluster
    - Task 1 - Create an AKS cluster (or start an existing one)
  - Exercise: Decide Which Microservices Design Pattern to Use
    - Available projects
  - Exercise: Create Additional Azure Resources
    - Task 1 - Create an Azure Container Registry
    - Task 2 - Create an Azure Key Vault
    - Task 3 - Assign permissions to Azure Container Registry
  - Exercise: Use Helm to Deploy the Application
    - Task 1 - Build and push all the images to your ACR
    - Task 2 - Install a Helm Chart in your AKS cluster
    - Task 3 - Use Helm to deploy updates to your application
    - Task 4 - Use Helm to control which files are included when the chart is updated
  - Exercise: Use Visual Studio 2019 to Debug Microservices Locally with Bridge to Kubernetes
    - Task 1 - Prepare your microservice in Visual Studio 2019
    - Task 2 - Debug your microservice using Visual Studio 2019 and the Bridge to Kubernetes extension
  - Exercise: Use VS Code to Debug Microservices Locally with Bridge to Kubernetes
    - Task 1 - Prepare your microservice in VS Code and verify that it runs locally
    - Task 2 - Debug the microservice using VS Code and the Bridge to Kubernetes extension

- Exercise: Add Monitoring to Entire Application
  - Task 1 - Create an Application Insights and configure all the microservices to use it
- DO NOT DELETE ANY OF THE AZURE RESOURCES YOU CREATED IN THIS LAB!
- YOU WILL USE THE SAME ACR AND AKS CLUSTER IN LAB 5

## Create a Basic Development Cluster

For the exercises in this module, you'll need simple AKS cluster.

### Task 1 - Create an AKS cluster (or start an existing one)

1. Select the region closest to your location. Use '**eastus**' for United States workshops, '**westeurope**' for European workshops.
2. Define variables (update as needed)

```
$INITIALS="abc"
```

```
$YOUR_INITIALS="$(($INITIALS)).ToLower()  
$AKS_RESOURCE_GROUP="azure-$($INITIALS)-rg"  
$LOCATION="@lab.Variable(region)"  
$VM_SKU="Standard_D2as_v5"  
$AKS_NAME="aks-$($INITIALS)"  
$NODE_COUNT="2"
```

3. Create Resource Group

```
az group create --location $LOCATION  
--resource-group $AKS_RESOURCE_GROUP
```

4. Create Basic cluster.

```
az aks create --node-count $NODE_COUNT  
--generate-ssh-keys  
--node-vm-size $VM_SKU  
--name $AKS_NAME  
--resource-group $AKS_RESOURCE_GROUP
```

5. Connect to local environment

```
az aks get-credentials --name $AKS_NAME  
--resource-group $AKS_RESOURCE_GROUP
```

## 6. Verify connection

```
kubectl get nodes
```

## Exercise: Decide Which Microservices Design Pattern to Use

There are two applications available for the exercises in this lab. Which one you choose is entirely up to you. You'll be able to complete all the tasks with either application.

The screenshots and example in this lab use the **Chained** project (picked at random) for consistency.

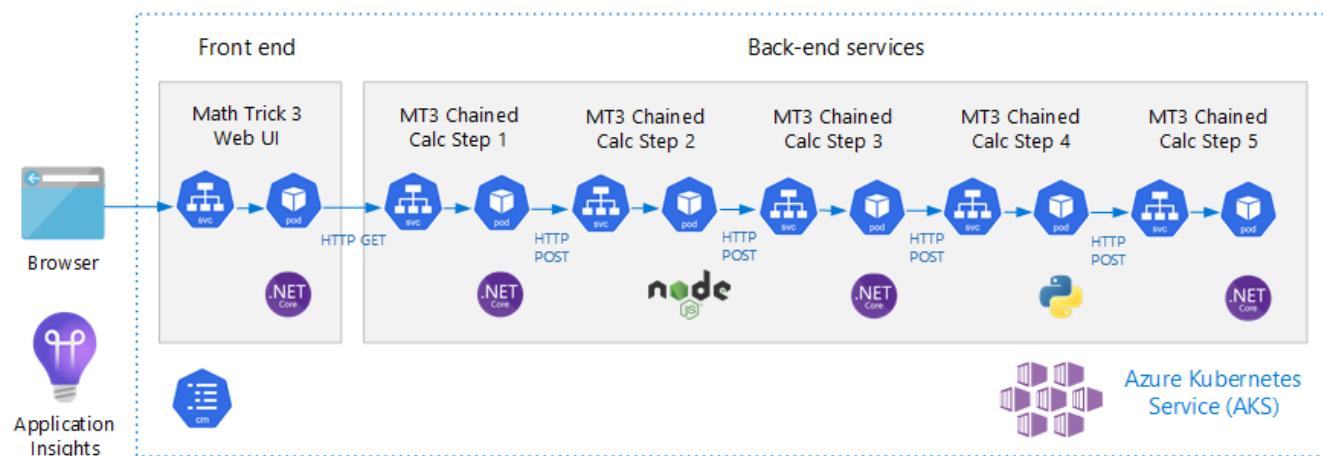
### Available projects

There are 2 subfolders in the **\Labs\MathTrick** folder:

```
\Labs
  \MathTrick
    \Chained
    \Gateway
```

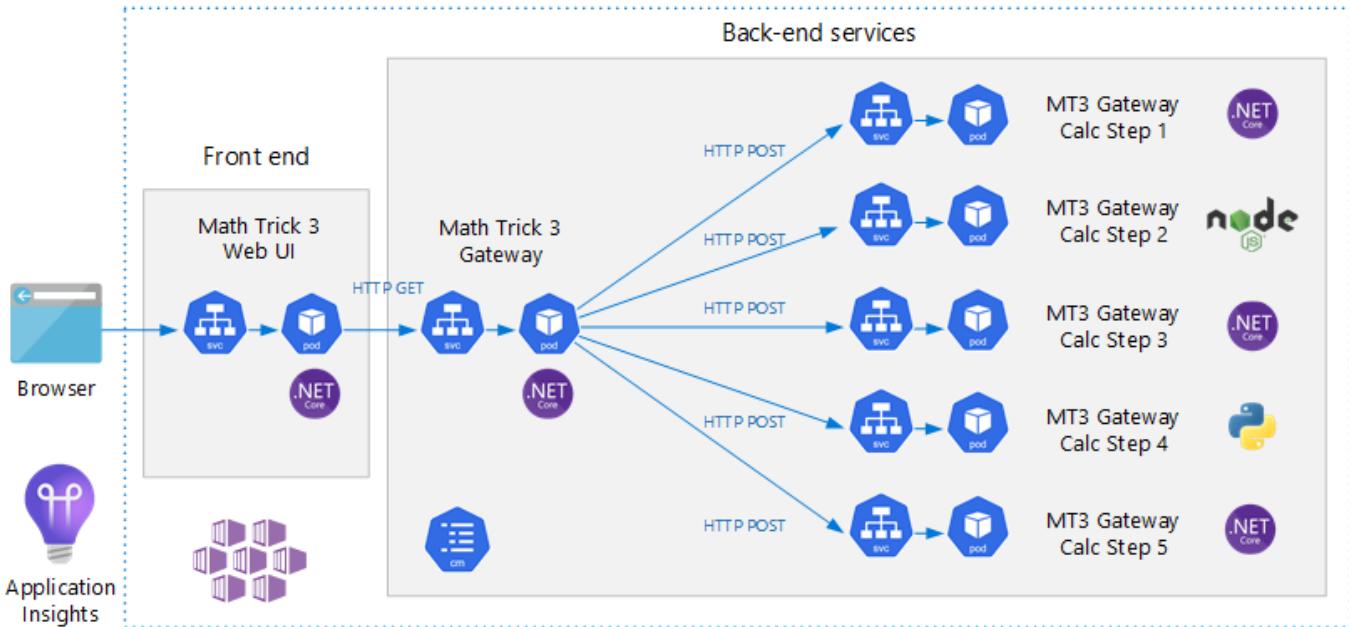
The **Chained** folder contains the files related to the Math Trick application using the **Chained Microservices Design Pattern** (6 microservices total).

## Chained Microservices Pattern - Kubernetes Implementation



The **Gateway** folder contains the files related to the **Gateway Microservices Pattern** implementation of the Math Trick application (7 microservices total).

## Gateway Kubernetes Implementation



## Exercise: Create Additional Azure Resources

In this exercise you will create the external accounts needed to complete the rest of this lab.

### Task 1 - Create an Azure Container Registry

1. Define ACR name.

```
$SUFFIX=(Get-Date -Format "MMddyy")
$ACR_NAME="acr$($YOUR_INITIALS)$($SUFFIX)"
Write-Host "ACR Name: $ACR_NAME"
```

Note the **ACR Name**. You'll use it throughout the rest of this lab

2. Create an Azure Container Registry.

```
$ACR_ID=$(az acr create --sku Premium --name $ACR_NAME --resource-group
$AKS_RESOURCE_GROUP --query id -o tsv)

$ACR_ID
```

### Task 2 - Create an Azure Key Vault

The Azure Key Vault will be used to store the image tags of all the microservices.

While this key vault will not be needed until Module 5, it's a good idea to create it here when all the related variables are available.

- Define variables.

```
$KV_NAME="kv${$YOUR_INITIALS}"
Write-Host "Key Vault Name: $KV_NAME"
```

- Note the **Key Vault Name**. You'll use it in lab 5.

- Create an Azure Key Vault.

```
az keyvault create --name $KV_NAME
--resource-group $AKS_RESOURCE_GROUP
--enable-rbac-authorization false
--sku Standard
```

### Task 3 - Assign permissions to Azure Container Registry

- Get Service Principal Ids from the managed identities of your AKS cluster.

```
$AKS_ID=$(az ad sp list --display-name "$AKS_NAME" --query [0].appId -o tsv)
$AKS_POOL_ID=$(az ad sp list --display-name "$AKS_NAME-agentpool" --query
[0].appId -o tsv)
```

- Assign the AKS cluster **ACR Pull** permissions to your ACR.

```
az role assignment create --role "AcrPull" --assignee $AKS_ID --scope $ACR_ID
az role assignment create --role "AcrPull" --assignee $AKS_POOL_ID --scope $ACR_ID
```

- Verify the assignment in the Portal.

The screenshot shows the Azure portal's Access control (IAM) blade for a container registry. The sidebar on the left has 'Access control (IAM)' selected. The main area displays a table of role assignments under the 'AcrPull' role. The table has columns for 'Name', 'Type', and 'Permissions'. Three entries are listed: 'kiz-kube-sp' (App, AcrPull), 'kiz-kubernetes-sp' (App, AcrPull), and 'kizaks' (App, AcrPull). The 'kizaks' row is highlighted with a red border.

Name	Type	Permissions
kiz-kube-sp	App	AcrPull
kiz-kubernetes-sp	App	AcrPull
<b>kizaks</b>	App	AcrPull

## Exercise: Use Helm to Deploy the Application

In this exercise you will use the Helm package manager to deploy an entire application in a single step, rather than applying individual manifest files.

## Task 1 - Build and push all the images to your ACR

Before you can deploy your application, the images must exist in a container registry.

1. Change the current folder to **MathTrick**.

```
cd C:\k8s\labs\MathTrick
```

2. Login into your ACR.

```
az acr login --name $ACR_NAME --expose-token
```

3. You can use ACR to build and push your images to the Azure Container Registry (itself). Run the script below. This operation will take 10-15 minutes because local base images are not cached between builds.

```
.\buildmt3chainedallacr.ps1 -acrname $ACR_NAME
```

4. Verify all images have been built and pushed. Open your ACR in the Azure Portal.

The screenshot shows the Azure Container Registry (ACR) interface. On the left, there's a sidebar with various navigation options: Overview, Activity log, Access control (IAM), Tags, Quick start, Events, Settings, Access keys, and Encryption. The main area is titled "kizacr | Repositories". It features a search bar at the top left and a "Refresh" button. Below the search bar is a "Search to filter repository" input field. A list of repositories is displayed, including "mt3chained-step1", "mt3chained-step2", "mt3chained-step2-nodejs", "mt3chained-step3", "mt3chained-step4", "mt3chained-step5", and "mt3chained-web".

## Task 2 - Install a Helm Chart in your AKS cluster

1. Change current folder to **C:\k8s\labs\MathTrick\Chained\Helm\mt3chained**.

```
cd C:\k8s\labs\MathTrick\Chained\Helm\mt3chained
```

2. Examine the contents of **values.yaml**. This file defines the parameters Helm will use when it creates a release.

```
repo: kizacr.azurecr.io
namespace: chained
platform: dotnet
tags:
  mt3chainedweb: latest
  mt3chainedstep1: latest
  mt3chainedstep2: latest
  mt3chainedstep2nodejs: latest
  mt3chainedstep3: latest
  mt3chainedstep4: latest
  mt3chainedstep5: latest
```

3. Change the name of the **repo** to your ACR name.

When you install the chart to your cluster, the chart itself is saved to in the current namespace. This is not related to the namespace that's specified in your manifests. You may choose to keep all your charts in the default namespace, regardless of where the resources are deployed.

4. When Helm is used, it looks for the chart in a folder or a registered repo. Go up one folder to the **Helm** folder so the **mt3chained** folder is directly under it and can be specified as the source of the chart.

```
cd ..
```

5. Install the chart to your cluster. The release name doesn't matter. In this case, the chart name is the folder containing all the files and manifests.

```
helm install chaineddemo mt3chained -n default
```

As an alternative, you can use the **upgrade** command to install the chart if it doesn't exist. This allows you to same command for install and upgrade operations, which is very handy in DevOps.

```
helm upgrade chaineddemo mt3chained -n default --install
```

6. Verify the pods have all been deployed to the cluster.

```
kubectl get pods -n chained
```

NAME	READY	STATUS	RESTARTS	AGE
mt3chained-step1-dep-9b8b5b576-z8n46	1/1	Running	0	55s
mt3chained-step2-dep-689bd49fffd-vt2xg	1/1	Running	0	55s
mt3chained-step3-dep-668bcf8f7-xvrvb	1/1	Running	0	55s
mt3chained-step4-dep-95b7f7b49-jghp7	1/1	Running	0	55s
mt3chained-step5-dep-5f7748f995-f8mkh	1/1	Running	0	55s
mt3chained-web-dep-7b95b58486-tp7hf	1/1	Running	0	55s

7. Get the External IP address of the service and open a browser with that address.

```
kubectl get svc -n chained
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
mt3chained-step1-svc	ClusterIP	10.0.2.115	<none>	5010/TCP
mt3chained-step2-svc	ClusterIP	10.0.228.217	<none>	5020/TCP
mt3chained-step3-svc	ClusterIP	10.0.124.81	<none>	5030/TCP
mt3chained-step4-svc	ClusterIP	10.0.0.77	<none>	5040/TCP
mt3chained-step5-svc	ClusterIP	10.0.253.66	<none>	5050/TCP
mt3chained-web-svc	LoadBalancer	10.0.119.159	20.62.158.133	80:31194/TCP

The site should come up in the browser:

Math Trick 3 - Math Trick - Ends X

Not secure 20.62.158.133

# Always Ends with 3 Math Trick

**Chained** [View System Diagram](#)

You pick a number and then perform the calculation steps below. After all the calculations have completed, your final result will always be 3, regardless of the number you picked.

## Process:

Start by picking a number between 1 and 10.

Calculation Steps:

1. Double the number.
2. Add 9 to result.
3. Subtract 3 from the result.
4. Divide the result by 2.
5. Subtract the original number from result.

Final result will always be 3.

## Try it Yourself:

Pick a number:

Final Result: X

Perform Calculations:

Failure Rate: 5%

Calculation Actions Performed:

Platform	Step	Calculation	Result
----------	------	-------------	--------

8. Click the **Start** button. If any of the steps return an error, click the **Start** button again. The results should be returned:

# Try it Yourself:

Pick a number:

Final Result: 3

Perform Calculations:

Failure Rate: 5%

Calculation Actions Performed:

Platform	Step	Calculation	Result
.NET Core	1	$5 \times 2$	10
.NET Core	2	$10 + 9$	19
.NET Core	3	$19 - 3$	16
.NET Core	4	$16 / 2$	8
.NET Core	5	$8 - 5$	3

Task 3 - Use Helm to deploy updates to your application

Use Helm to apply updates to your application when **ANYTHING** changes, without keeping track of those changes.

1. Open the **C:\k8s\labs\MathTrick\Chained\Helm\mt3chained\templates\mt3chained-web-dep.yaml** file in an editor. Go to the labels section under "template" and add another label.

```
template:
  metadata:
    annotations:
      checksum/config: {{ include "mt3chained-web.checksum" . | b64enc }}
    labels:
      tier: mt3chained-web-pod
      mylabel: somevalue
```

2. Upgrade the chart.

```
helm upgrade chaineddemo mt3chained
```

3. Check the pods. Anytime there's a change to the **template** section, the pod is terminated and a new one is created.

```
kubectl get pods -n chained
```

NAME	READY	STATUS	RESTARTS	AGE
mt3chained-step1-dep-9b8b5b576-z8n46	1/1	Running	0	51m
mt3chained-step2-dep-689bd49ffd-vt2xg	1/1	Running	0	51m
mt3chained-step3-dep-668bcf8f7-xvrvb	1/1	Running	0	51m
mt3chained-step4-dep-95b7f7b49-jghp7	1/1	Running	0	51m
mt3chained-step5-dep-5f7748f995-f8mkh	1/1	Running	0	51m
mt3chained-web-dep-5c88c8d54c-x6kch	1/1	Running	0	8s
mt3chained-web-dep-7b95b58486-tp7hf	0/1	Terminating	0	51m

Notice only the single pod has been recreated, while all the others have remained unchanged.

4. Open the **C:\k8s\labs\MathTrick\Chained\Helm\mt3chained\templates\mt3chained-cm.yaml** file in an editor. Change the **FAILURE\_RATE** to a different whole number from 0 to 100. Save the file.
5. Open the **C:\k8s\labs\MathTrick\Chained\Helm\mt3chained\templates\mt3chained-web-dep.yaml** file in an editor. Notice the **annotations** value.

```
template:
  metadata:
    annotations:
      checksum/config: {{ include (print $.Template.BasePath "/mt3chained-cm.yaml") . | sha256sum }}
```

When Helm parses this file, it gets the **sha256sum** value of the ConfigMap file and saves that value here. That means whenever the ConfigMap changes and the Helm chart is upgraded, a new value is written to the annotation. Since the change occurred in the **template** section, the pod will be recreated. Since all the deployments have the same code in the annotations section, as they all dependent on the values in the ConfigMap, all the pods will be recreated when you upgrade the chart after updating the ConfigMap.

6. Upgrade the chart.

```
helm upgrade chaineddemo mt3chained
```

7. Check the pods. Notice all the pods have been recreated:

```
kubectl get pods -n chained
```

NAME	READY	STATUS	RESTARTS	AGE
mt3chained-step1-dep-786ccb6fdd-kk22v	1/1	Running	0	9s
mt3chained-step1-dep-9b8b5b576-z8n46	0/1	Terminating	0	57m
mt3chained-step2-dep-67ff66c594-bpmdv	1/1	Running	0	9s
mt3chained-step2-dep-689bd49ffd-vt2xg	0/1	Terminating	0	57m
mt3chained-step3-dep-5556f5748f-6k4ft	1/1	Running	0	8s
mt3chained-step4-dep-7546c4d789-68zs8	1/1	Running	0	8s
mt3chained-step4-dep-95b7f7b49-jghp7	0/1	Terminating	0	57m
mt3chained-step5-dep-d99c6859-x9wvk	1/1	Running	0	8s
mt3chained-web-dep-846cbd4f75-zgmx6	1/1	Running	0	9s

8. Open the **C:\k8s\labs\MathTrick\Chained\Helm\mt3chained\templates\steps\mt3chained-step2-dep.yaml** file in an editor. Notice the **containers** section.

```
spec:
  containers:
    - name: mt3chained-step2
      {{ if eq .Values.platform "dotnet" }}
      image: {{ .values.repo }}/mt3chained-step2:{{ .values.tags.mt3chainedstep2 }}
      {{ else }}
      image: {{ .values.repo }}/mt3chained-step2-nodejs:{{ .values.tags.mt3chainedstep2nodejs }}
      {{ end}}
```

When Helm renders this manifest file, it uses the value of the **platform** parameter to determine which container to use for this microservice. The default **platform** value is set to **dotnet**, which will cause the **.Net** container to be deployed. If **values.yaml** is updated or you overwrite the value in the command-line with any other value, the **Node JS** container will be used instead.

```
helm upgrade chaineddemo mt3chained --set platform=multi
```

9. Wait for the image to be updated then run the calculations again on the website.

# Try it Yourself:

Pick a number:

Final Result: 3

Perform Calculations: Start

Failure Rate: 10%

Calculation Actions Performed:

Platform	Step	Calculation	Result
.NET Core	1	$5 \times 2$	10
node	2	$10 + 9$	19
.NET Core	3	$19 - 3$	16
.NET Core	4	$16 / 2$	8
.NET Core	5	$8 - 5$	3

Task 4 - Use Helm to control which files are included when the chart is updated

1. Open the **.helmignore** file. Notice it contains a single line. This tells Helm to ignore any file or folder that starts with an underscore (\_).

```
_*
```

2. In the **C:\k8s\labs\MathTrick\Chained\Helm\mt3chained\templates\steps** folder, rename the file **mt3chained-step3-dep.yaml** to **\_mt3chained-step3-dep.yaml**.

3. Upgrade the chart.

```
helm upgrade chaineddemo mt3chained
```

4. Check the pods. Notice step 3 pod is gone:

```
kubectl get pods -n chained
```

NAME	READY	STATUS	RESTARTS	AGE
mt3chained-step1-dep-786ccb6fdd-kk22v	1/1	Running	0	6m41s
mt3chained-step2-dep-67ff66c594-bpmdv	1/1	Running	0	6m41s
mt3chained-step4-dep-7546c4d789-68zs8	1/1	Running	0	6m40s
mt3chained-step5-dep-d99c6859-x9wvk	1/1	Running	0	6m40s
mt3chained-web-dep-846cbd4f75-zgmx6	1/1	Running	0	6m41s

5. In the **C:\k8s\labs\MathTrick\Chained\Helm\mt3chained\templates\steps** folder, rename the file **\_mt3chained-step3-dep.yaml** back to **mt3chained-step3-dep.yaml**.
6. Rename the **steps** folder to **\_steps**.
7. Upgrade the chart.

```
helm upgrade chaineddemo mt3chained
```

8. Check the pods. Notice **ALL** the steps pods are gone. The web front-end remains because its manifest is in the root folder.

```
kubectl get pods -n chained
```

NAME	READY	STATUS	RESTARTS	AGE
mt3chained-web-dep-846cbd4f75-zgmx6	1/1	Running	0	10m

9. Rename the **\_steps** folder back to **steps**. Upgrade the chart. Check the pods. Notice all the pods come back as expected.
10. To uninstall your app as a whole, simply uninstall the release.

```
helm uninstall chaineddemo
```

11. Check the all the resources. In about a minute or two, all the resources will be removed.

```
kubectl get pods -n chained
```

## Exercise: Use Visual Studio 2019 to Debug Microservices Locally with Bridge to Kubernetes

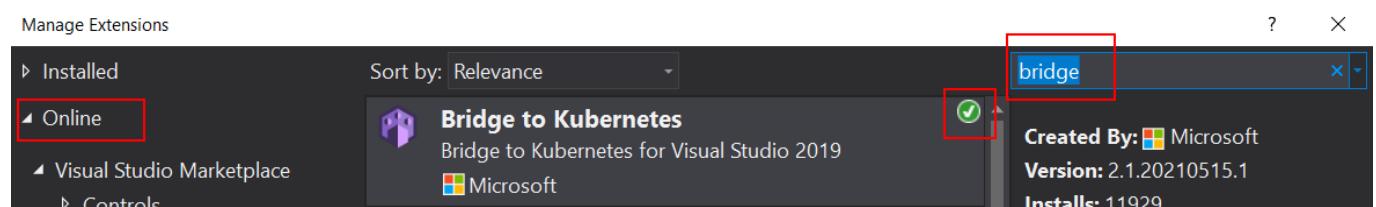
When debugging an application, it's often not practical to run all the services and their dependencies locally. Instead, you can use **Bridge to Kubernetes** to connect to your cluster and swap out only the microservice you're currently working on.

## Task 1 - Prepare your microservice in Visual Studio 2019

1. Reinstall the application if you uninstalled it in the previous task.

```
helm upgrade chaineddemo mt3chained -n default --install
```

2. Open Visual Studio 2019 (can be without code). To verify that **Bridge to Kubernetes** is installed, select **Extensions** and select **Online** list. Type in "bridge" in the search box. If the extension is installed, there will be a green check in the upper right-hand corner.



If it's not installed, go ahead and install it. Restart Visual Studio 2019.

3. Open the **MT3Chained-Step3** solution in the **C:\k8s\labs\MathTrick\Chained\MT3Chained-Step3** folder in Visual Studio 2019.

4. Run the project to make sure it's working correctly.

```
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:5030
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\MIP\ContainersMicroservices\M
d\MT3Chained-Step3
```

Take note of the Port the service is listening on.

5. Stop the debugger.

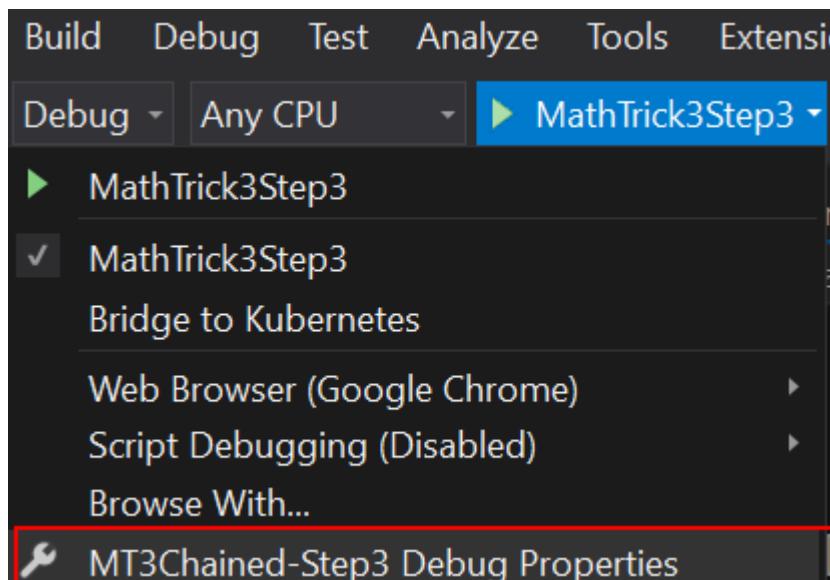
6. Open the **Services -> CalculationService.cs** file. Find the **CalculateStep** method and add 2 breakpoints at the locations indicated on the image:

```
2 references
public List<CalculationStepModel> CalculateStep(int pickedNumber, double currentResult)
{
    var steps = new List<CalculationStepModel>();

    int failureRate = _configuration.GetValue<int>("FAILURE_RATE", 0);
    var rand = new Random();
    if (rand.NextDouble() < (failureRate / 100.0))
    {
        throw new Exception("Some random problem occurred.");
    }
    // Perform current step
    // Step 3 - Subtract 3 from the result.
    int subtrahend = _configuration.GetValue<int>("CalcStepVariable", 3);
    double previousResult = currentResult;
    currentResult = currentResult - subtrahend;
```

Task 2 - Debug your microservice using Visual Studio 2019 and the Bridge to Kubernetes extension

1. Open the Profile option dropdown and click on **MT3Chained-Step3 Debug Properties**.



2. Select **Bridge to Kubernetes** from the Profile dropdown. Click the **Change** button.

Profile: **Bridge to Kubernetes**

Launch: **Bridge to Kubernetes**

Launch browser:

Bridge to Kubernetes

AKS Cluster:

Namespace:

Service:

Local Launch Profile:

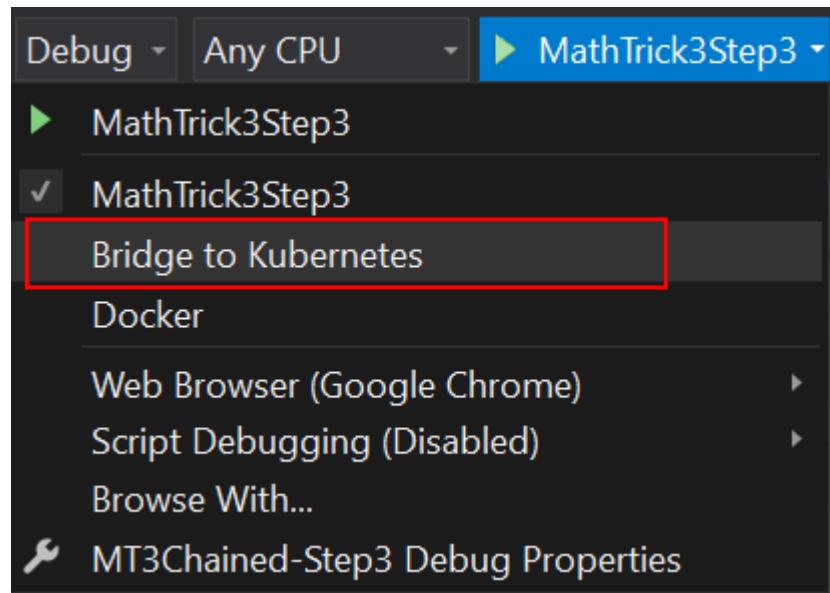
**Change...**

3. Select the cluster, namespace (**chained**) and service name (**mt3chained-svc**). Click the OK button when done configuring the parameters.

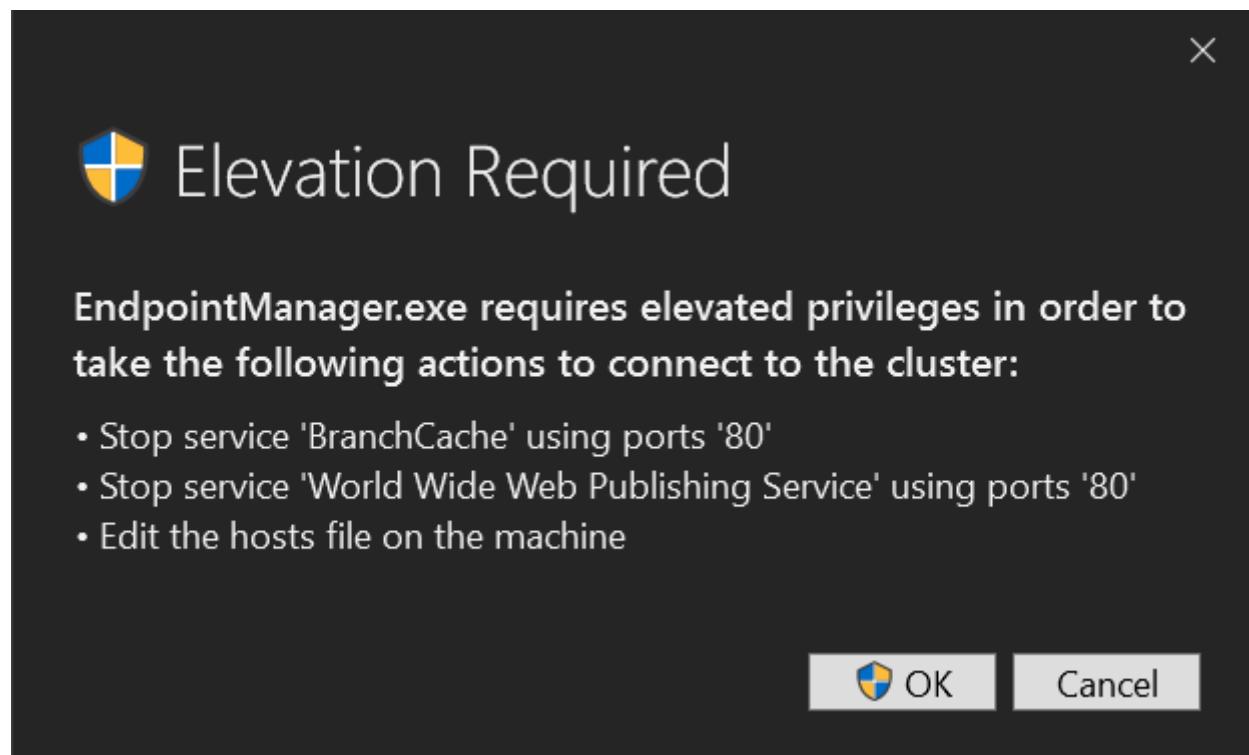
## Edit profile for Bridge to Kubernetes

Cluster source	Local Kube Config	Refresh
 arn:aws:eks:us-east-1:270376981755:cluster/atlas	https://0B59F07BDA3849FAB5D0E37AD9743E8E.sk1.us-east-1.eks.amazonaws.com	Namespace chained
 atlas-prod	https://CFDC64441FB5F28868B0A45F26B4E9BD.gr7.us-east-1.eks.amazonaws.com	Service mt3chained-web-svc
 docker-desktop	https://kubernetes.docker.internal:6443	<input type="checkbox"/> Enable routing isolation Only redirect requests originating from "d9f7" subdomain to your machine (DNS propagation). <a href="#">Learn about routing isolation</a>
 kizaks	https://kizaks-dns-7ae09179.hcp.eastus.azmk8s.io:443	Application url <a href="http://20.62.158.133">http://20.62.158.133</a>
 kizsamplesaks	https://kizsamplesaks-dns-b490220c.hcp.eastus.azmk8s.io:443	

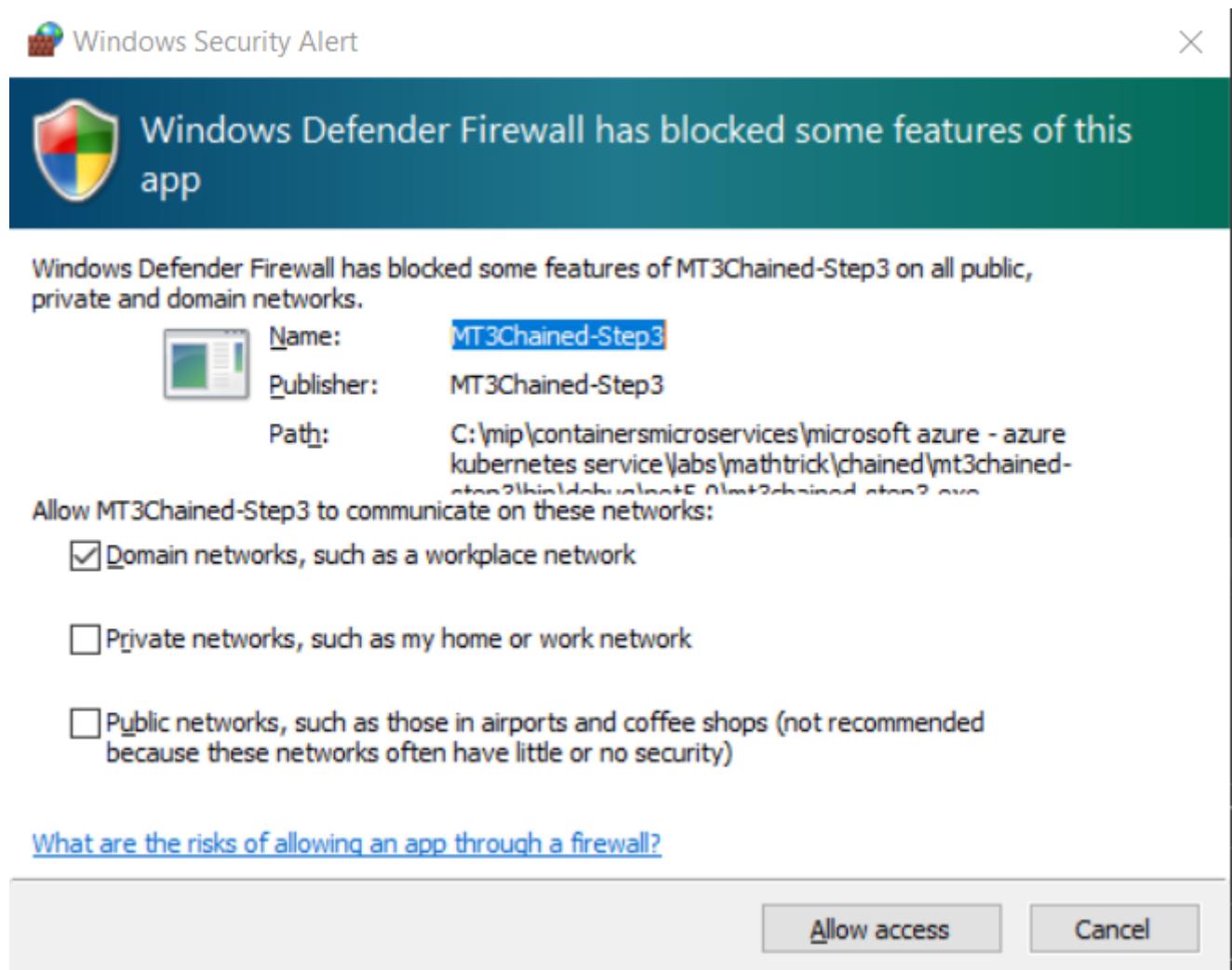
4. Select the **Bridge to Kubernetes** from the Debug profile dropdown and run it.



5. An **Endpoint Manager** dialog will be displayed requesting elevated privilege to update your **hosts** file. Click the **OK** button.

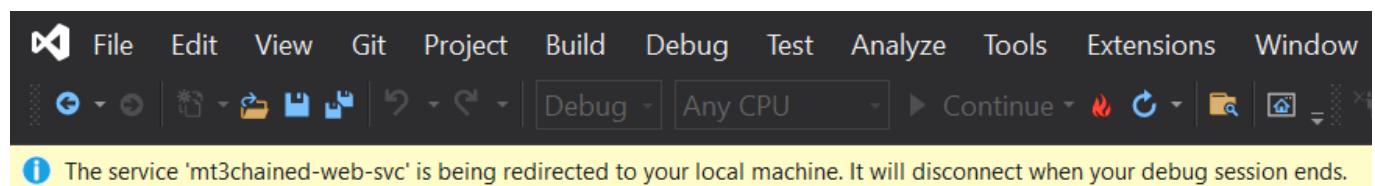


6. You may see a Windows Defender Firewall message. Click the **Allow access** button.



7. **Bridge to Kubernetes** will attempt to connect to your cluster. It may not work the first time, so please try again.

8. When you see the cluster has been connected, you are ready for debugging.



9. Watch the image in your deployment and know when Bridge to Kubernetes replaced it with its own.

```
kubectl get deploy mt3chained-step3-dep -w -n chained -o=jsonpath='{$.spec.template.spec.containers[:1].image}'
```

10. Get the External IP of your Web service. Open a browser window with that IP. The site should come up as normal.

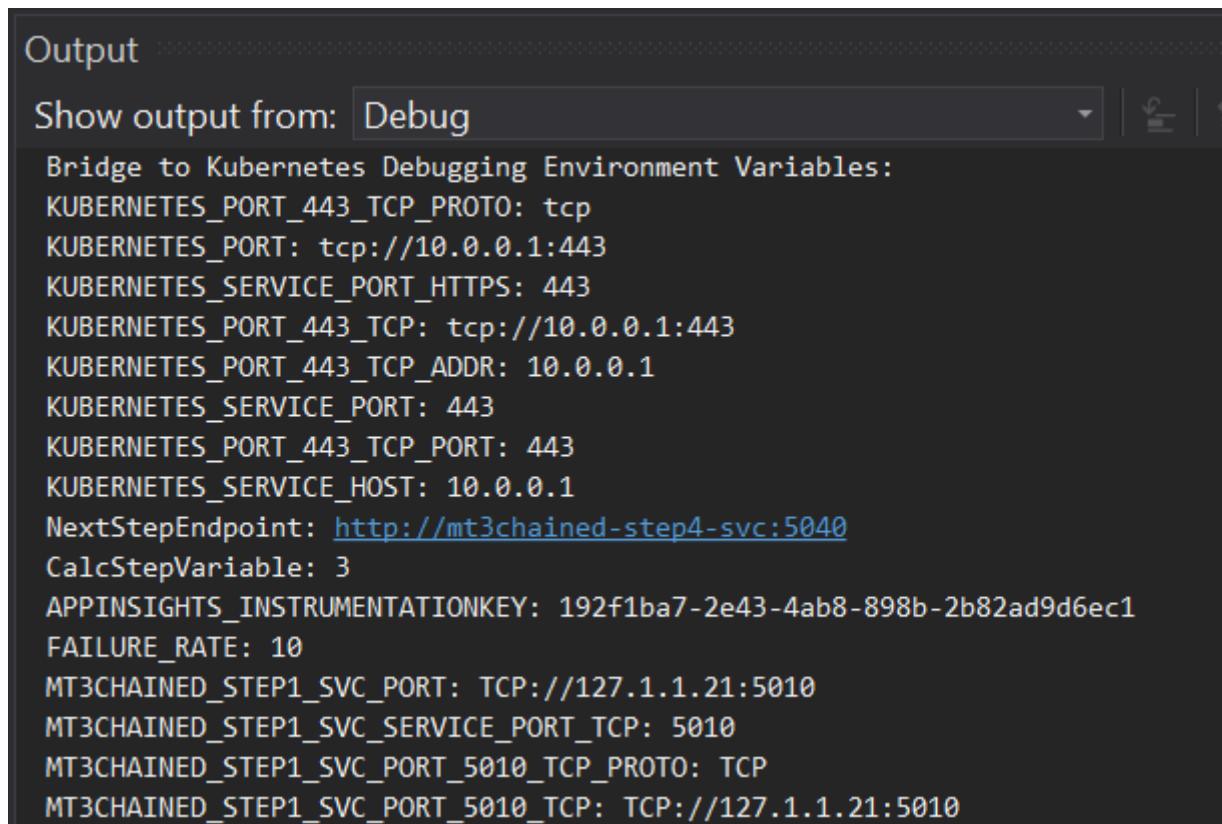
```
kubectl get svc -n chained
```

11. Click the **Start** button. Breakpoint in your code should be triggered:

```
public List<CalculationStepModel> CalculateStep(int pickedNumber, double currentResult)
{
    var steps = new List<CalculationStepModel>();

    int failureRate = _configuration.GetValue<int>("FAILURE_RATE", 0);
    var rand = new Random();
    if (rand.NextDouble() < (failureRate / 100.0))
    {
        throw new Exception("Some random problem occurred.");
    }
}
```

12. Open the **Output** window and take note of the environment variables from the pod have been transferred to your local machine.

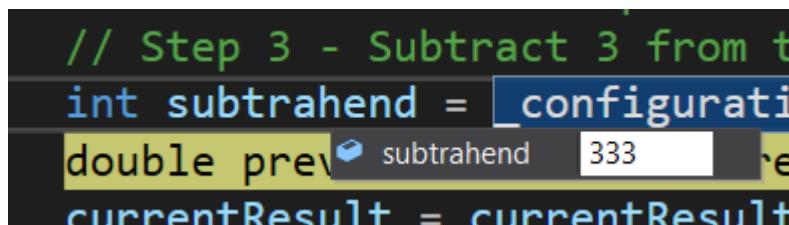


The screenshot shows the 'Output' tab in Visual Studio Code. The title bar says 'Output'. Below it, a dropdown menu says 'Show output from: Debug'. The main area displays a list of environment variables:

- Bridge to Kubernetes Debugging Environment Variables:
- KUBERNETES\_PORT\_443\_TCP\_PROTO: tcp
- KUBERNETES\_PORT: tcp://10.0.0.1:443
- KUBERNETES\_SERVICE\_PORT\_HTTPS: 443
- KUBERNETES\_PORT\_443\_TCP: tcp://10.0.0.1:443
- KUBERNETES\_PORT\_443\_TCP\_ADDR: 10.0.0.1
- KUBERNETES\_SERVICE\_PORT: 443
- KUBERNETES\_PORT\_443\_TCP\_PORT: 443
- KUBERNETES\_SERVICE\_HOST: 10.0.0.1
- NextStepEndpoint: <http://mt3chained-step4-svc:5040>
- CalcStepVariable: 3
- APPINSIGHTS\_INSTRUMENTATIONKEY: 192f1ba7-2e43-4ab8-898b-2b82ad9d6ec1
- FAILURE\_RATE: 10
- MT3CHAINED\_STEP1\_SVC\_PORT: TCP://127.1.1.21:5010
- MT3CHAINED\_STEP1\_SVC\_SERVICE\_PORT\_TCP: 5010
- MT3CHAINED\_STEP1\_SVC\_PORT\_5010\_TCP\_PROTO: TCP
- MT3CHAINED\_STEP1\_SVC\_PORT\_5010\_TCP: TCP://127.1.1.21:5010

13. Click the **Continue** button to go to the next breakpoint.

14. Hover the mouse over the **subtrahend** variable and click on it to change its current value to **333**. Press the **Enter** key.



The screenshot shows the code in the editor with a breakpoint. The variable **subtrahend** is highlighted in yellow. A tooltip shows its current value as **333**. The code snippet is:

```
// Step 3 - Subtract 3 from t
int subtrahend = _configuration.GetValue<int>("SUBTRAHEND", 0);
double previousSubtrahend = subtrahend;
currentResult = currentResult - subtrahend;
```

15. Click the **Continue** button. The service will finish processing locally and then call the next service in the chain, passing it the incorrect value. When the process finishes, the final results online will be incorrect.

Pick a number:  ▼

Final Result: X

Perform Calculations: Failure Rate: 10%

Start

Calculation Actions Performed:

Platform	Step	Calculation	Result
	1	$5 \times 2$	10
	2	$10 + 9$	19
	3	$19 - 333$	-314
	4	$-314 / 2$	-157
	5	$-157 - 5$	-162

16. Your service replaced the microservice on the cluster and performed a different calculation. This was done without running the other services locally.
17. Stop your debug session. [Bridge to Kubernetes](#) will restore the previous image in the pod and the app will function as usualy.
18. Return to your browser and click the **Start** button again. You'll see the calculations are once again correct.

## Exercise: Use VS Code to Debug Microservices Locally with Bridge to Kubernetes

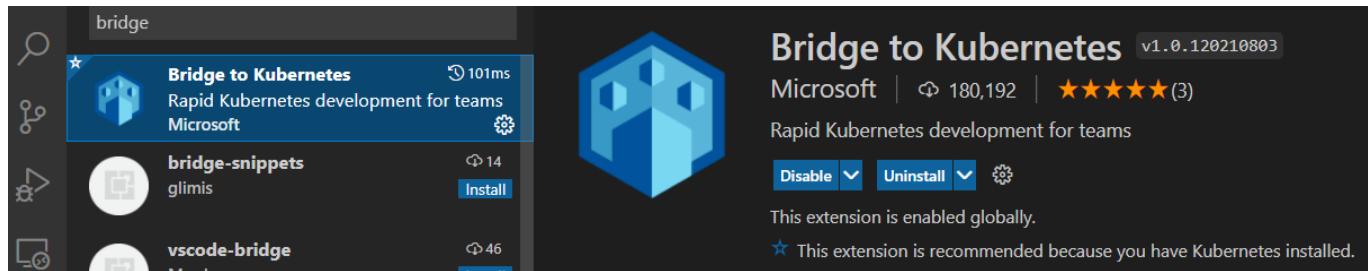
When debugging an application, it's often not practical to run all the services and their dependencies locally. Instead, you can use **Bridge to Kubernetes** to connect to your cluster and swap out only the microservice you're currently working on.

### Task 1 - Prepare your microservice in VS Code and verify that it runs locally

1. Reinstall the application if you uninstalled it in the previous task.

```
helm upgrade chaineddemo mt3chained -n default --install
```

2. Open VS Code. To verify that **Bridge to Kubernetes** is installed, select **Extensions** and select the **Bridge to Kubernetes** extension.



If the extension is not installed, please install it. Restart VS Code.

3. Open the **C:\k8s\labs\MathTrick\Chained\MT3Chained-Step2-NodeJS** folder.

4. Run the microservice locally. In the **Debug Console** window, notice the local port the microservice is listening to. This will be used for redirection.

```
C:\Program Files\nodejs\node.exe .\server.js
Started on PORT: 3000
```

5. Stop the debugger.
6. Open **server.js** and set a breakpoint at the following line:

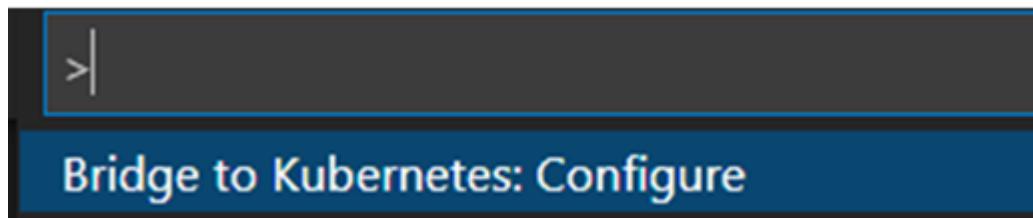
```
74 | let addend = Number(process.env.CalcStepVariable) || 9;
● 75 | let previousResult = currentResult;
76 | currentResult = currentResult + addend;
```

## Task 2 - Debug the microservice using VS Code and the Bridge to Kubernetes extension

1. The **Bridge to Kubernetes** extension in VS Code requires that the current namespace be set to the namespace you're working in. Cluster to use the correct namespace.

```
kubectl config set-context --current --namespace chained
```

2. Open the **Command palette** (Ctrl+Shift+p) and run the "Bridge to Kubernetes: Configure"



3. Select the **mt3Chained-step2** service.

Choose a service to redirect to your machine

mt3chained-step1-svc

mt3chained-step2-svc

mt3chained-step3-svc

4. Enter **3000** as the port to redirect traffic to.

3000

Enter your local port such as 80, or 0 if traffic

5. Select the existing configuration you tested with earlier. Bridge to Kubernetes will use this as the template for its configuration.

Choose the launch configuration to use to run your application

Launch NodeJS Program

Choose this launch configuration if it is the one you want to use

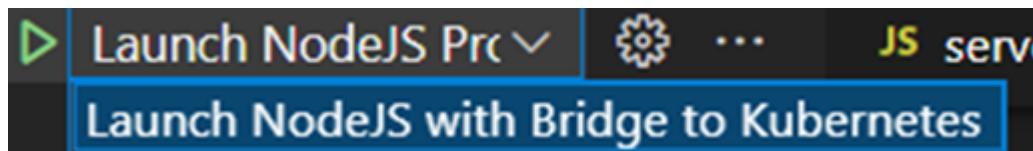
6. Select **No** for the Isolation mode.

Isolate your local version of "mt3chained-step2-svc"

No

Redirect all incoming requests to your machine,

7. From the debug menu, run the new configuration:



8. Answer **Yes** to any elevated privilege requests from **Endpoint Manager**.

When Bridge to Kubernetes comes up, it shows the container port being redirected to the local port. All the other services in the namespace are redirected to local addresses (so that Endpoint Manager

can send traffic back to the cluster). The bar at the bottom turn orange and it shows the currently active connection.

```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

Preparing to run Bridge To Kubernetes configured as pod chained/
Connection established.
Hosts file updated.
Service 'mt3chained-step2-svc' is available on 127.1.1.1:5020.
Service 'mt3chained-step3-svc' is available on 127.1.1.2:5030.
Service 'mt3chained-step4-svc' is available on 127.1.1.3:5040.
Service 'mt3chained-step1-svc' is available on 127.1.1.4:5010.
Service 'mt3chained-web-svc' is available on 127.1.1.5:80.
Service 'mt3chained-step5-svc' is available on 127.1.1.6:5050.
Container port 80 is available at localhost:3000.
#####
Environment started. #####
Run C:\Users\makizhne\AppData\Local\Temp\tmp-10104Fdz0fAb4WmHk.e

Terminal will be reused by tasks, press any key to close it.

```

> OUTLINE  
 > NPM SCRIPTS

⑧ 0 △ 0 ➔ Launch NodeJS with Bridge to Kubernetes (MT3Chained-Step2-NodeJS) kizaks default ⚡ Kubernetes:

9. Watch the image in your deployment to know when **Bridge to Kubernetes** replaced it with its own and when it puts the original one back.

```
kubectl get deploy mt3chained-step3-dep -w -n chained -
o=jsonpath='{$.spec.template.spec.containers[:1].image}'
```

10. Get the **External IP** of your Web service. Open a browser window to that IP. The site should come up as normal.

```
kubectl get svc -n chained
```

11. Click the **Start** button. Breakpoint in your code should be triggered:

VARIABLES	
Block	69
addend: 9	70
currentResult: 10	71
nextStep: undefined	72
pickedNumber: 5	73
previousResult: u... ● 75	74
this: undefined	76
total: 15	77

```

}
var req = request.body;
// Perform current step
// Step 2 - Add 9 to result
let pickedNumber = Number(req.pickedNumber);
let currentResult = Number(req.currentResult);
let addend = Number(process.env.CalcStepVariable) || 9;
let previousResult = currentResult;
currentResult = currentResult + addend;
return data;

```

12. Double click on the **addend** field on the left and change the value from **9** to **99**.

A screenshot of a debugger interface showing the 'VARIABLES' section. The 'Block' node is expanded, displaying the following variable values:

- addend: 99
- currentResult: 10
- nextStep: undefined
- pickedNumber: 5
- previousResult: undefined
- this: undefined

13. Click the **Continue** button in the debugger.
14. Return to the website page and confirm that the incorrect value was passed to other services and that the final result is wrong.

## Try it Yourself:

Pick a number:

Final Result: **48**

Perform Calculations: **Start**

Failure Rate: **10%**

Calculation Actions Performed:

Platform	Step	Calculation	Result
.NET Core	1	$5 \times 2$	10
node	2	$10 + 99$	109
.NET Core	3	$109 - 3$	106
.NET Core	4	$106 / 2$	53
.NET Core	5	$53 - 5$	48

15. Stop the debugger. Bridge to Kubernetes will now restore the previous image in the pod and the app will function as usually.

16. Return to your browser and click the **Start** button again. You'll see the calculations are once again correct.
17. Restore your cluster back to the **default** namespace.

```
kubectl config set-context --current --namespace default
```

#### NOTES:

If you remember from the Helm section, setting the value of **platform** at the command line forces the **Node JS** container to be deployed for **step 2** instead of the default **.Net** version. However, no mention was made here about using a different platform value before running this debug session with the **NodeJS** service. Why?

How could you debug a Node JS microservice locally when a Node JS container isn't installed in the cluster prior to the session? It's because it doesn't matter what is installed in the cluster. The previous container could have been based on **ANY IMAGE WHATSOEVER** before it was replaced.

Bridge to Kubernetes replaces the container with its **routing manager** image for the duration of the debugging session. Then, when the session is over, it does a **rollout undo** to restore the original image back to the Deployment. At no time does Bridge to Kubernetes know (or care) what the previous container image was.

When creating a new microservice, you can connect your local IDE to a cluster before even creating the **Dockerfile** for your microservice. Then test how the service will work **IF** you made it into a container and **IF** you deployed that container to the cluster. Simply create a **placeholder** deployment and service to deploy to the cluster and have Bridge to Kubernetes replace its container with the **routing manager** when you're debugging locally.

As you can tell, Bridge to Kubernetes can greatly enhance your Kubernetes microservice development and debugging experience.

## Exercise: Add Monitoring to Entire Application

Azure Application Insights can be configured to add monitoring across your entire application.

To integrate Application Insights into the code, follow these steps:

- Install the SDK for your platform : <https://docs.microsoft.com/en-us/azure/azure-monitor/app/platforms>
- Create an Application Insights instance in Azure or get the Instrumentation Key of an existing one.
- Configure the app to use the Instrumentation Key, either via a settings file or set it to known **APPINSIGHTS\_INSTRUMENTATIONKEY** environment variable (all SDKs automatically look for this variable).
- Start/initialize the Application Insights process in your app.

That's it! When the application runs, the SDK will collect all types of metrics and log data and automatically send it to an Application Insights instance in Azure.

Task 1 - Create an Application Insights and configure all the microservices to use it

If you're installing both versions of the sample app (Chained and Gateway), make sure you create a separate Application Insights instance for each one.

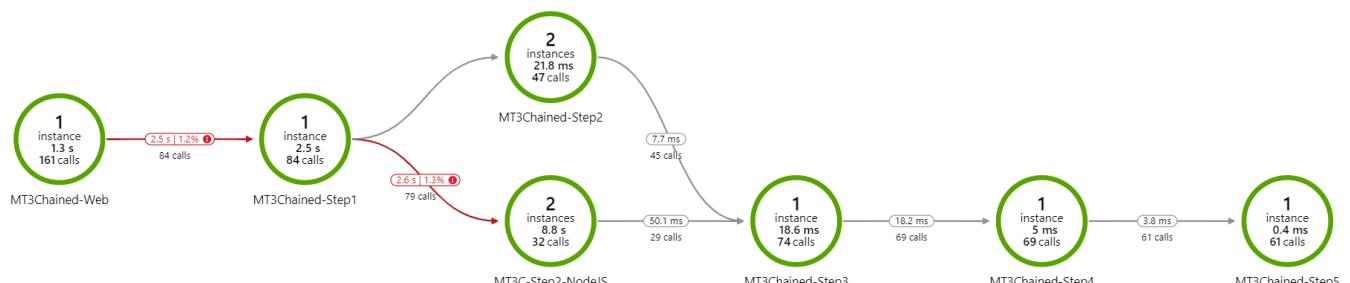
1. Open the Azure portal.
2. Search for Application Insights.
3. Click the **Create** button. Give the instance a meaningful name, like `mt3chained-ai`.
4. When it's ready, go to the instance and take note of the **Instrumentation Key**

5. Open the `mt3chained-cm.yaml` file in the `C:\k8s\Labs\MathTrick\Chained\Helm\mt3chained\templates` folder.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mt3chained-config
  namespace: {{ .values.namespace }}
data:
  FAILURE_RATE: "10"
  APPINSIGHTS_INSTRUMENTATIONKEY: "192f1ba7-2e43-4ab8-898b-2b82ad9d6ec1"
```

Since all the microservices reference the same ConfigMap, they'll all have access to this environment variable. All the microservices have the SDK installed and running.

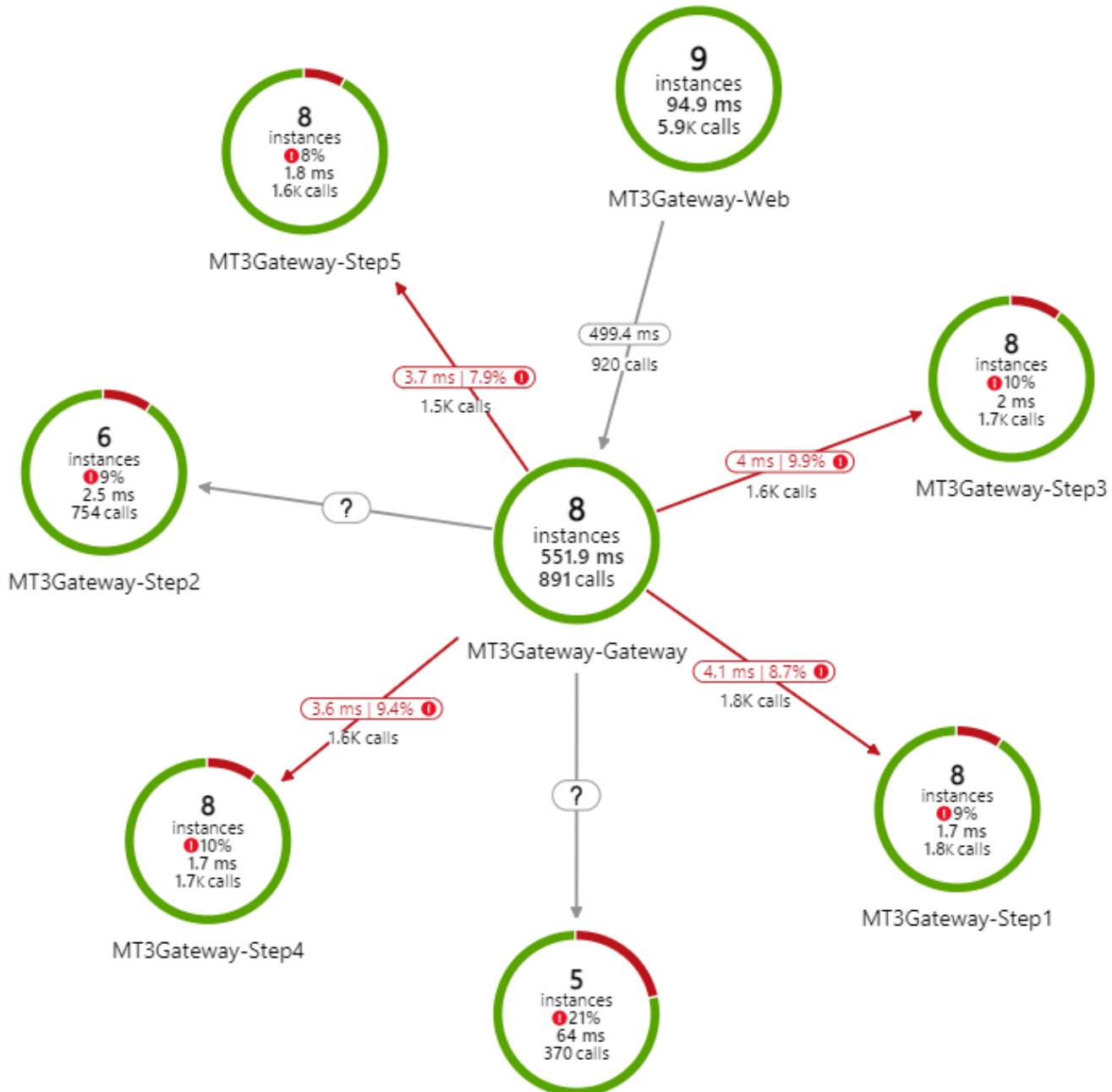
6. Upgrade the Helm chart.
7. Use your app for a few minutes. Open the Application Insights instance in Azure.
8. Select **Application Map** from the side menu. The diagram will look something like this, especially if multiple platforms were deployed.



The Application Map is automatically generated based on the structure of the application. It's used to verify that the application flow is correct.

The "2" instances for **Step 2** are the result of using [Bridge to Kubernetes](#) to connect/disconnect from your cluster. Every time it disconnects and restores the old image, Kubernetes create a new instance of the pod, with a different IP. AI picks that up as a different instance.

9. If the **Gateway** pattern was used, the map will look something like this:



DO NOT DELETE ANY OF THE AZURE RESOURCES YOU CREATED IN THIS LAB!

YOU WILL USE THE SAME ACR AND AKS CLUSTER IN LAB 5