Microsoft

Kubernetes Technical Briefing

Module 4:
Application Development

Agenda

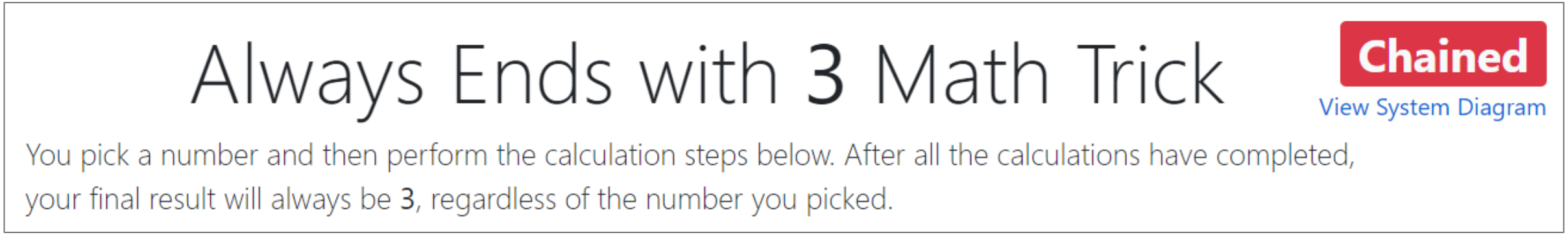
- Microservices Design Patterns
- Microservices with Azure and AKS
  - Azure Container Registry
- Bridge To Kubernetes
  - Visual Studio
  - Visual Studio Code
- Application Monitoring

Microservices Design Patterns

1

Example Microservices Project

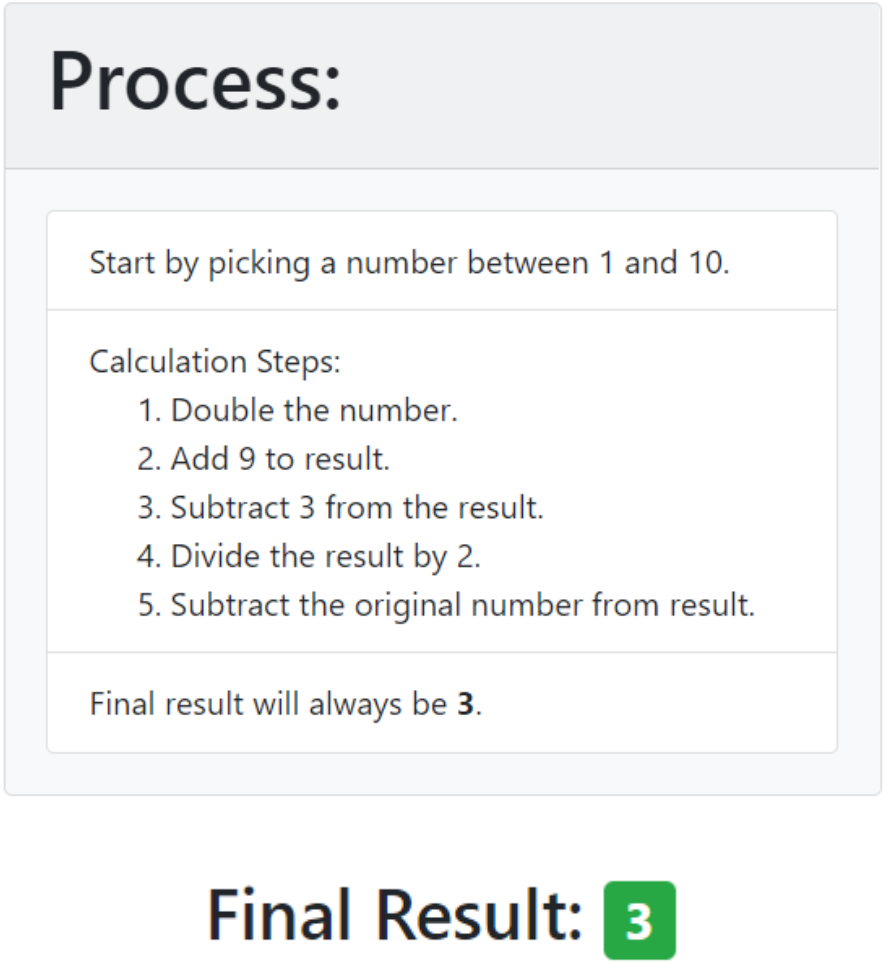
- The sections in this module use an example Microservices project to demonstrate various design, deployment and debugging concepts.



- The example project implements individual microservices to calculate the steps of the math trick.

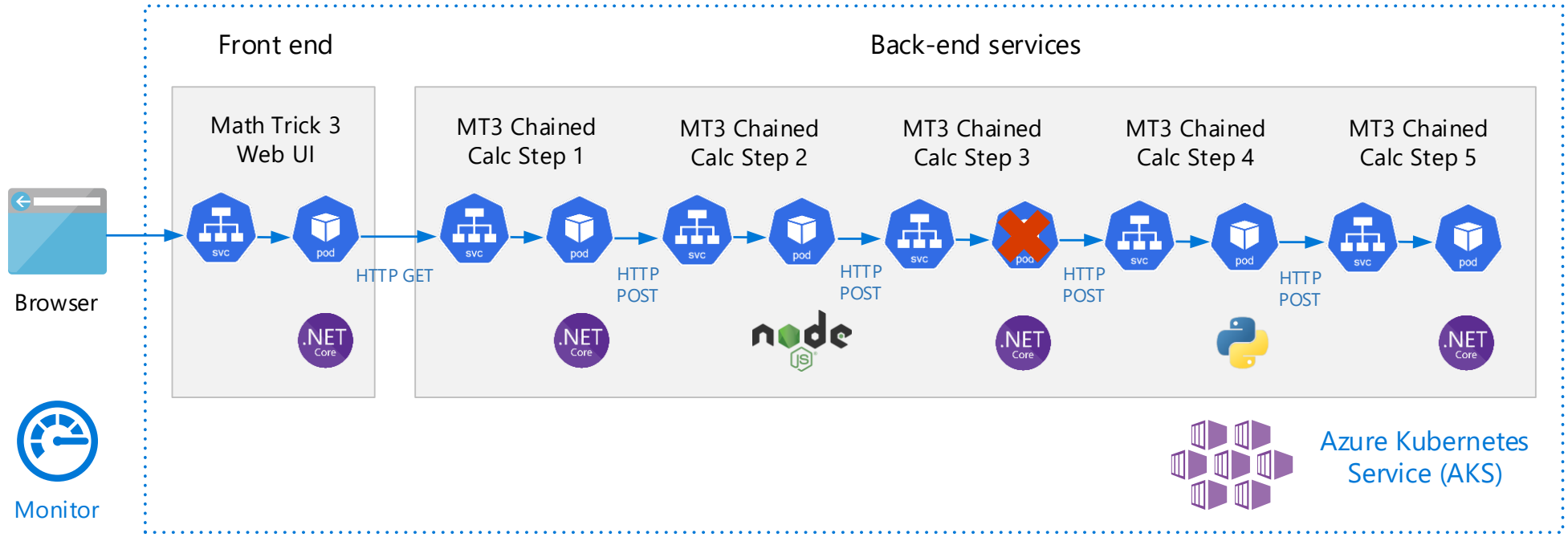
The Math Trick Web UI and Rules

- A user is asked to pick a number between 1 and 10.
- The application calls microservices to perform the calculations.
- Each microservice performs a single calculation and returns the result it achieves.
- All the calculations and results are combined and displayed on the calling web page.
- If all the microservices perform their calculations correctly, the ***final*** result will ALWAYS be **3**.
- As an extra twist, each service could fail at random times due to a preset ***Failure Rate***.



Microservice Design Pattern - Chained

There are two versions of the microservices project. The first implements the Chained Pattern, by having each microservice call the next microservice in the "chain".



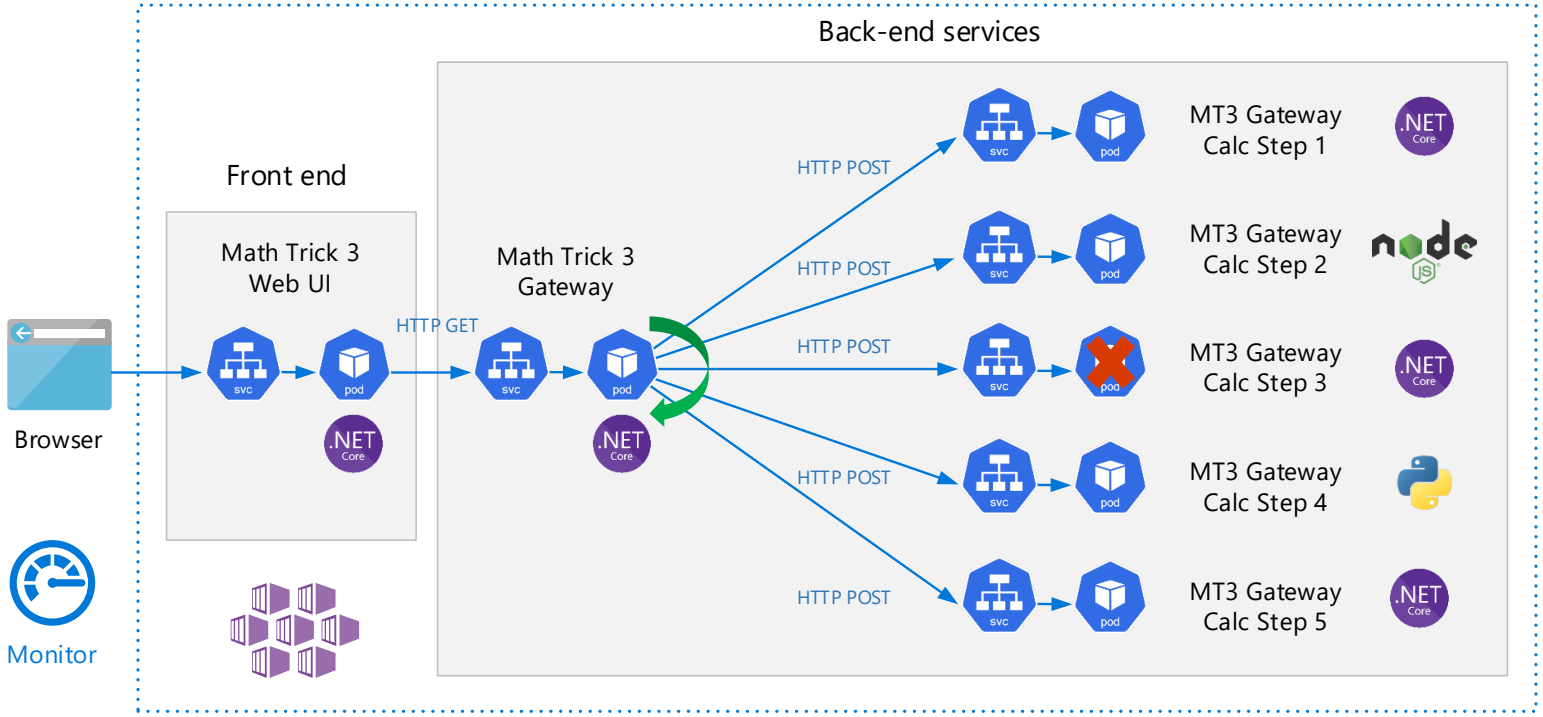
A failure in any microservice results in the entire process failing.

Chained Design Pattern Demo

A set of chained microservices are called in a chain to produce a final result

Microservice Design Pattern - Gateway

The second version implements the Gateway Pattern, by having a gateway microservice call each calculation microservice in a preset order.



The gateway implements retry logic when it detects failures in calculation steps.

Gateway Design Pattern Demo

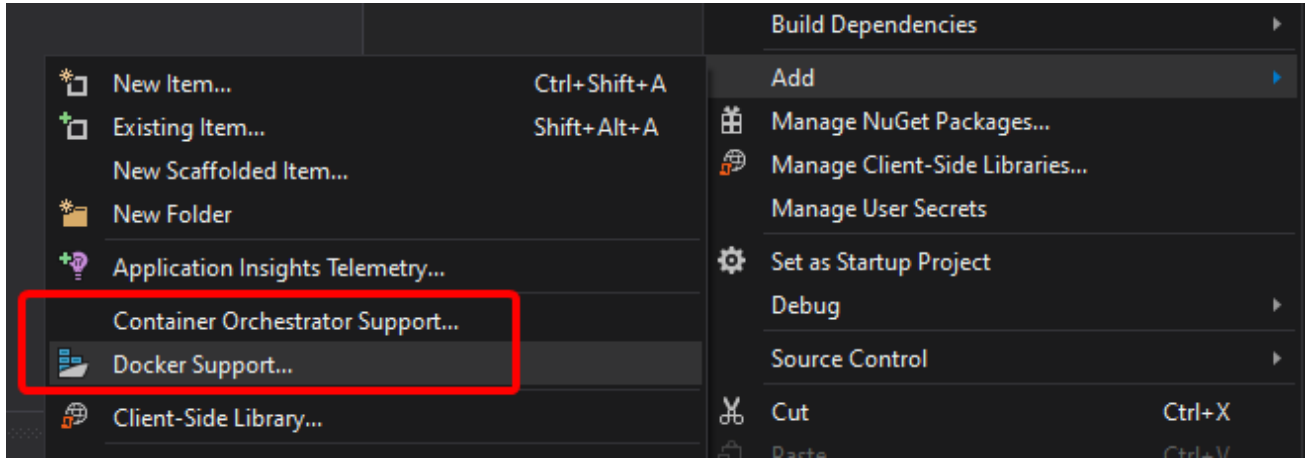
Calls to a set of microservices is coordinated from a single endpoint

Microservices with Azure and AKS

2

Visual Studio Tools for Docker

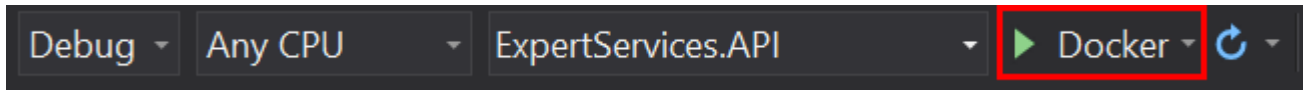
- Visual Studio 2019 provides built-in support for containerizing your .Net Core microservices applications into Docker images.
- A customized Dockerfile is created and added to your project.
- When Docker for Windows/Mac is running on your development machine, Visual Studio can build and run your container and attached the VS Debugger to your running container.




```
#See https://aka.ms/containerfastmode to understand how Visual Studio uses this Dockerfile to build
  images for faster debugging.

FROM mcr.microsoft.com/dotnet/aspnet:3.1 AS base
WORKDIR /app
EXPOSE 80

FROM mcr.microsoft.com/dotnet/sdk:3.1 AS build
WORKDIR /src
COPY ["ExpertServices.API/ExpertServices.API.csproj", "ExpertServices.API/"]
COPY ["ExpertServices.Core/ExpertServices.Core.csproj", "ExpertServices.Core/"]
COPY ["ExpertServices.Repository/ExpertServices.Repository.csproj", "ExpertServices.Repository/"]
RUN dotnet restore "ExpertServices.API/ExpertServices.API.csproj"
COPY . .
WORKDIR "/src/ExpertServices.API"
RUN dotnet build "ExpertServices.API.csproj" -c Release -o /app/build
```



Azure Container Registry

- When you're ready to push your images to a container registry, you can create an account on Docker Hub or create a private container registry in Azure.
- Azure offers a private container registry as a service called ***Azure Container Registry (ACR)***.
- Azure Container Registry handles private Docker container images as well as related content formats, such as Helm charts.
- When your AKS cluster has **AcrPull** permission to an ACR registry, you don't need to specify a Docker Secret in your YAML files.
- You can also use ACR to *build and push* your images, without needing a local installation of Docker.

```
az acr build -t sample/hello-world:{{.Run.ID}} -r MyRegistry .
```

Azure Container Registry Demo

Use Azure Container Registries to store your Docker images
and other cluster related artifacts

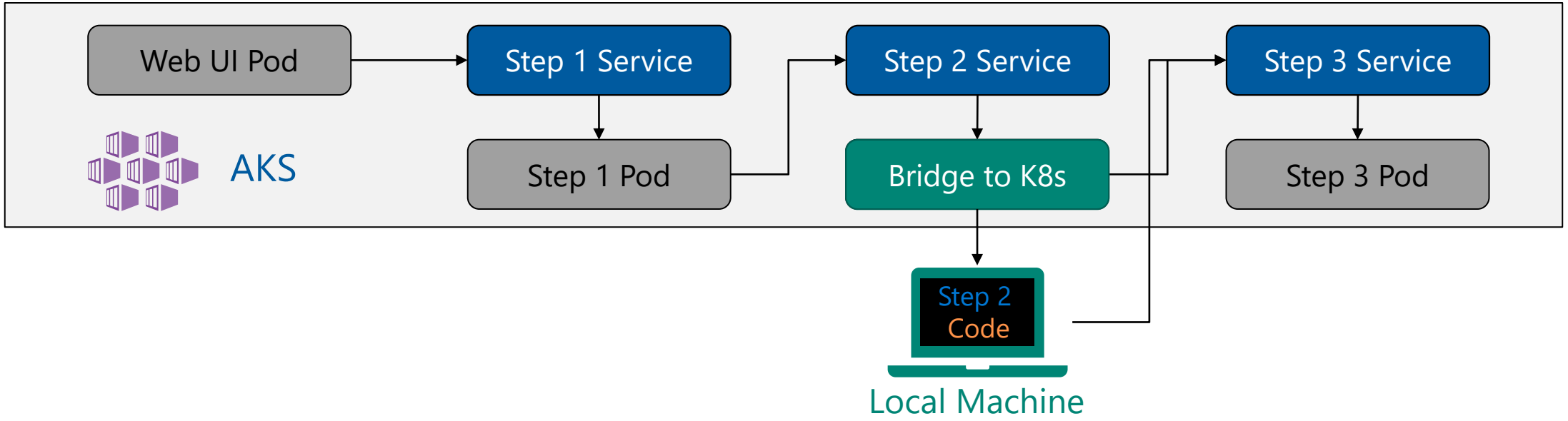
Bridge To Kubernetes

Bridge to Kubernetes

- When working on a large application, it's challenging to debug individual microservices once the application has been deployed to a cluster.
- It's not practical to run all the microservices locally and accurately replicate application dependencies like ingress, service, volumes, etc.
- Bridge to Kubernetes allows you debug individual microservices on your local development machine, while interacting with related microservices as they continue to run in the cluster.
- Bridge to Kubernetes replicates environment variables and configures service routing, making it seem as if you are a part of the cluster while you're running and debugging your microservice locally in Visual Studio or VS Code.

Bridge to Kubernetes

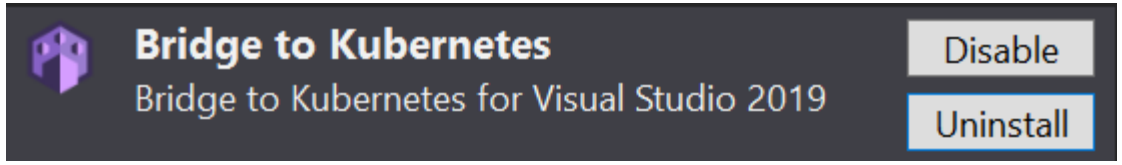
- Bridge to Kubernetes redirects traffic for a microservice to your development computer and then redirects traffic from it back to the cluster.



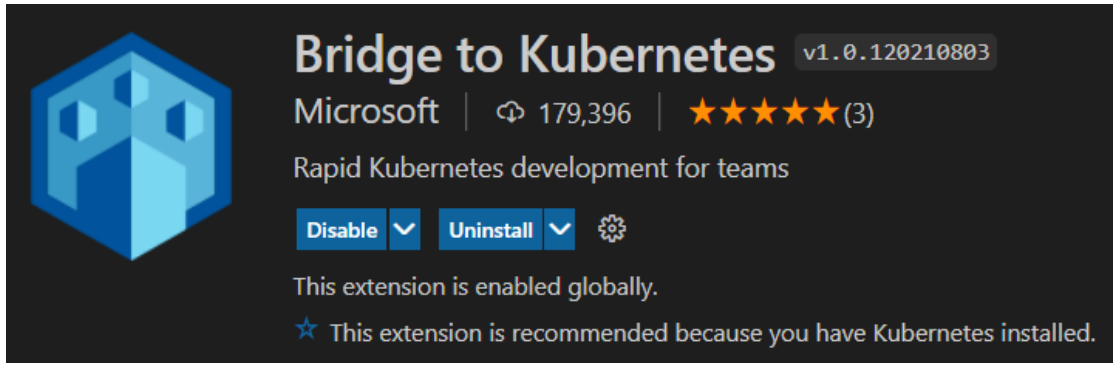
- When you disconnect from your cluster, Bridge to Kubernetes automatically reverts your cluster back to its previous state.

Bridge to Kubernetes Installation and Limitations

- Bridge to Kubernetes is available as an ***extension*** for Visual Studio 2019 and VS Code.
- You can redirect microservices that have only a ***single pod*** and that ***pod may only have a single container***.
- Bridge to Kubernetes only support Linux containers.
- The Bridge to Kubernetes extension needs elevated permissions to run on your local computer in order to be able to update your *hosts* file.
- Bridge to Kubernetes is not compatible with Azure Dev Spaces.



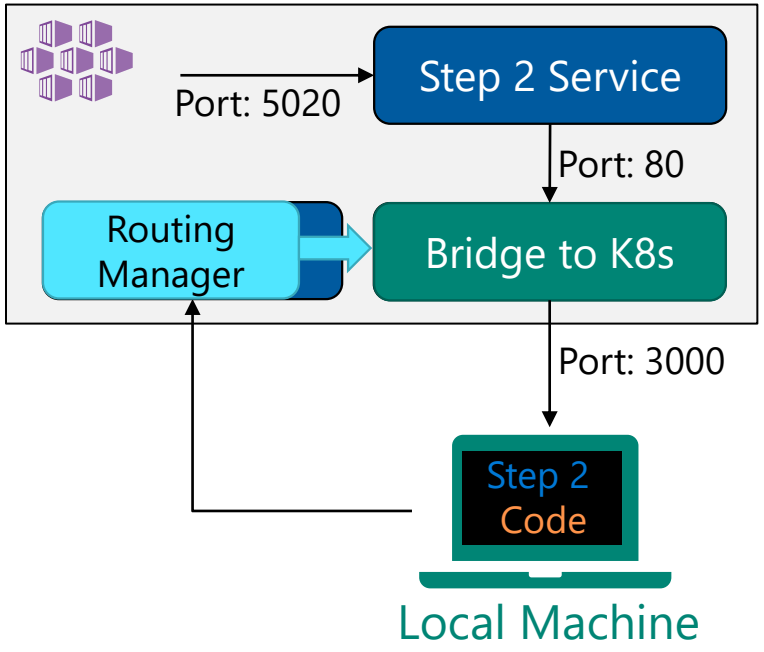
Visual Studio 2019 Extension



VS Code Extension

How Bridge to Kubernetes Works

- Bridge to Kubernetes installs a ***Routing Manager*** app as a deployment in your cluster.
- When the routing manager receives connection request from your local machine, it replaces the container in the target pod with its own container.
- When the new container receives traffic, it forwards that traffic to your local machine on the configured port. Your microservice must handle requests on that port.
- As part of the connection, an ***Endpoint Manager*** app runs on your machine and configures your *hosts* file. This app requires elevated privileges.
- When your local code makes requests to other services, those requests are routed back to the cluster.

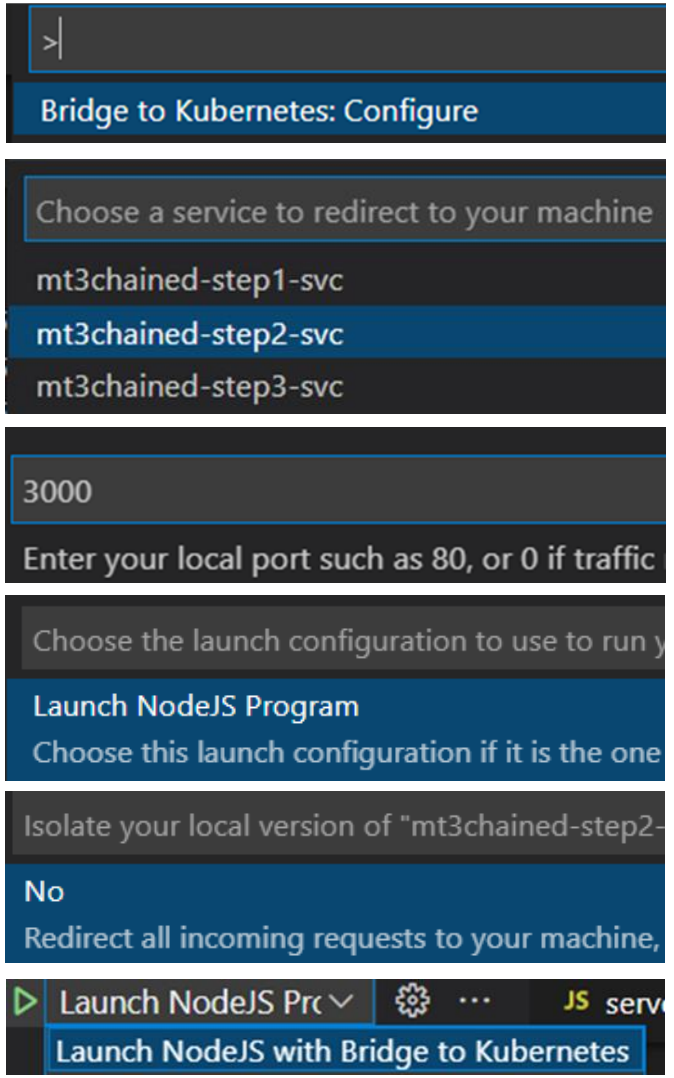



```
# Added by Bridge To Kubernetes
127.1.1.18 mt3chained-step4-svc mt3chained-s
127.1.1.17 mt3chained-step1-svc mt3chained-s
127.1.1.16 mt3chained-web-svc mt3chained-web
127.1.1.15 mt3chained-step3-svc mt3chained-s
127.1.1.14 mt3chained-step2-svc mt3chained-s
127.1.1.13 mt3chained-step5-svc mt3chained-s
# End of section
```

hosts

Configure Bridge to Kubernetes in VS Code

- Before you start debugging in VS Code, your cluster must be set to the namespace you'll use during the session:
  `kubectl config set-context --current --namespace chained`
- In your Command Palette, select **Bridge to Kubernetes: Configure**
- Select the service you wish to redirect to your machine.
- Enter the local port to redirect traffic to. Ensure your app is listening on that port.
- Select an existing configuration to use as a template when running your application with Bridge to Kubernetes.
- Select **No** for the Isolation mode.
- From the *debug* menu, run the new configuration

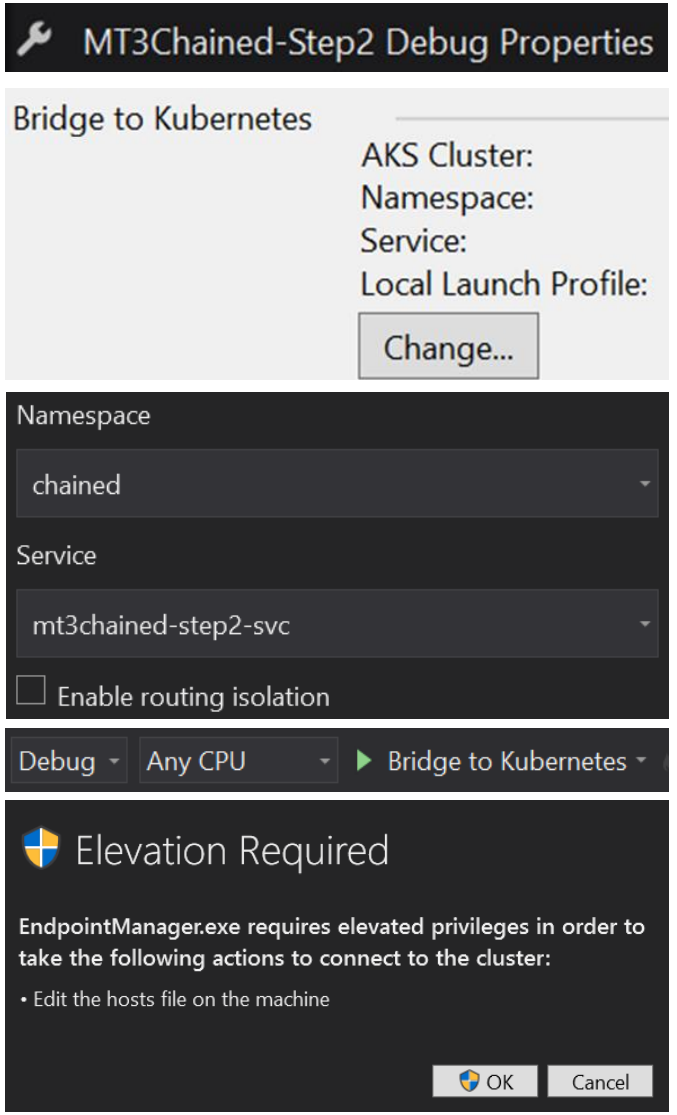


Bridge to Kubernetes with VS Code Demo

Connect local project running in VS Code to a running AKS cluster

Configure Bridge to Kubernetes in Visual Studio

- Unlike VS Code, the cluster does not need to be in the namespace you're connecting to.
- Select **Debug Properties** from the Profiles menu.
- Select the **Bridge to Kubernetes** profile and click the **Change** button.
- Configure the *Namespace* and *Service*, leaving the *Enable routing isolation* checkbox **unchecked**.
- The port listed in the ***launchSettings.json*** file will be used for redirection, so there's no need to specify one.
- From the *debug* menu, run the new profile.
- Click **OK** when *EndpointManager.exe* asks for elevated privileges.



Bridge to Kubernetes Visual Studio Demo

Connect local project running in Visual Studio to a running AKS cluster

Header Propagation in Isolation Mode

- Bridge to Kubernetes can also run in isolation mode, where *only your traffic (from a custom URL)* is redirected to your machine, while the original container stays in place.
- For isolation mode to work, you'll need to modify your code to propagate a special header (**"kubernetes-route-as"**) from incoming requests to any subsequent service requests.

- For .NET code in C#, you can use code like this in every request:

```
var request = new HttpRequestMessage();
request.RequestUri = new Uri("http://mywebapi/api/values/1");
if (this.Request.Headers.ContainsKey("kubernetes-route-as"))
{   // Propagate the bridge routing header
    request.Headers.Add("kubernetes-route-as", this.Request.Headers["kubernetes-route-as"]as IEnumerable<string>);
}
var response = await client.SendAsync(request);
```

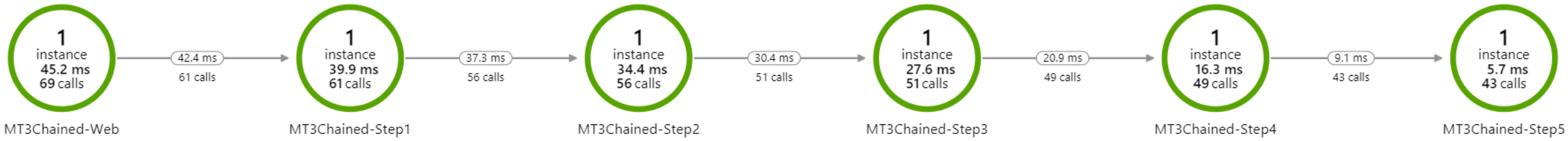
- A better option is to use the Microsoft.AspNetCore.HeaderPropagation NuGet package to perform this propagation automatically for every request. See example here.

Application Monitoring

Infrastructure Monitoring is Not Enough

- Azure Kubernetes Service provides *infrastructure monitoring* with Container Insights using a Log Analytics Workspace.

- Complex applications should also implement an *application monitoring* system, like **Application Insights**, to track application specific metrics and communications between various microservices.



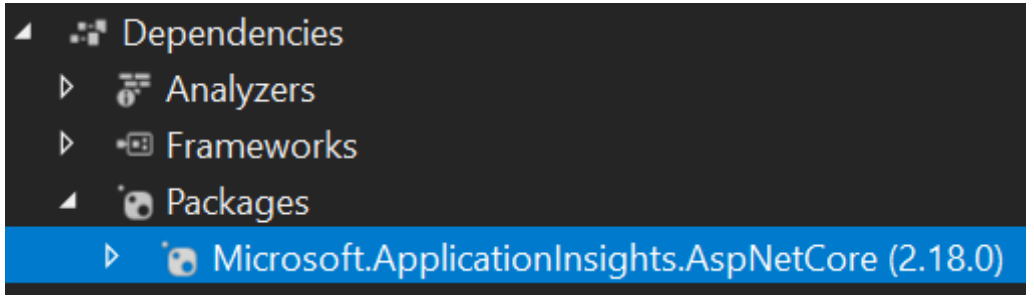
Application Insights SDKs

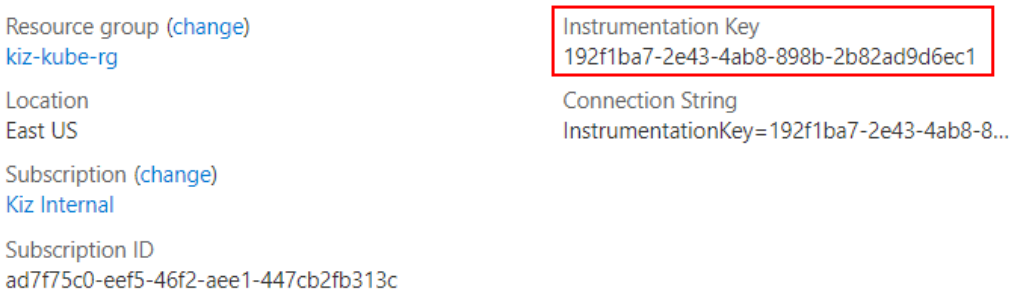
- To use Application Insights, you must integrate one of the support SDKs into your application and configuring it to send its telemetry to an Application Insights instance.

- The following platforms are “officially” supported (others available on GitHub):

Integrating Application Insights

- Integrating Application Insights into each microservice requires you to perform 3 simple steps:
  - Install the platform specific SDK
  - Specify the ***Instrumentation Key*** of an Application Insights instance
  - Initialize the SDK in the startup of your code
- In a Kubernetes cluster, the Instrumentation Key can be specified as an **environment variable** in a common Config Map.
- This ensures all microservices send telemetry to a common AI instance.



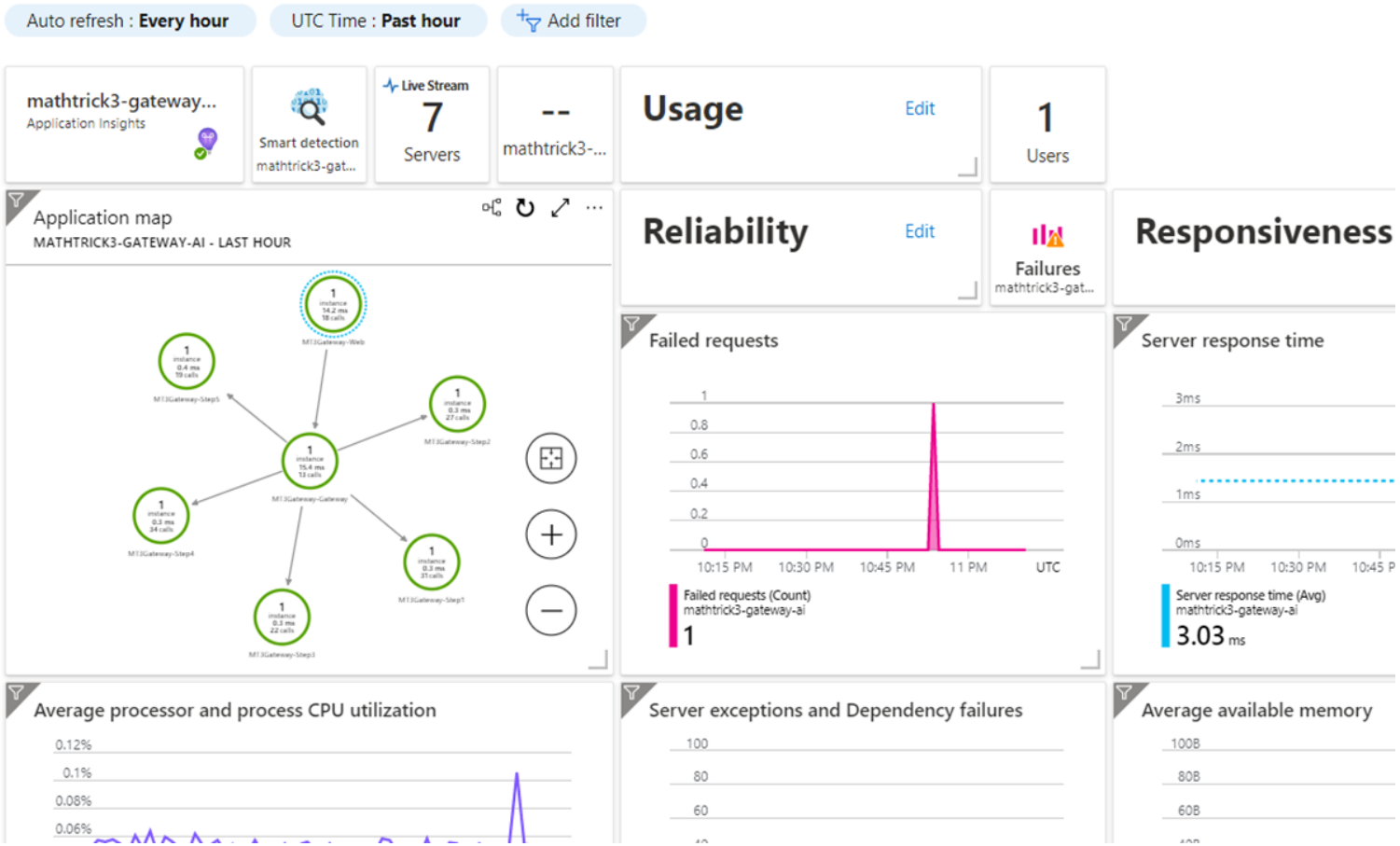


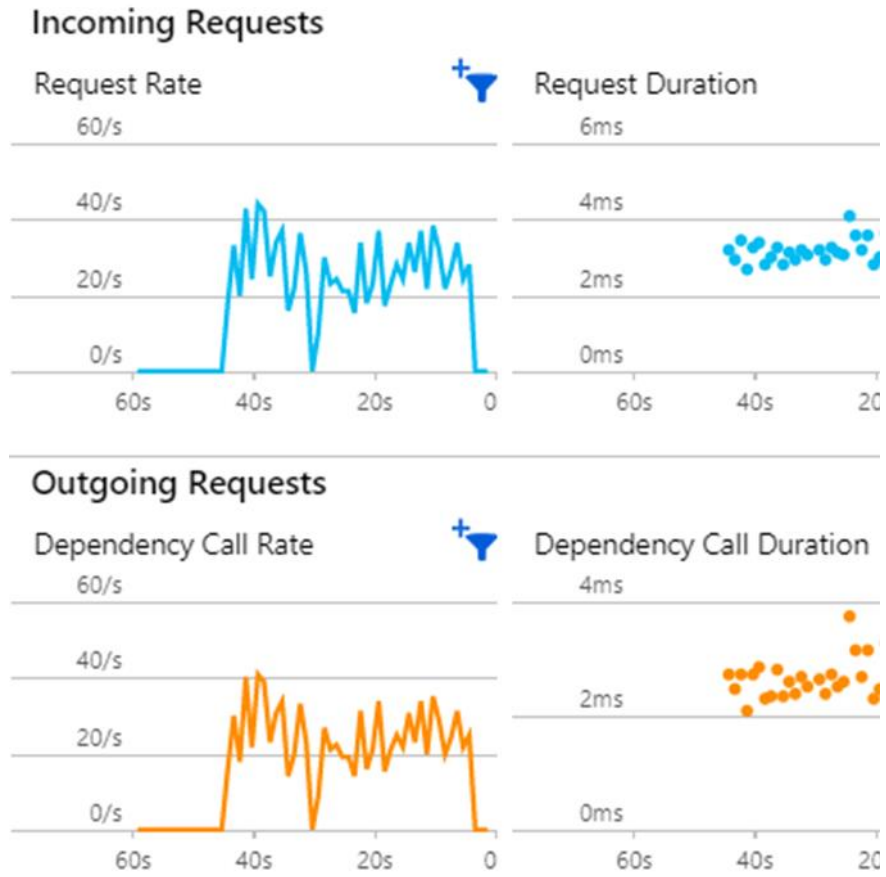

```
GlobalSettings.CloudRoleName = "MT3Chained-Web";
services.AddApplicationInsightsTelemetry();
services.AddSingleton<ITelemetryInitializer,
                      CloudRoleNameInitializer>();
```

```
data:
  FAILURE_RATE: "10"
  APPINSIGHTS_INSTRUMENTATIONKEY: "192f1ba7-2e43-4ab8-8
```

Application Insights

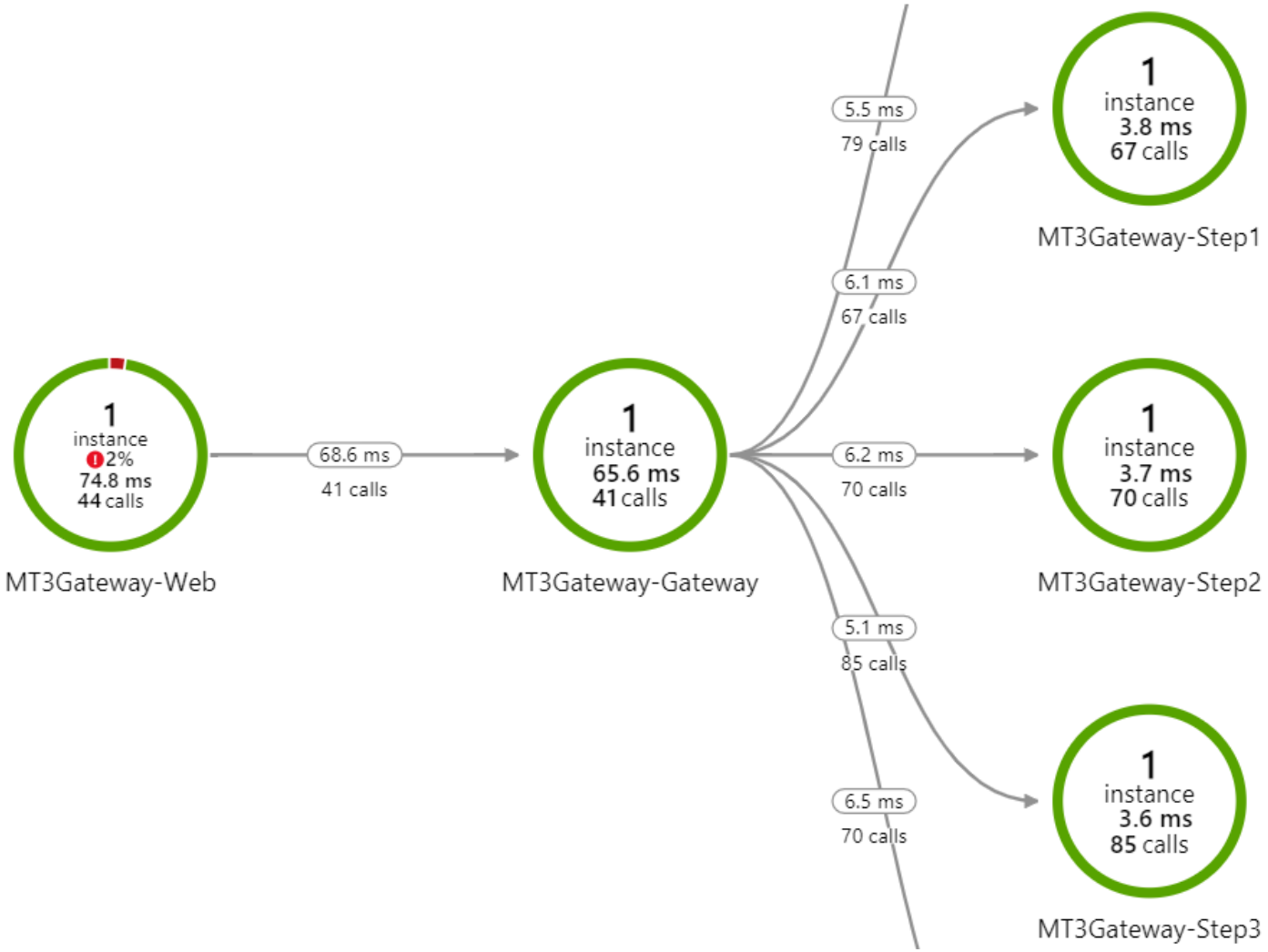
- Application Insights provides you application performance indicators and metrics





Application Insights

- Application Insights can automatically determine your application's internal and external dependencies.
- Using the **Application Map** feature, you can view how your microservices are communicating with each other and determine where the bottlenecks are.



Monitoring with Application Insights Demo

Collect metrics and logs from all your microservices using Application Insights SDKs

Lab – Module 4

Application Development with Kubernetes

Microsoft

Thank you