



# Modernization Without the Pain:

**The .NET Guide to  
Azure Container Apps**

Dev/  
Div/



**Hi! I'm Jiachen.**

**I work as a Product Manager, with a focus on .NET in Azure Container Apps.**

**What does**  
**Azure Container Apps**  
**offer .NET developers?**

**Azure Container Apps**  
**offers a different way to think**  
**about Kubernetes.**

# **Distributed Applications 101**

# What are the benefits of distributed applications?

## **High level of independent scalability**

Solve bottlenecks by scaling up specific microservices instead of the entire application.

## **Zero down-time deployments**

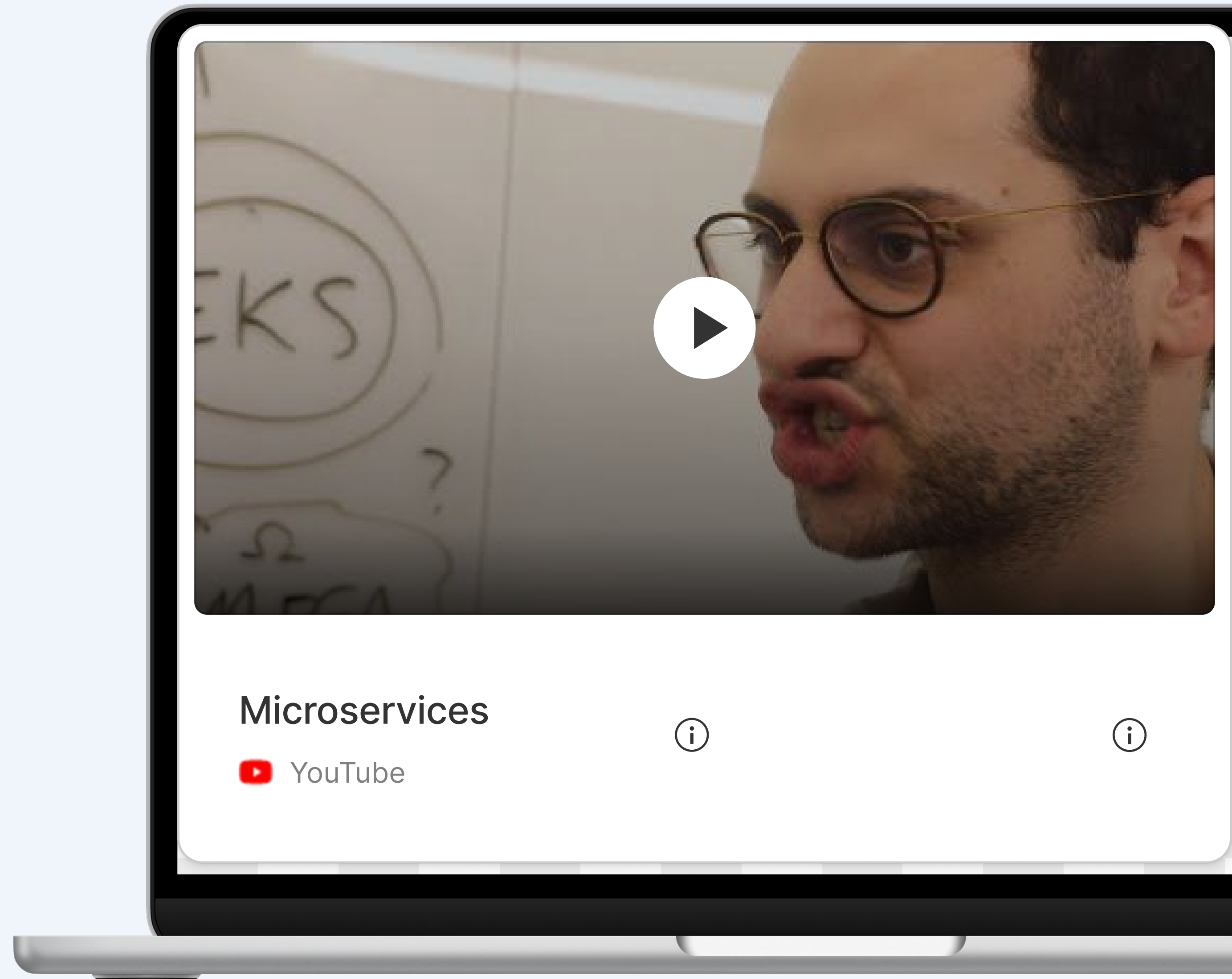
Add new features, patch vulnerabilities, and make changes to individual microservices without affecting the overall application.

## **Fault isolation and resiliency**

The application can continue running even when one microservice goes down.

# Distributed applications are not always the right option.

In three minutes, I will show you why.







## What are the cons of distributed applications?

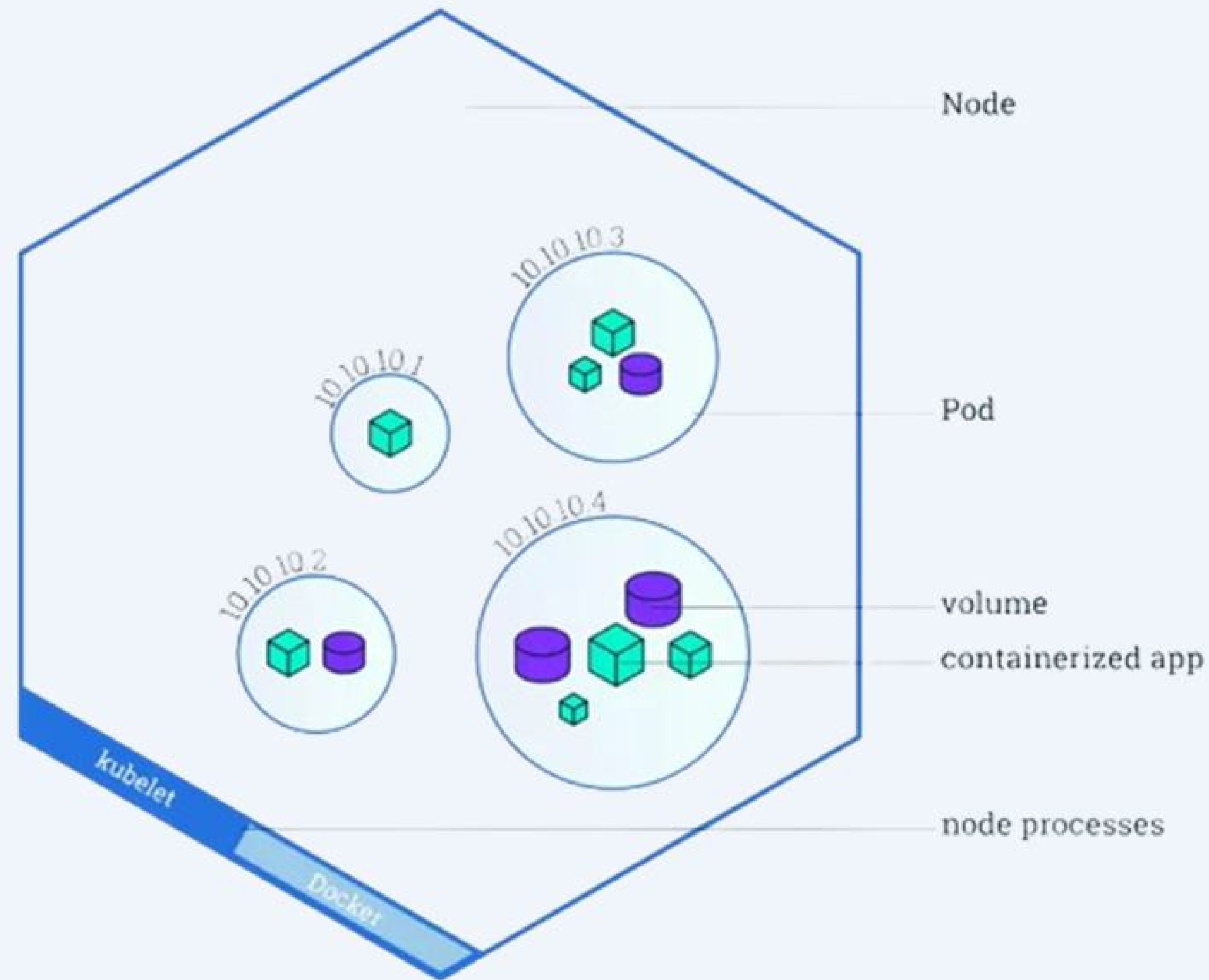


### **High upfront investment and much to learn.**

You might spend a lot of money, time, and energy learning about something that might not actually be a good fit.

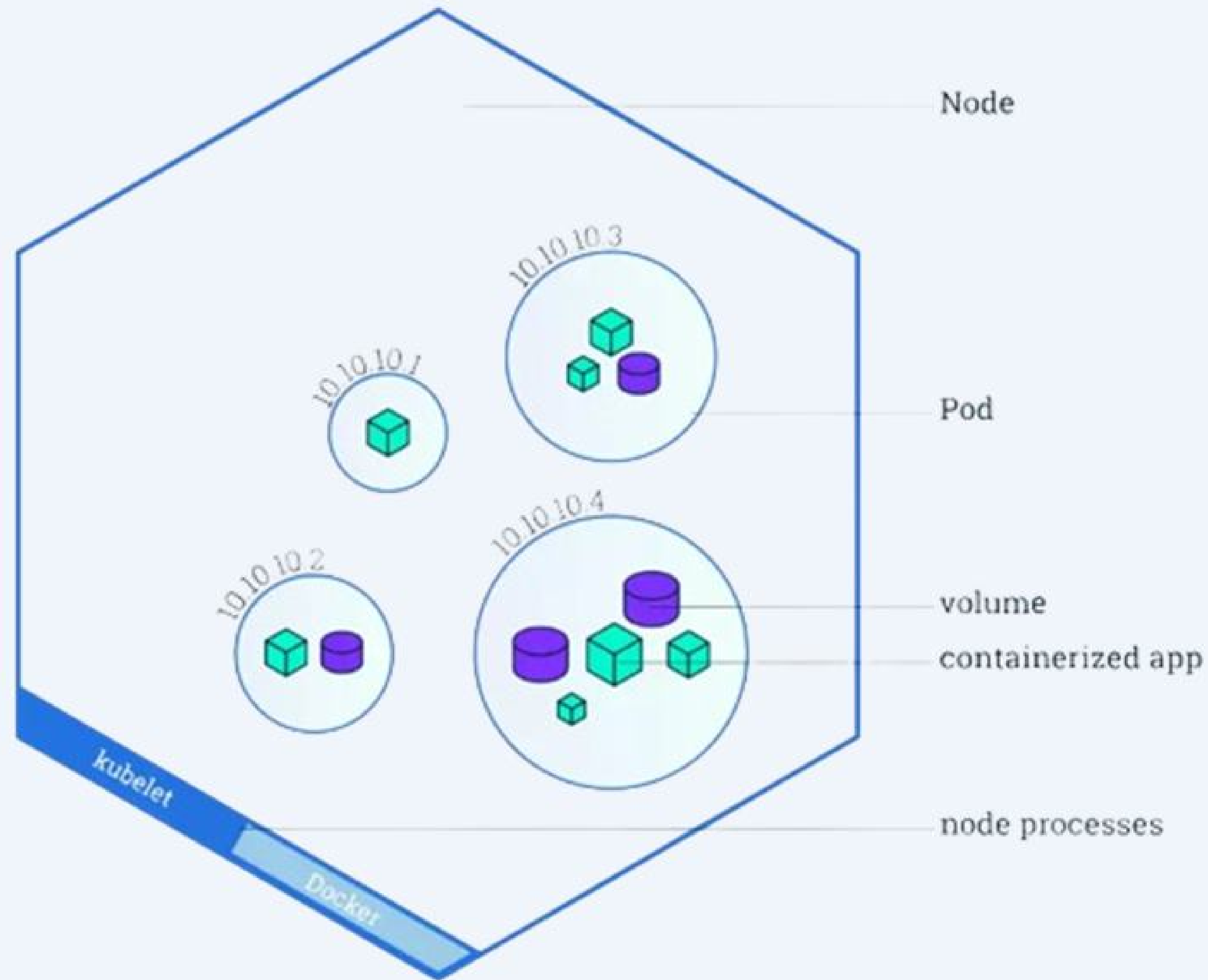
**How does Azure Container Apps  
make Kubernetes more accessible?**

# Kubernetes



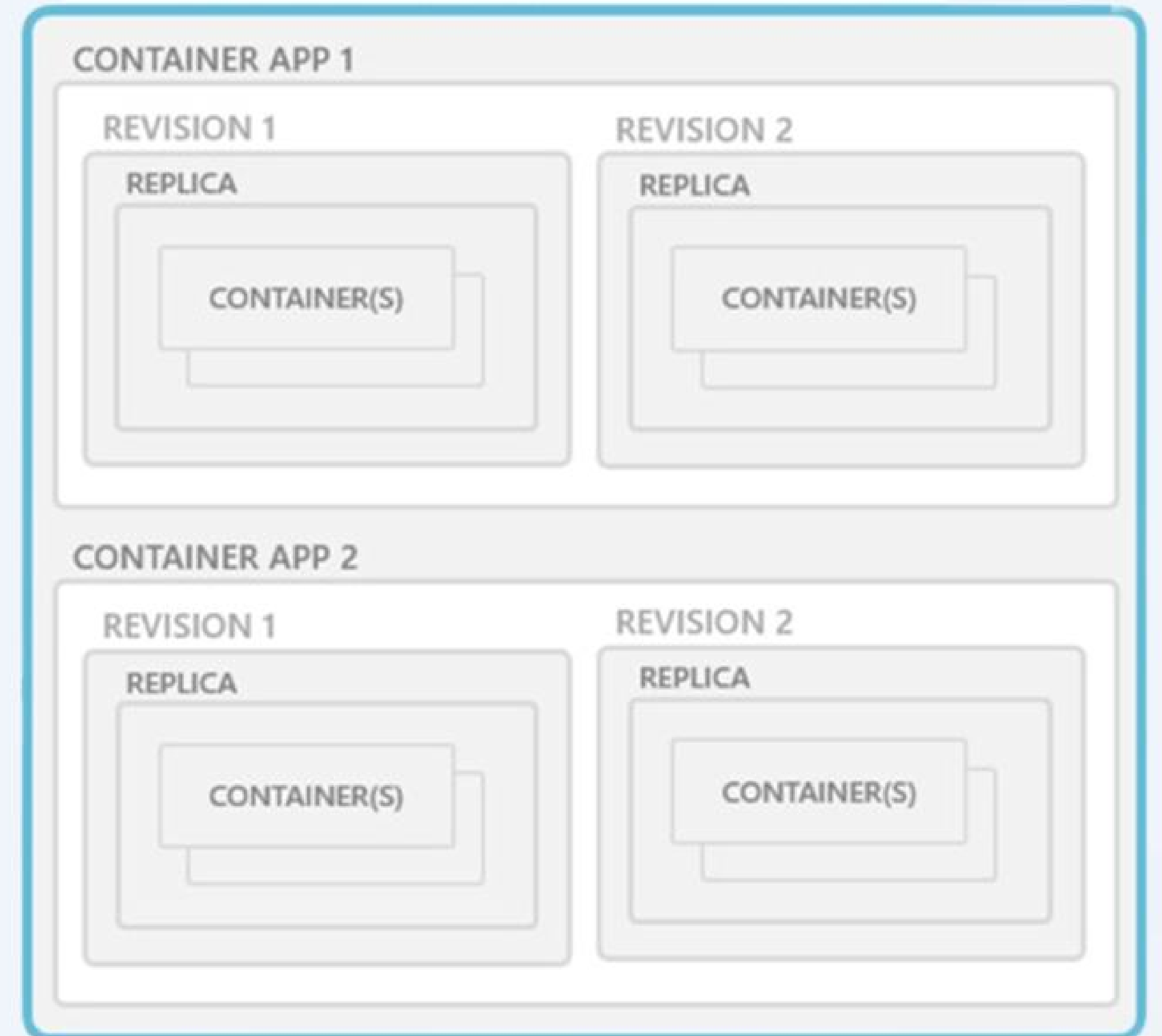
- Deploy a **cluster** - like the central nervous system - of **nodes**.
- Nodes host **pods**, which contain **containers** and what they need to run.
  - Specifically, containers require a **Kubelet** and **container runtime**, with optional Kube-proxy.

# Kubernetes



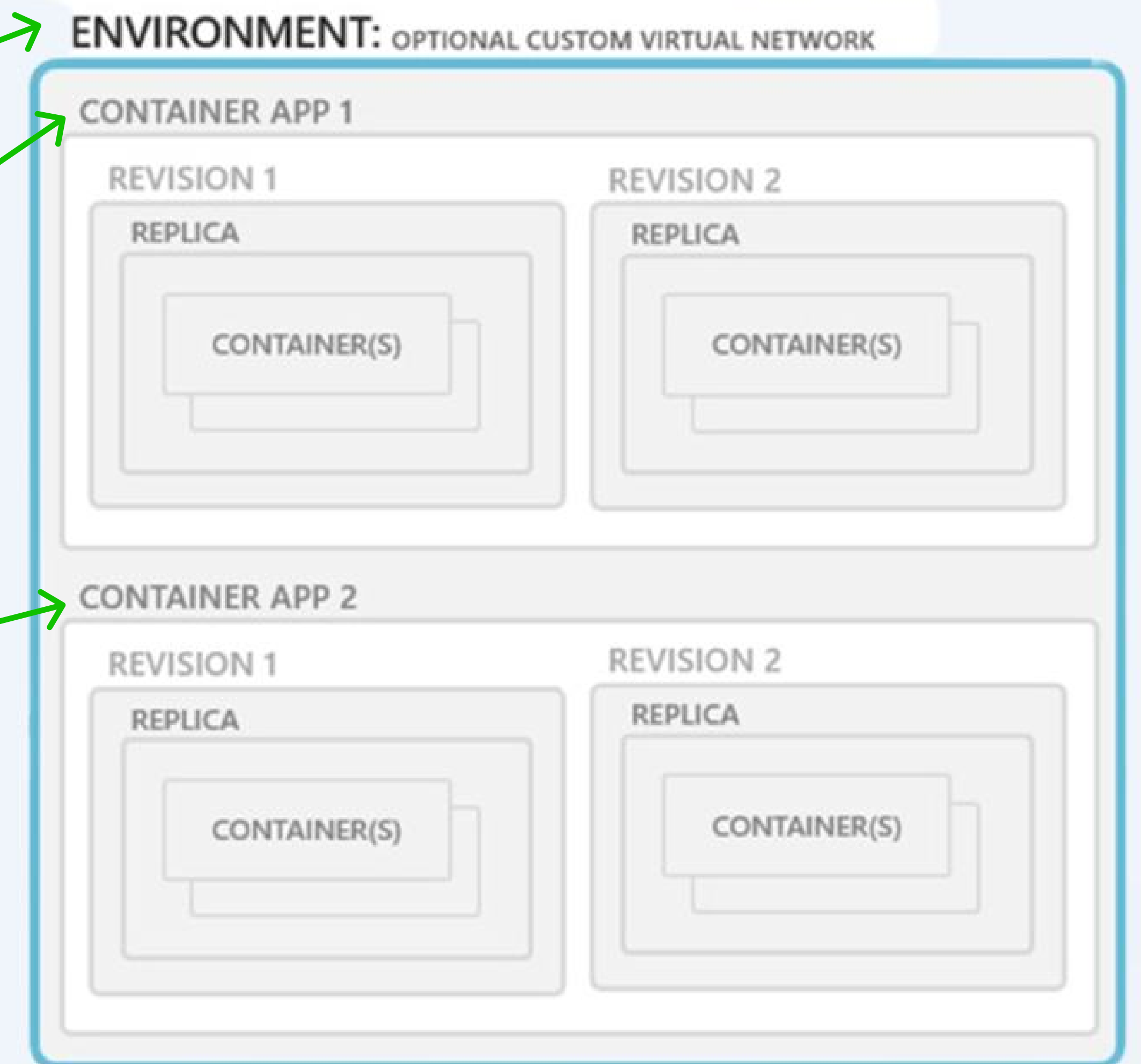
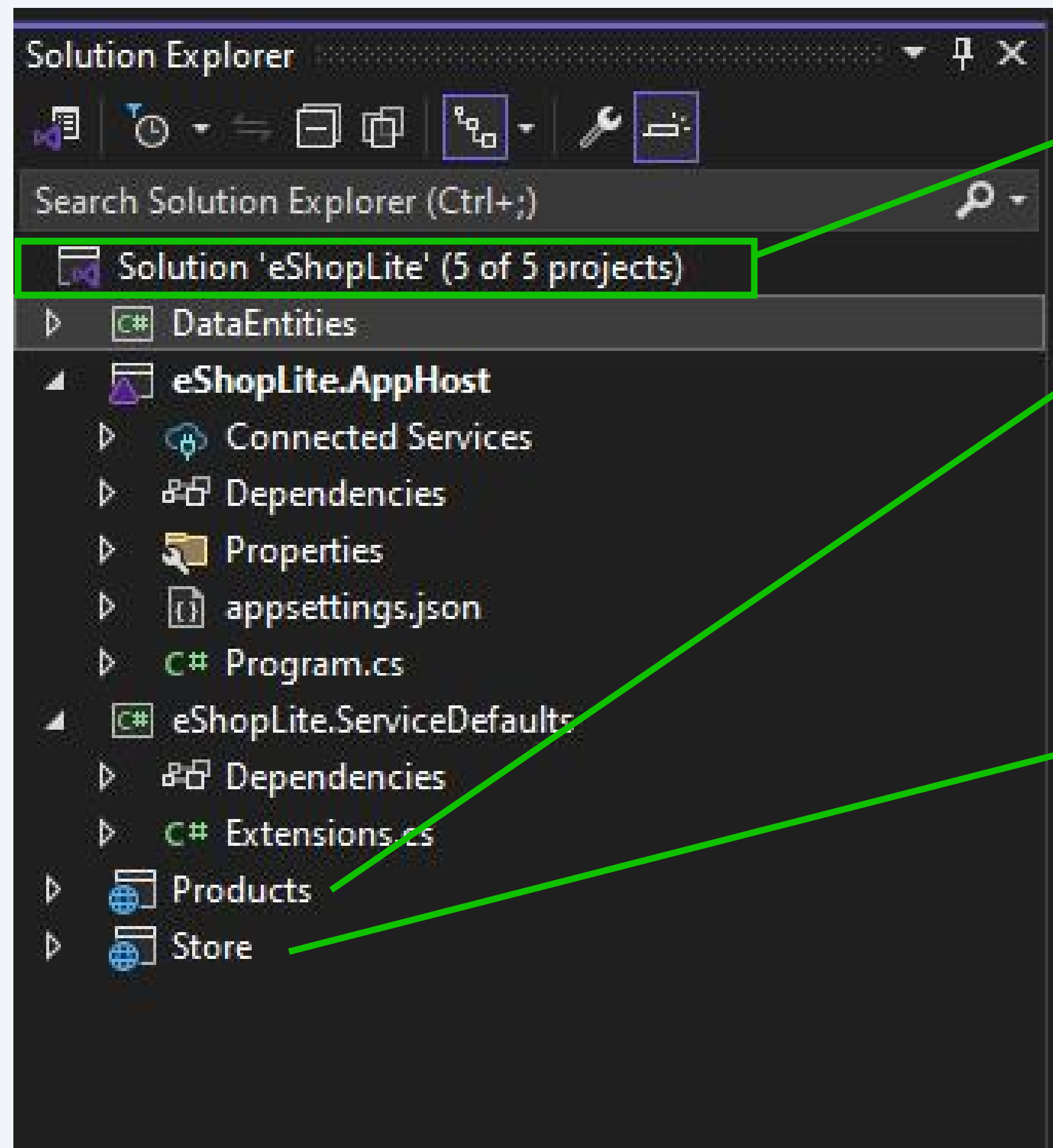
# Container Apps

ENVIRONMENT: OPTIONAL CUSTOM VIRTUAL NETWORK



# .NET Aspire

# Container Apps



**One Solution → One Environment, One App → One Container App**

# What are the cons of distributed applications?



## **High upfront investment and much to learn.**

You might spend a lot of money, time, and energy learning about something that might not actually be a good fit.

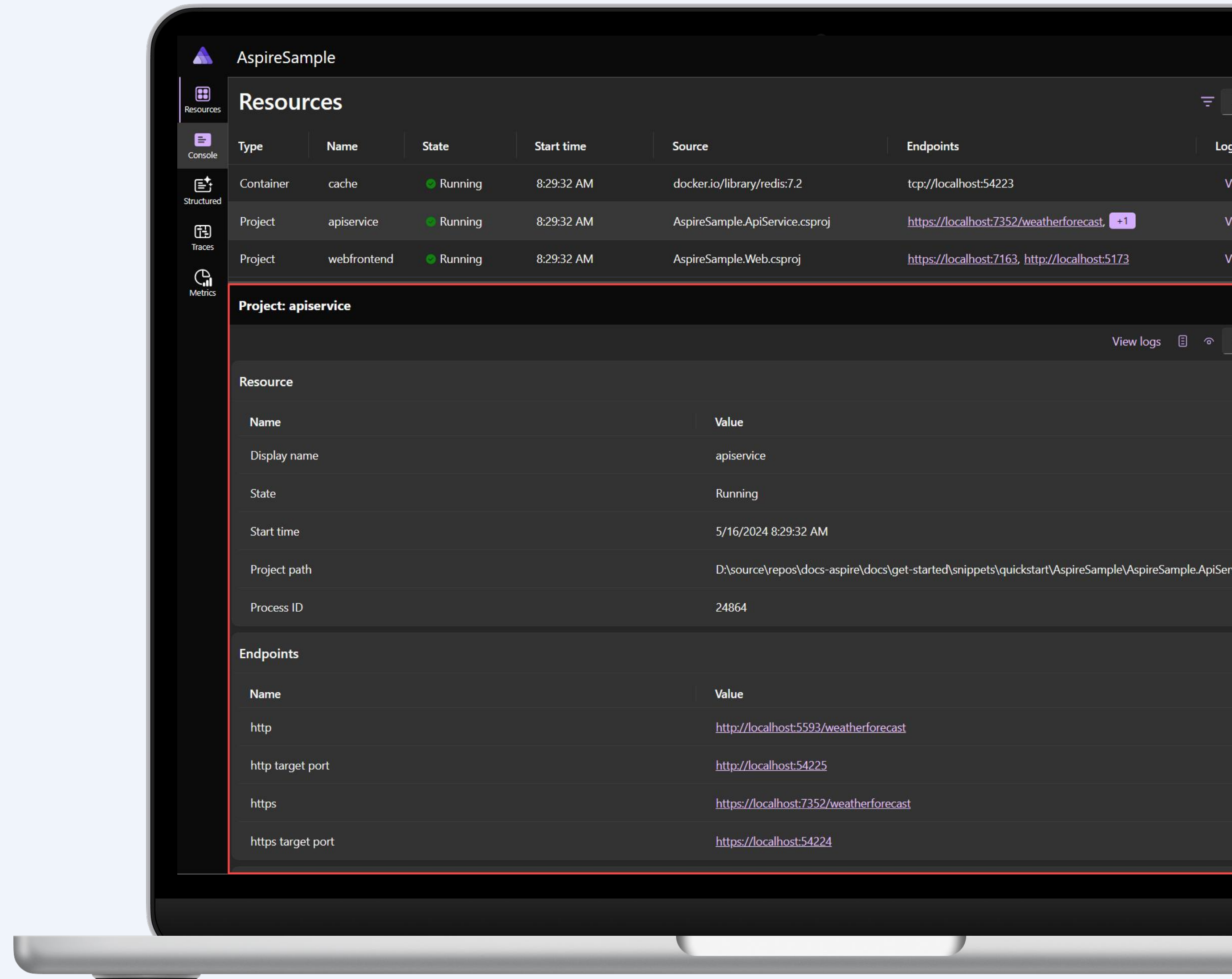
## **More complexity; requires optimization to avoid bloat.**

Multiple services means more communication, coordination, and integration testing.

**How do you optimize your  
distributed applications with  
.NET Aspire + Azure Container Apps?**

# Demo - Optimize your microservices

- Use the Aspire dashboard to identify inefficiencies.
- Add a Redis cache store to simplify communication across services.



The screenshot displays the Aspire dashboard for a project named 'AspireSample'. The 'Resources' section shows a table with columns for Type, Name, State, Start time, Source, and Endpoints. Three resources are listed: 'cache' (Container), 'apiservice' (Project), and 'webfrontend' (Project). The 'apiservice' resource is highlighted, and its details are shown in a separate pane below. This pane includes a 'Resource' section with fields for Name, Display name, State, Start time, Project path, and Process ID. It also features an 'Endpoints' section with fields for Name, http, http target port, https, and https target port.

Type	Name	State	Start time	Source	Endpoints
Container	cache	Running	8:29:32 AM	docker.io/library/redis:7.2	tcp://localhost:54223
Project	apiservice	Running	8:29:32 AM	AspireSample.ApiService.csproj	<a href="https://localhost:7352/weatherforecast">https://localhost:7352/weatherforecast</a> , <a href="#">+1</a>
Project	webfrontend	Running	8:29:32 AM	AspireSample.Web.csproj	<a href="https://localhost:7163">https://localhost:7163</a> , <a href="http://localhost:5173">http://localhost:5173</a>

Project: apiservice	
Resource	Value
Name	apiservice
Display name	apiservice
State	Running
Start time	5/16/2024 8:29:32 AM
Project path	D:\source\repos\docs-aspire\docs\get-started\snippets\quickstart\AspireSample\AspireSample.ApiService.csproj
Process ID	24864

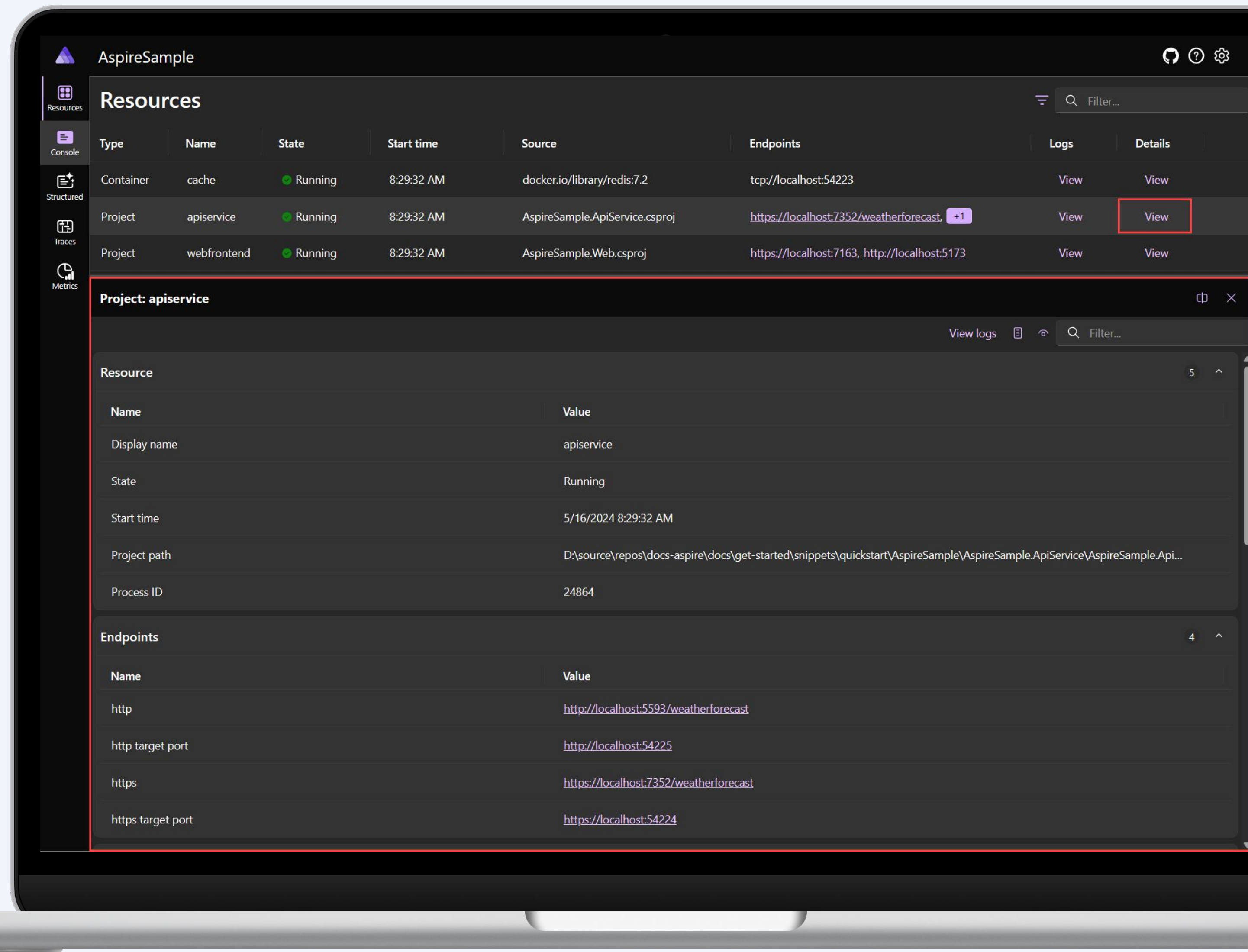
  

Endpoints	
Name	Value
http	<a href="http://localhost:5593/weatherforecast">http://localhost:5593/weatherforecast</a>
http target port	<a href="http://localhost:54225">http://localhost:54225</a>
https	<a href="https://localhost:7352/weatherforecast">https://localhost:7352/weatherforecast</a>
https target port	<a href="https://localhost:54224">https://localhost:54224</a>



# Before

1. Receive the request in the web app microservice.
2. Call the authentication microservice to check the users' identity if they're logged in.
3. Call the shopping basket microservice to find out what items and what quantities are in the basket.
4. Call the product catalog microservice to obtain full details of each product.
5. Call the images microservice to obtain image blobs for each product.
6. Call the stock taking microservice to check stock levels.
7. Call the shipping microservice to calculate shipping costs for the user's location and preferences.



Search  Refresh Delete

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Resource visualizer
- Settings
- Ingress
- Apps
- Services
- Monitoring
- Automation
- Help

Essentials

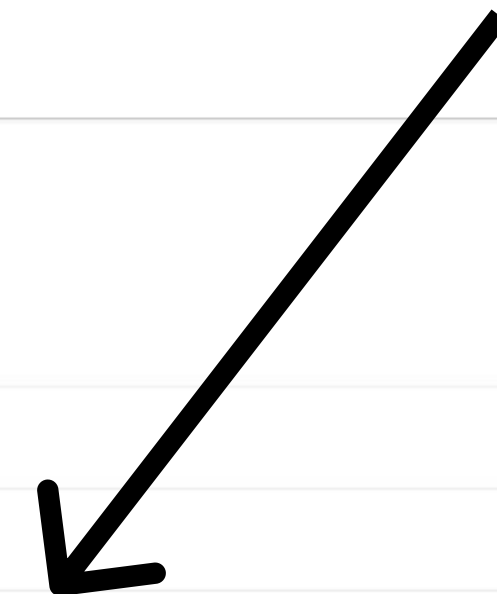
Resource group (move) : [rg-devup-recording](#)  
Location (move) : [Central US](#)  
Subscription (move) : [Private Test Sub JIACJIAN](#)  
Subscription ID : d0f67cc0-c14c-4fb2-92cf-e8093812d058

Environment type : Consumption only  
Static IP : 4.249.234.83  
Applications : 2  
KEDA version : 2.14.0  
Dapr version : 1.12.5  
.NET Aspire Dashboard : [Open dashboard](#) (settings)

Tags (edit) : azd-env-name : devup-recording

Applications [Get started](#) [Tutorials](#)

Name ↑	App Type	Resource Group
<a href="#">ca-api-bk2lemnhthr4o</a>	Container App	rg-devup-recording
<a href="#">ca-web-bk2lemnhthr4o</a>	Container App	rg-devup-recording



# Resources

Filter...

Type	Name	State	Start time	Source	Endpoints	Logs	Details	Commands
Container	<a href="#">ca-api-bk2lemnhthr4o</a>	<span style="color: green;">✔</span> Running	8/15/2024 10:22:27 AM		<a href="#">https://ca-api-bk2lemnhthr4o--azd-17237...</a>	<a href="#">View</a>	<a href="#">View</a>	...
Container	<a href="#">ca-web-bk2lemnhthr4o</a>	<span style="color: green;">✔</span> Running	8/15/2024 10:22:27 AM		<a href="#">https://ca-web-bk2lemnhthr4o--azd-1723...</a>	<a href="#">View</a>	<a href="#">View</a>	...

```
1 var builder = DistributedApplication.CreateBuilder(args);
2
3 | var products = builder.AddProject<Projects.Products>("product
4
5 | builder.AddProject<Projects.Store>("store").WithReference(pr
6
7 | builder.Build().Run();
8
```

Search Solution Explorer (Ctrl+):

- ✓ Solution 'eShopLite' (5 of 5 projects)
  - GitHub Actions
  - DataEntities
  - ✦ eShopLite.AppHost
    - ▶ Connected Services
    - ▶ Dependencies
    - ▶ Properties
    - ▶ appsettings.json
    - ▶ Program.cs
  - ✦ eShopLite.ServiceDefaults
    - ▶ Dependencies
    - ▶ Extensions.cs
  - Products
  - Store
    - ▶ Connected Services
    - ▶ Dependencies
    - ▶ Properties
    - ▶ wwwroot
    - ▶ Components
    - ▶ Services
    - ▶ appsettings.json
    - ▶ Program.cs

% No issues found | Ln: 7 Ch: 23 SPC CRLF

Error List

Build + IntelliSense

0 Errors 0 Warnings 0 Messages

Search Error List

Code	Description	Project	File	Line	Suppression State
------	-------------	---------	------	------	-------------------

Adding the Redis Aspire package

# After

1. Receive the request in the web app microservice.
2. Retrieve data from Redis cache store
3. ???
4. Profit! (or at least, the shopping cart is populated)

The screenshot displays the Visual Studio Aspire interface for a project named 'AspireSample'. The 'Resources' view is active, showing a table of resources:

Type	Name	State	Start time	Source	Endpoints	Logs	Details
Container	cache	Running	8:29:32 AM	docker.io/library/redis:7.2	tcp://localhost:54223	View	View
Project	apiservice	Running	8:29:32 AM	AspireSample.ApiService.csproj	<a href="https://localhost:7352/weatherforecast">https://localhost:7352/weatherforecast</a> +1	View	View
Project	webfrontend	Running	8:29:32 AM	AspireSample.Web.csproj	<a href="https://localhost:7163">https://localhost:7163</a> , <a href="http://localhost:5173">http://localhost:5173</a>	View	View

The 'Project: apiservice' details pane is expanded, showing the following information:

Resource	Value
Name	apiservice
Display name	apiservice
State	Running
Start time	5/16/2024 8:29:32 AM
Project path	D:\source\repos\docs-aspire\docs\get-started\snippets\quickstart\AspireSample\AspireSample.ApiService\AspireSample.Api...
Process ID	24864

The 'Endpoints' section of the details pane lists the following:

Name	Value
http	<a href="http://localhost:5593/weatherforecast">http://localhost:5593/weatherforecast</a>
http target port	<a href="http://localhost:54225">http://localhost:54225</a>
https	<a href="https://localhost:7352/weatherforecast">https://localhost:7352/weatherforecast</a>
https target port	<a href="https://localhost:54224">https://localhost:54224</a>

# What are the cons of distributed applications?



## **High upfront investment and much to learn.**

You might spend a lot of money, time, and energy learning about something that might not actually be a good fit.

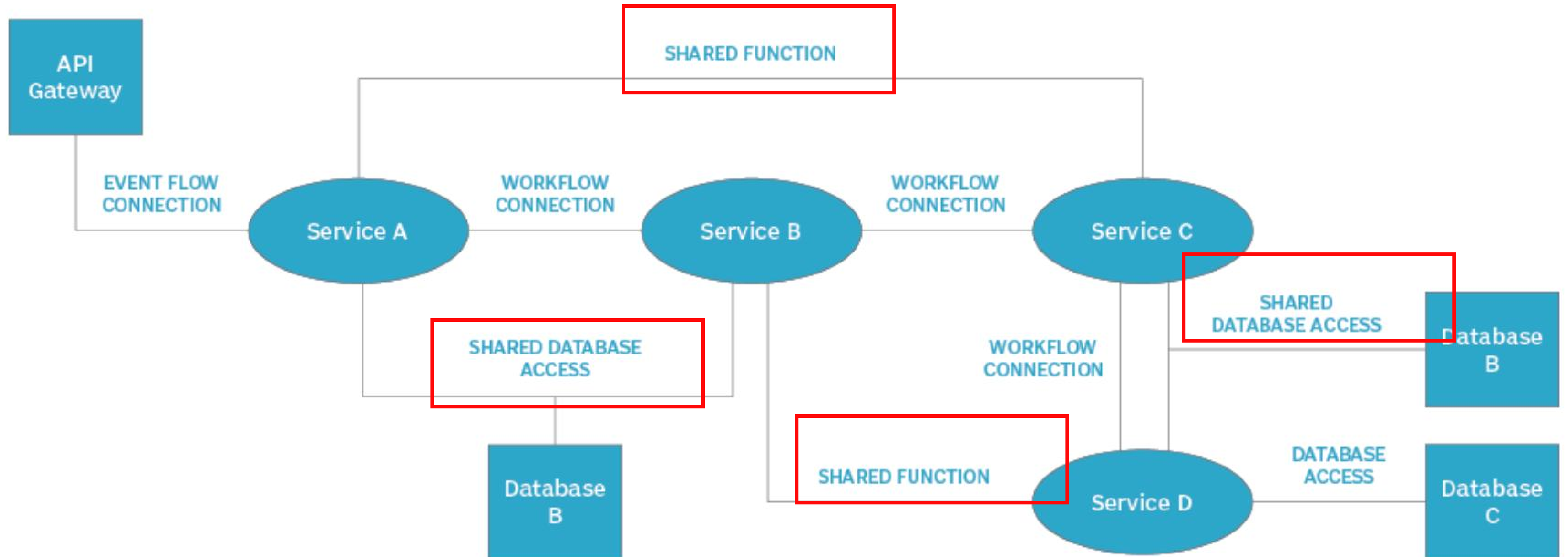
## **More complexity and much to learn.**

Multiple services means more communication, coordination, and integration testing.

## **Needs to be implemented carefully to avoid the “distributed monolith.”**

If your architecture is not designed well, you can get the negatives of both monolithic and distributed architectures.

# What is a distributed monolith?



## How to prevent a distributed monolith

**Start simple.**

**Refactor to  
microservices  
incrementally,  
only when  
needed.**

If the only tool you have  
is a hammer, **you tend to  
see every problem as a  
nail**



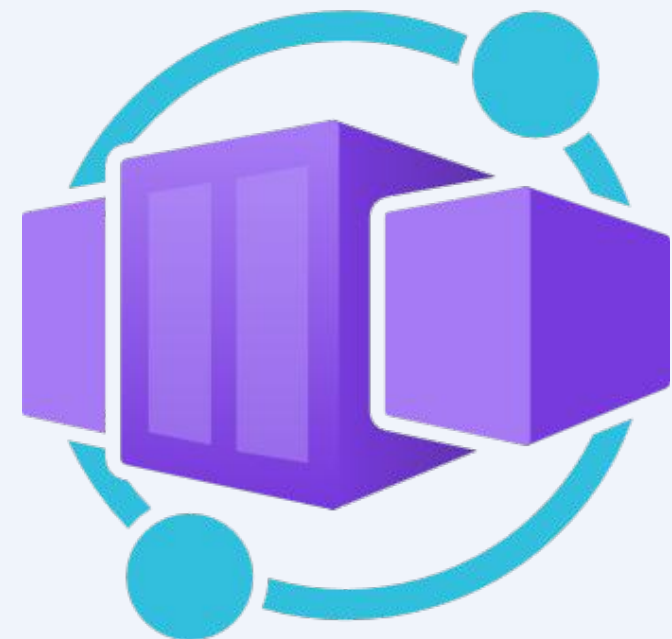
Abraham Maslow

# Resources in Azure Container Apps

## Apps

Runs continuously and restarts automatically upon exit of process.

Ex. HTTP APIs, web apps, and background services



## Jobs

Runs for a set time and exits; each one represents a separate unit of work.

Ex. Batch processes (on-demand and scheduled)



## Sessions

Secure sandbox environments for running code that require isolation.

- Ex. Running AI generated code or commands submitted by users





## What can lead to a distributed monolith?

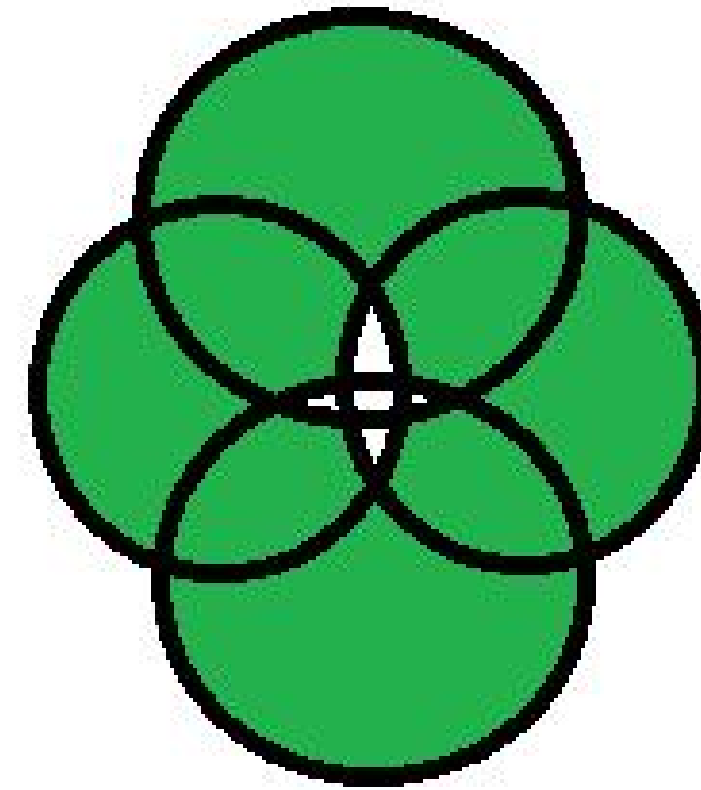
**Ensure you are decomposing your application effectively.**

### The Goldilocks Rule



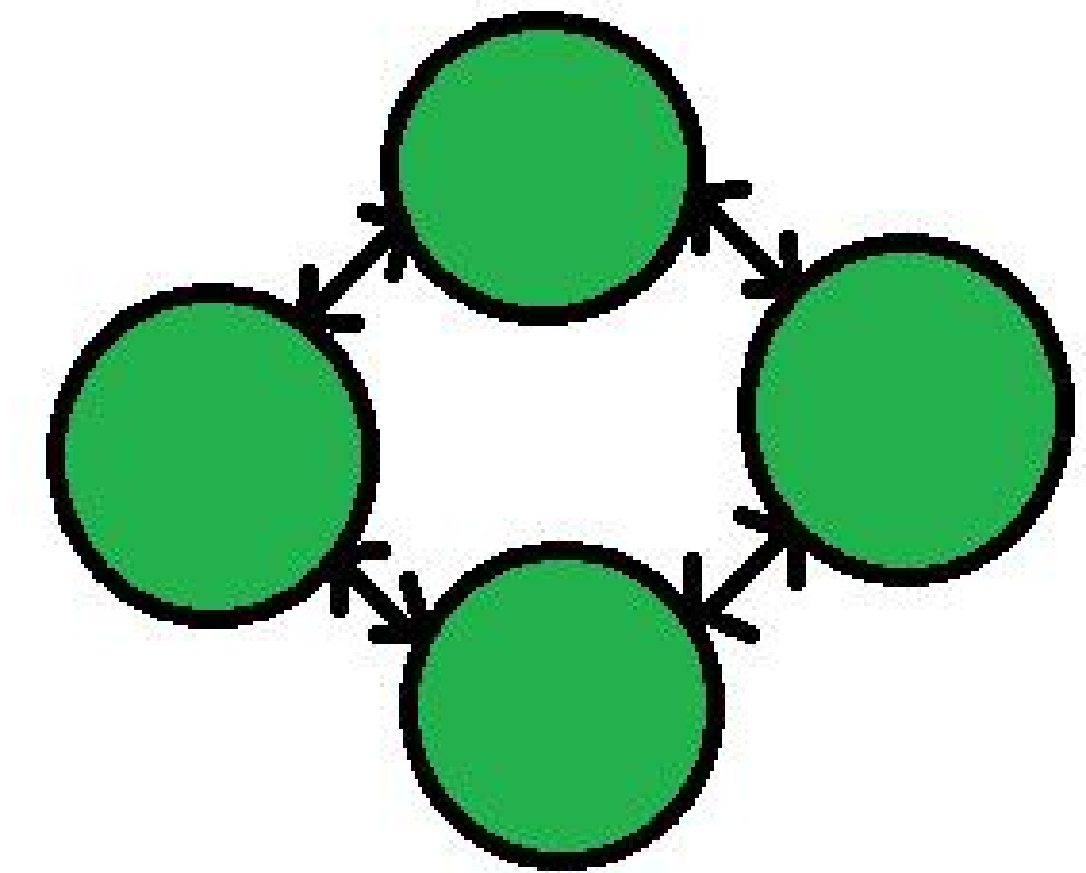
# What can lead to a distributed monolith?

**Loosely couple  
components.**



**Tight coupling:**

1. More Interdependency
2. More coordination
3. More information flow



**Loose coupling:**

1. Less Interdependency
2. Less coordination
3. Less information flow

# Approaches for building quality distributed apps

## Rolling

Default deployment strategy in Kubernetes. **Incrementally replace pod instances with a new version.** based on the server instance.

Results in **minimal downtime** as current versions stay live.

However, **can result in high latency and rolling back is often difficult.**

## Canary

**Release updates incrementally to a subset of users.** Lowest risk and computationally cheaper than blue-green deployment.

Results in **early feedback from users and detection of performance/latency issues** before full release.

However, requires smart traffic switching method instead of just a load balancer and **can be technically challenging.**

## Blue-Green

**Have two identical environments - one “blue” (staging) and the other “green” (production.)** Easy rollback and traffic is switched over instantly. Multiple versions of an app never get run in parallel - good for legacy apps.

Results in **quality assurance and minimizes risk/downtime** when updating your app.

However, requires twice the computational resources and **can be technically challenging.**

**How do you test the  
performance and latency of  
your app using  
Azure Container Apps?**

# Approaches for building quality distributed apps

## Rolling

Default deployment strategy in Kubernetes. **Incrementally replace pod instances with a new version.** based on the server instance.

Results in **minimal downtime** as current versions stay live.

However, **can result in high latency and rolling back is often difficult.**

## Canary

**Release updates incrementally to a subset of users.** Lowest risk and computationally cheaper than blue-green deployment.

Results in **early feedback from users and detection of performance/latency issues** before full release.

However, requires smart traffic switching method instead of just a load balancer and **can be technically challenging.**

## Blue-Green

**Have two identical environments - one “blue” (staging) and the other “green” (production.)** Easy rollback and traffic is switched over instantly. Multiple versions of an app never get run in parallel - good for legacy apps.

Results in **quality assurance and minimizes risk/downtime** when updating your app.

However, requires twice the computational resources and **can be technically challenging.**

```
export APP_NAME=<APP_NAME>
export APP_ENVIRONMENT_NAME=<APP_ENVIRONMENT_NAME>
export RESOURCE_GROUP=<RESOURCE_GROUP>

# A commitId that is assumed to correspond to the app code currently in production
export BLUE_COMMIT_ID=fb699ef
# A commitId that is assumed to correspond to the new version of the code to be deployed
export GREEN_COMMIT_ID=c6f1515

# create a new app with a new revision
az containerapp create --name $APP_NAME \
  --environment $APP_ENVIRONMENT_NAME \
  --resource-group $RESOURCE_GROUP \
  --image mcr.microsoft.com/k8se/samples/test-app:$BLUE_COMMIT_ID \
  --revision-suffix $BLUE_COMMIT_ID \
  --env-vars REVISION_COMMIT_ID=$BLUE_COMMIT_ID \
  --ingress external \
  --target-port 80 \
  --revisions-mode multiple

# Fix 100% of traffic to the revision
az containerapp ingress traffic set \
  --name $APP_NAME \
  --resource-group $RESOURCE_GROUP \
  --revision-weight $APP_NAME--$BLUE_COMMIT_ID=100

# give that revision a label 'blue'
az containerapp revision label add \
  --name $APP_NAME \
  --resource-group $RESOURCE_GROUP \
  --label blue \
  --revision $APP_NAME--$BLUE_COMMIT_ID
```

## Multi-revision mode

Customize unique identifier for the revision

Work with one version of the app (single revision mode) or multiple

How much traffic gets directed to each revision - in this case, 100% to blue

Creates URL that can point to different revisions

**How do you host non-Aspire  
.NET projects on  
Azure Container Apps?**

# Step 1: Containerize your code (Dockerfile)

Write a Dockerfile or generate one using Visual Studio or VS Code.

```
docker

FROM mcr.microsoft.com/dotnet/sdk:8.0@sha256:35792ea4ad1db051981f62b313f1be3b46b1f45cadbaa3c288cc
WORKDIR /App

# Copy everything
COPY . ./

# Restore as distinct layers
RUN dotnet restore

# Build and publish a release
RUN dotnet publish -c Release -o out

# Build runtime image
FROM mcr.microsoft.com/dotnet/aspnet:8.0@sha256:6c4df091e4e531bb93bdbfe7e7f0998e7ced344f54426b7e8
WORKDIR /App
COPY --from=build-env /App/out .
ENTRYPOINT ["dotnet", "DotNet.Docker.dll"]
```

Additional information

ASP.NET Core Web App (Razor Pages) C# Linux macOS Windows Cloud Service Web

Framework ⓘ  
 .NET 8.0 (Long Term Support)

Authentication type ⓘ  
 None

Configure for HTTPS ⓘ

Enable Docker ⓘ

Docker OS ⓘ  
 Linux

Do not use top-level statements ⓘ

Back Create



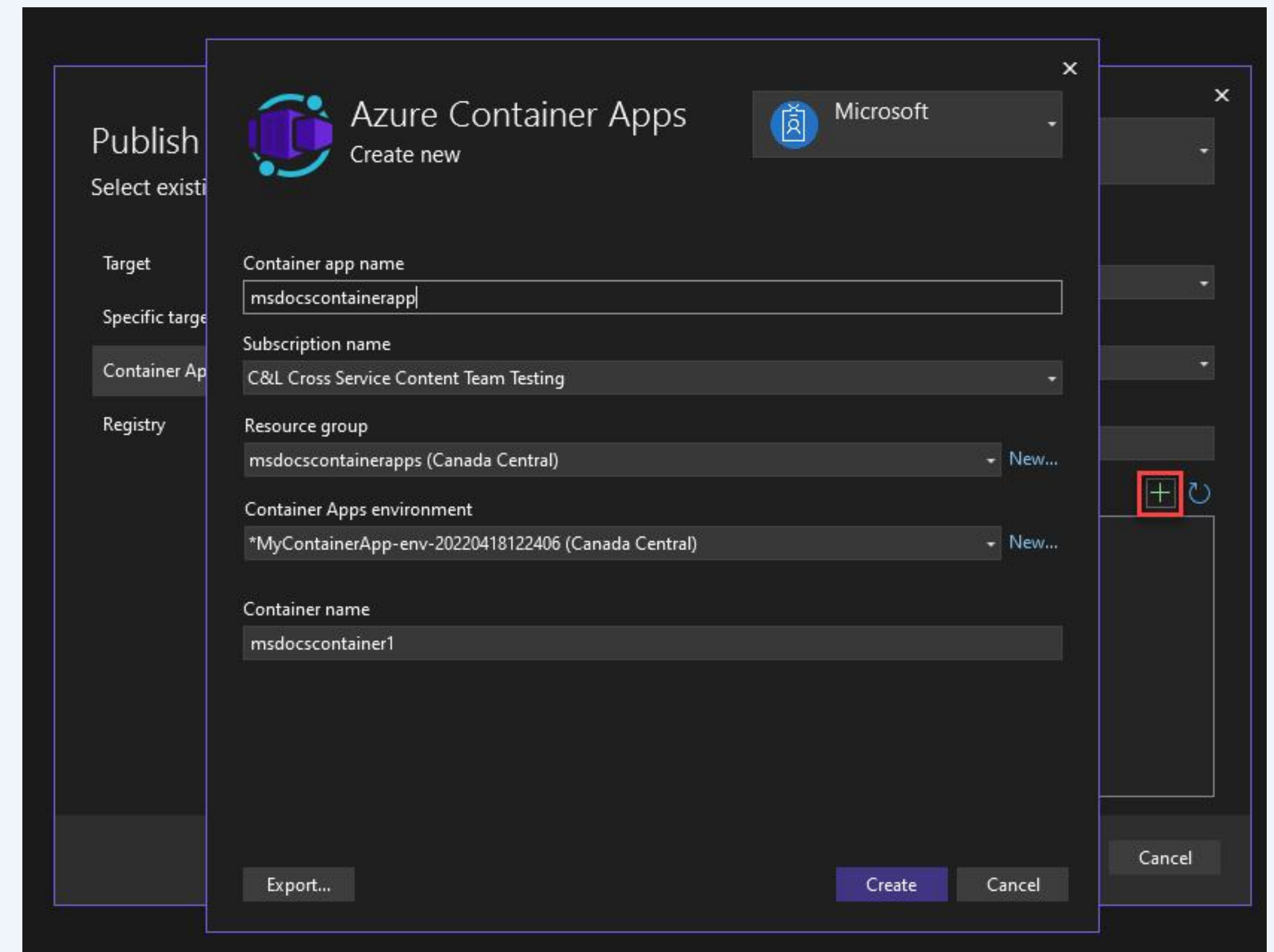
## Step 1: Containerize your code (no Dockerfile)

You can also containerize without a Dockerfile by using **dotnet publish command** or deploying to Container Apps directly from Visual Studio or VS Code.

To publish the .NET app as a container, use the following `dotnet publish` command:

```
.NET CLI
```

```
dotnet publish --os linux --arch x64 /t:PublishContainer
```



<https://aka.ms/dotnet-containerize-without-dockerfile>

## Step 2: Deploy your app

You can either deploy to Azure Container Apps from a container image or the source code.

**We recommend starting with a Dockerfile and/or a containerized application for more customization and easier debugging.**

## Deployment with existing image

```
az containerapp up \  
  --name my-container-app \  
  --resource-group my-container-apps \  
  --location centralus \  
  --environment 'my-container-apps' \  
  --image mcr.microsoft.com/k8se/quickstart:latest \  
  --target-port 80 \  
  --ingress external \  
  --query properties.configuration.ingress.fqdn
```

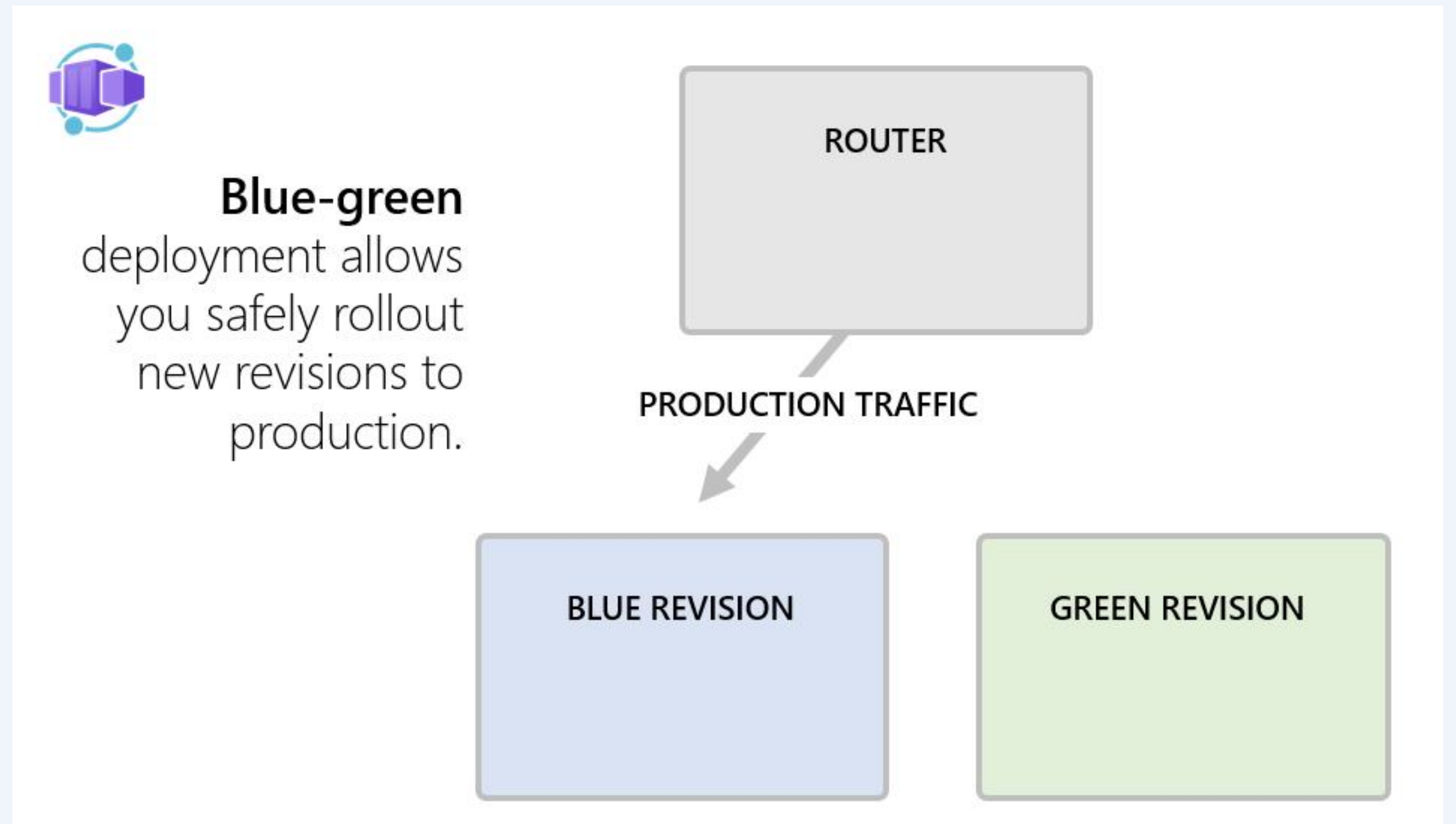
## Deployment with no Dockerfile (Buildpacks)

```
az containerapp up \  
  --name $API_NAME \  
  --location $LOCATION \  
  --environment $ENVIRONMENT \  
  --ingress external \  
  --target-port 8080 \  
  --source .
```

## Step 3: Test your app with multiple revisions

Using multi-revision mode allows you to implement deployment strategies and **evaluate the performance/latency of your services before full roll-out.**

Note: .NET Aspire currently does not support multi-revision mode - it should be implemented from the Container Apps side.



# What is not a great fit for Azure Container Apps?

## Traditional apps with simple infrastructure needs

Container Apps simplifies the infrastructure of complex, cloud-native workloads in ways that some apps do not need.

If you do not need much control over the underlying infrastructure, **Azure App Service** is the better fit.

## Simple, single-container apps that do not need to scale or load balance

Container Apps supports a lot around containers, but may not be necessary if you need very little.

If you are sticking to a single container and don't need certificates or scaling, **Azure Container Instances** is a good option.

## Complex, custom microservices that require access to the control plane API

Container Apps does not allow you to query and manipulate API objects (i.e. Pods, Namespaces, Events.) This can lead to some challenges in debugging.

If you need granular control, **Azure Kubernetes Service** is the better fit.

# Takeaways

## **Usability > New and shiny**

The goal is not to reinvent the wheel, but to build on existing functionality in order to make it more usable.

## **Test and validate the architecture of your distributed application.**

Incorrect division of services and tight coupling can lead to apps that have the worst of both monoliths and microservices.

## **.NET on Container Apps is more than just Aspire!**

The platform offers a lot for .NET developers, whether they are using Aspire or vanilla .NET.

# Thank you!

You can reach out to me at:

- @jiachenjiang\_ (Twitter/X)
- jiachen.jiang@microsoft.com (E-mail)

Please rate this session using



**.NET DeveloperDays Mobile App**  
(available in AppStore & Google Play)