Build and store an image by using Azure Container Registry

An Azure container registry is a private Docker registry deployed in Azure that you can keep network-close to your deployments. In this set of three tutorial articles, you learn how to use geo-replication to deploy an ASP.NET Core web application running in a Linux container to two Web Apps for Containers instances. You'll see how Azure automatically deploys the image to each Web App instance from the closest geo-replicated repository.

In this tutorial:

- Create a geo-replicated Azure container registry
- Clone application source code from GitHub
- Build a Docker container image from application source
- Push the container image to your registry

Before you begin

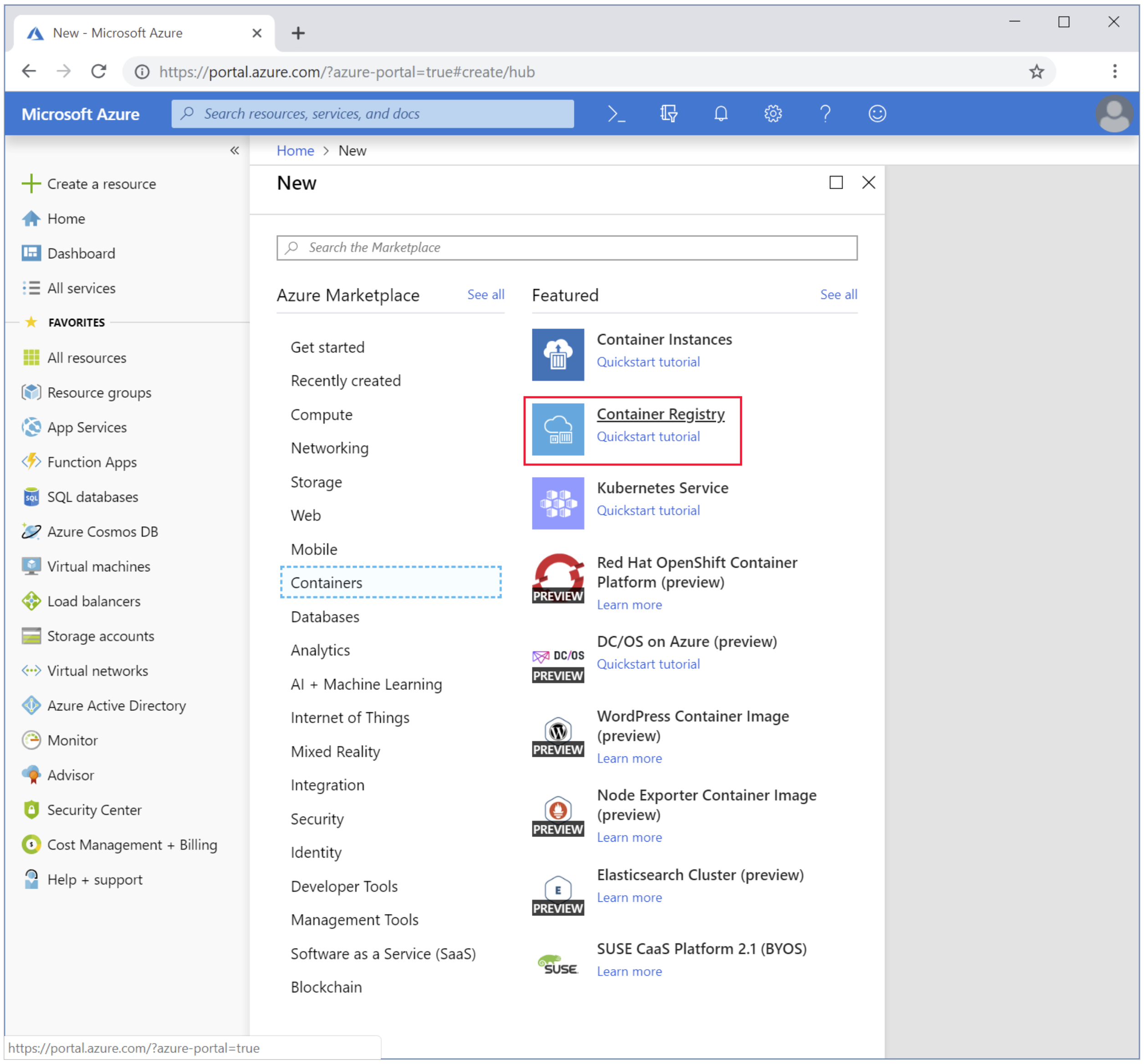
This tutorial requires a local installation of the Azure CLI (version 2.0.31 or later). Run `az --version` to find the version. If you need to install or upgrade, see Install Azure CLI.

To complete this tutorial, you need a local Docker installation. Docker provides installation instructions for macOS, Windows, and Linux systems.

Azure Cloud Shell does not include the Docker components required to complete every step this tutorial. Therefore, we recommend a local installation of the Azure CLI and Docker development environment.

Create a registry in Azure Container Registry

1. Sign in to the Azure portal with your Azure subscription.
2. Choose **Create a resource**, select **Containers**, and then select **Container Registry**.



3. Specify the values in the following table for each of the properties:

Property	**Value**
Registry name	Create a registry name that's globally unique within Azure, and contains 5-50 alphanumeric characters
Subscription	Select your default Azure subscription in which you are allowed to create and manage resources.

Property	Value
Resource Group	Create a new resource group with the name **learn-deploy-container-acr-rg** so that it will be easier to clean up these resources when you're finished with the module. If you choose a different resource group name, remember it for the rest of the exercises in this module.
Location	Select a location that is close to you.
Admin user	**Enable**
SKU	**Premium**

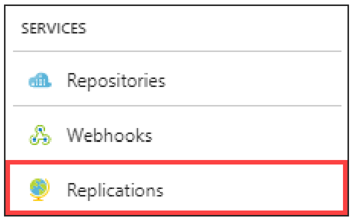
4. Click **Create**. Wait until the container registry has been created before you continue.

   Tip
   Because Azure container registries are typically long-lived resources that are used across multiple container hosts, we recommend that you create your registry in its own resource group. As you configure geo-replicated registries and webhooks, these additional resources are placed in the same resource group.

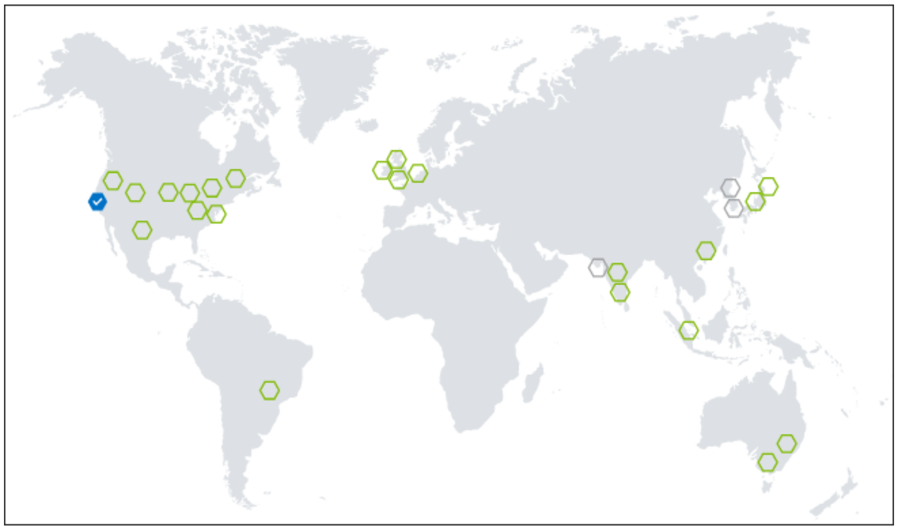
Configure geo-replication

Now that you have a Premium registry, you can configure geo-replication. Your web app, which you configure in the next tutorial to run in two regions, can then pull its container images from the nearest registry.

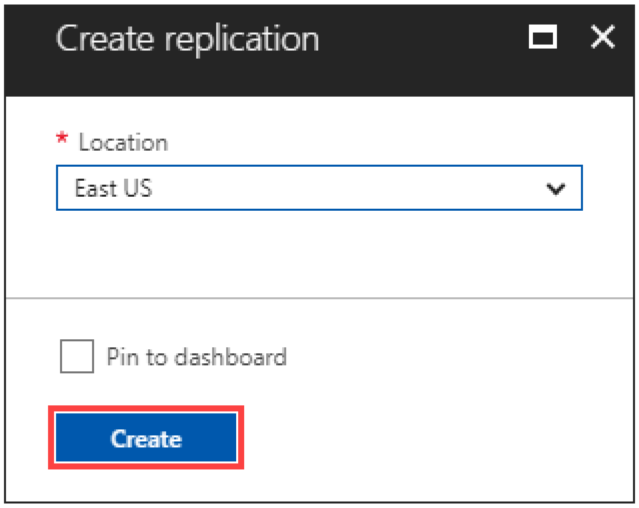
Navigate to your new container registry in the Azure portal and select **Replications** under **SERVICES**:



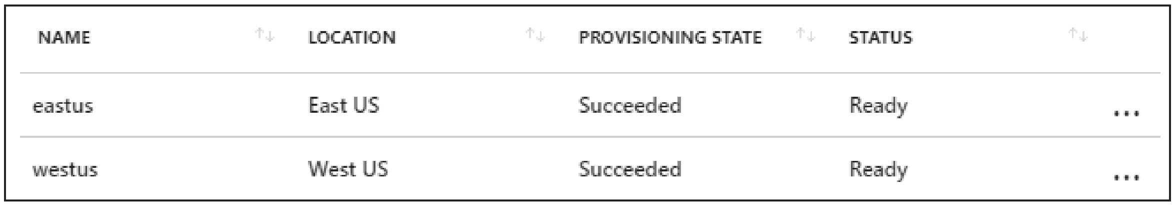
A map is displayed showing green hexagons representing Azure regions available for geo-replication:



Replicate your registry to the East US region by selecting its green hexagon, then select **Create** under **Create replication**:



When the replication is complete, the portal reflects *Ready* for both regions. Use the **Refresh** button to refresh the status of the replication; it can take a minute or so for the replicas to be created and synchronized.



NAME	LOCATION	PROVISIONING STATE	STATUS	
eastus	East US	Succeeded	Ready	...
westus	West US	Succeeded	Ready	...

Container registry login

Now that you've configured geo-replication, build a container image and push it to your registry. You must first log in to your ACR instance before pushing images to it.

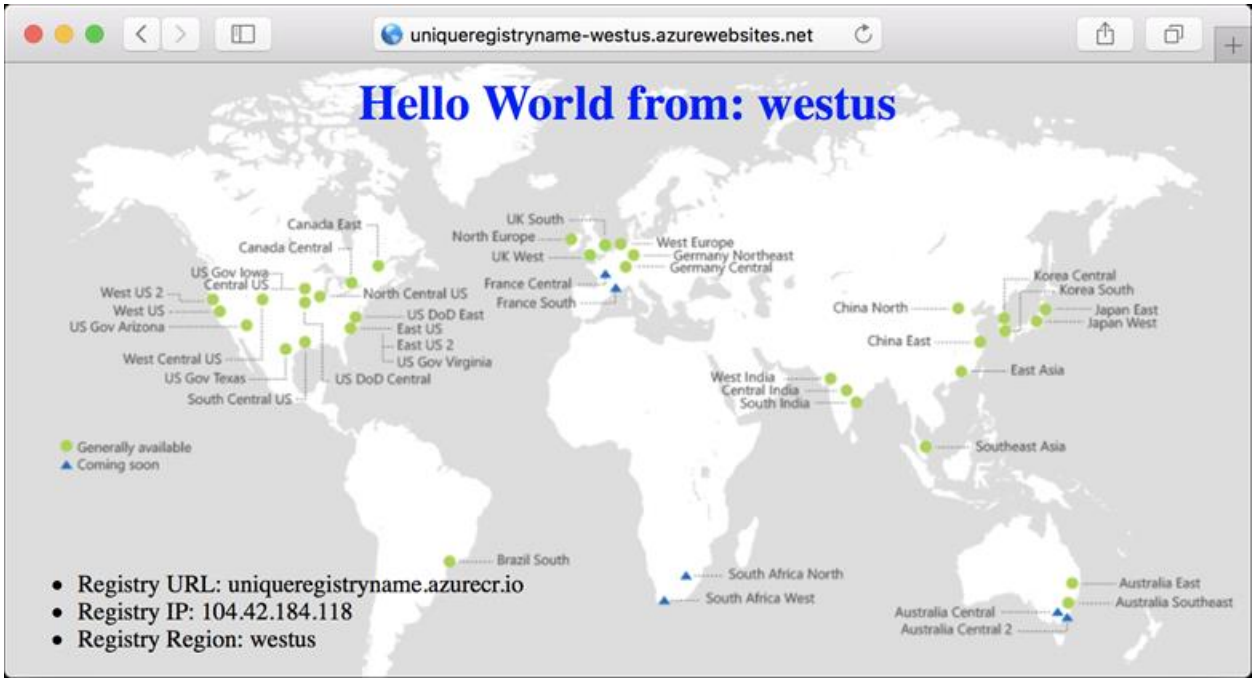
Use the az acr login command to authenticate and cache the credentials for your registry. Replace `<acrName>` with the name of the registry you created earlier.

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

The command returns `Login Succeeded` when complete.

Get application code

The sample in this tutorial includes a small web application built with ASP.NET Core. The app serves an HTML page that displays the region from which the image was deployed by Azure Container Registry.



Hello World from: westus

- Registry URL: uniqueregistryname.azurecr.io
- Registry IP: 104.42.184.118
- Registry Region: westus

Use git to download the sample into a local directory, and `cd` into the directory:

```
git clone https://github.com/Azure-Samples/acr-helloworld.git
cd acr-helloworld
```

If you don't have `git` installed, you can download directly from here.

Update Dockerfile

The Dockerfile included in the sample shows how the container is built. It starts from an official [aspnetcore]() image, copies the application files into the container, installs dependencies, compiles the output using the official [aspnetcore-build]() image, and finally, builds an optimized aspnetcore image.

The [Dockerfile]() is located at `./AcrHelloworld/Dockerfile` in the cloned source. Review the Dockerfile because you need to make changes for this exercise. Note the highlighted text below, you will need to update the Dockerfile to include your Registry name.

```
FROM microsoft/aspnetcore:2.0 AS base
# Update <acrName> with the name of your registry
# Example: uniqueregistryname.azurecr.io
ENV DOCKER_REGISTRY <acrName>.azurecr.io
WORKDIR /app
EXPOSE 80

FROM microsoft/aspnetcore-build:2.0 AS build
WORKDIR /src
COPY *.sln ./
COPY AcrHelloworld/AcrHelloworld.csproj AcrHelloworld/
RUN dotnet restore
COPY . .
WORKDIR /src/AcrHelloworld
RUN dotnet build -c Release -o /app

FROM build AS publish
RUN dotnet publish -c Release -o /app

FROM base AS production
WORKDIR /app
COPY --from=publish /app .
ENTRYPOINT ["dotnet", "AcrHelloworld.dll"]
```

The application in the *acr-helloworld* image tries to determine the region from which its container was deployed by querying DNS for information about the registry's login server. You must specify your registry login server's fully qualified domain name (FQDN) in the `DOCKER_REGISTRY` environment variable in the Dockerfile.

First, get the registry's login server with the `az acr show` command. Replace `<acrName>` with the name of the registry you created in previous steps.

```
az acr show --name <acrName> --query "{acrLoginServer:loginServer}" --output table
```

Output:

```
AcrLoginServer
-----------------------------
uniqueregistryname.azurecr.io
```

Next, update the `ENV DOCKER_REGISTRY` line in the Dockerfile with the FQDN of your registry's login server. This example reflects the example registry name, *uniqueregistryname*:

```
ENV DOCKER_REGISTRY uniqueregistryname.azurecr.io
```

Build container image

Now that you've updated the Dockerfile with the FQDN of your registry login server, you can use `docker build` to create the container image. Run the following command to build the image and tag it with the URL of your private registry, again replacing `<acrName>` with the name of your registry:

```
docker build . -f ./AcrHelloworld/Dockerfile -t <acrName>.azurecr.io/acr-
helloworld:v1
```

Use `docker images` to see the built and tagged image:

```
$ docker images
REPOSITORY                                      TAG    IMAGE ID        CREATED
SIZE
uniqueregistryname.azurecr.io/acr-helloworld    v1     01ac48d5c8cf    About a minute
ago    284MB
```

Push image to Azure Container Registry

Next, use the `docker push` command to push the *acr-helloworld* image to your registry. Replace `<acrName>` with the name of your registry.

```
docker push <acrName>.azurecr.io/acr-helloworld:v1
```

Because you've configured your registry for geo-replication, your image is automatically replicated to both the *West US* and *East US* regions with this single `docker push` command.

Summary

In this tutorial, you created a private, geo-replicated container registry, built a container image, and then pushed that image to your registry.

Deploy a web app from a geo-replicated Azure container registry

In this exercise, you take advantage of the geo-replicated registry by deploying the container to Web App instances in two different Azure regions. Each instance then pulls the container image from the closest registry.

In this tutorial:

- Deploy a container image to two *Web Apps for Containers* instances
- Verify the deployed application

Automatic deployment to Web Apps for Containers

Azure Container Registry provides support for deploying containerized applications directly to Web Apps for Containers. In this exercise, you use the Azure portal to deploy the container image created in the previous tutorial to two web app plans located in different Azure regions.

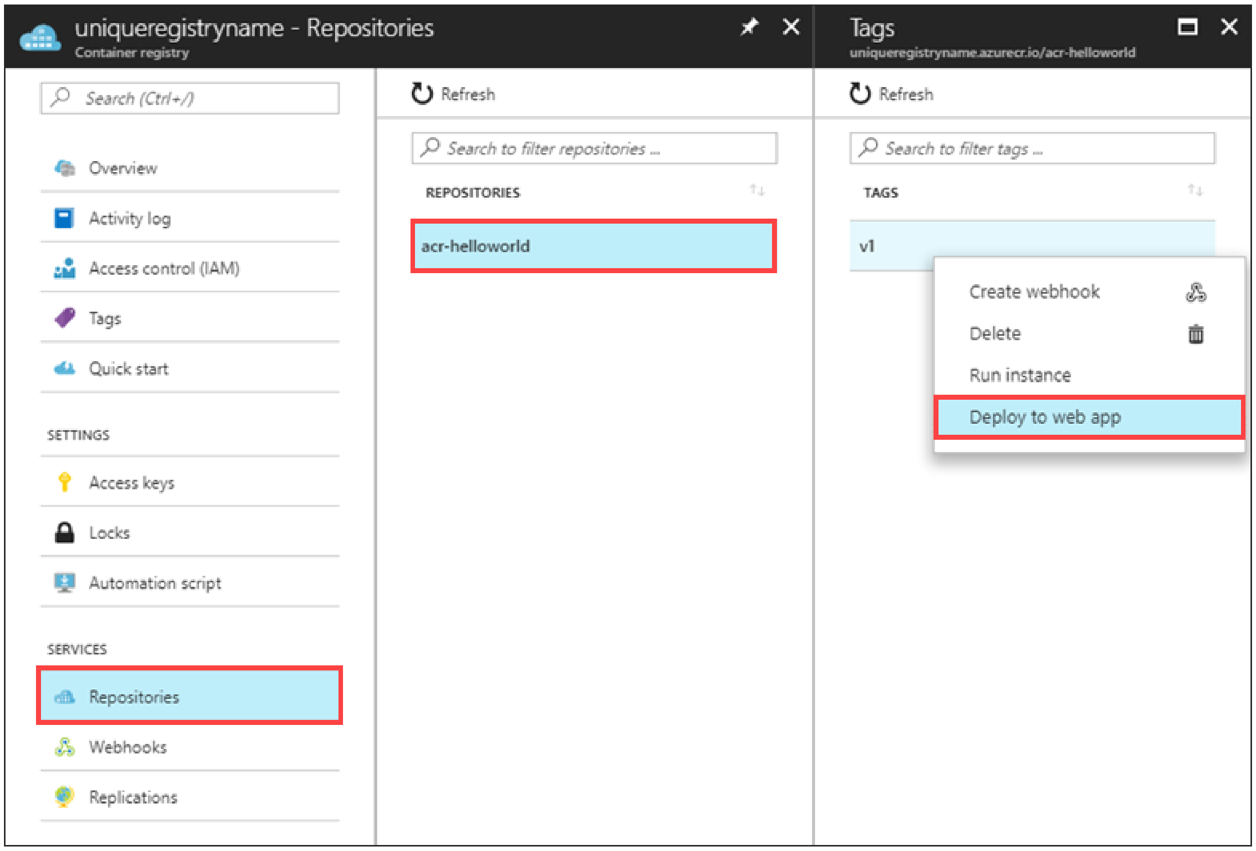
When you deploy a web app from a container image in your registry, and you have a geo-replicated registry in the same region, Azure Container Registry creates an image deployment webhook for you. When you push a new image to your container repository, the webhook picks up the change and automatically deploys the new container image to your web app.

Deploy a Web App for Containers instance

In this step, you create a Web App for Containers instance in the *West US* region.

Sign in to the Azure portal and navigate to the registry you created in the previous tutorial.

Select **Repositories** > **acr-helloworld**, then right-click on the **v1** tag under **Tags** and select **Deploy to web app**:



If "Deploy to web app" is disabled, you might not have enabled the registry admin user as directed in the previous exercise. You can enable the admin user in **Settings** > **Access keys** in the Azure portal.

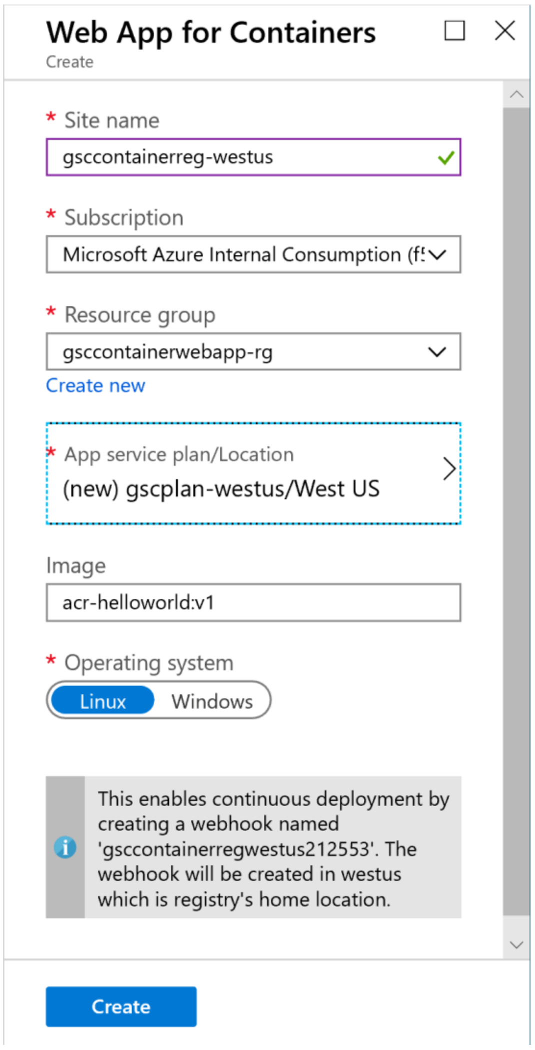
Under **Web App for Containers** that's displayed after you select "Deploy to web app," specify the following values for each setting:

Setting	**Value**
Site Name	A globally unique name for the web app. In this example, we use the format `<acrName>-westus` to easily identify the registry and region the web app is deployed from.
Resource Group	**Use existing** > `myResourceGroup`
App service plan/Location	Create a new plan named `plan-westus` in the **West US** region.

Setting	Value
Image	`acr-helloworld:v1`
Operating system	Linux

Note
When you create a new app service plan to deploy your containerized app, a default plan is automatically selected to host your application. The default plan depends on the operating system setting.

Select **Create** to provision the web app to the *West US* region.


Web App for Containers
Create

* Site name: gsccontainerreg-westus
* Subscription: Microsoft Azure Internal Consumption (f!
* Resource group: gsccontainerwebapp-rg
Create new
* App service plan/Location: (new) gscplan-westus/West US
Image: acr-helloworld:v1
* Operating system: [✓ Linux] Windows

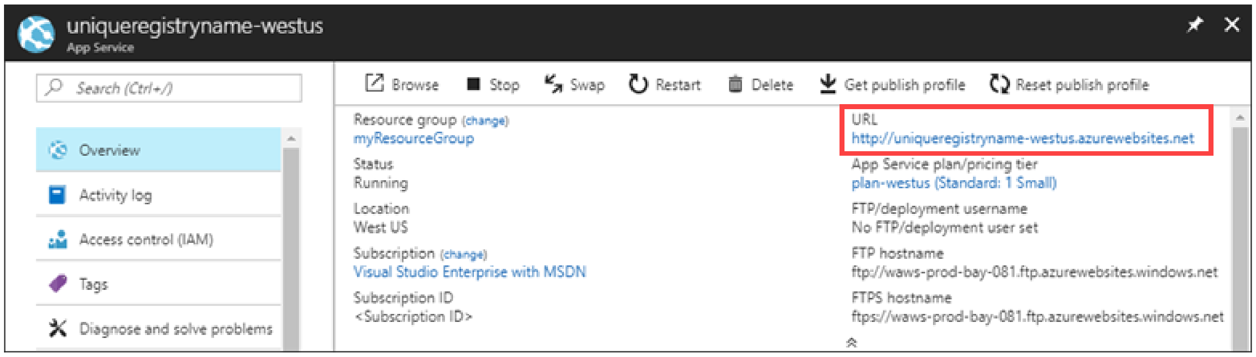
This enables continuous deployment by creating a webhook named 'gsccontainerregwestus212553'. The webhook will be created in westus which is registry's home location.

View the deployed web app

When deployment is complete, you can view the running application by navigating to its URL in your browser.

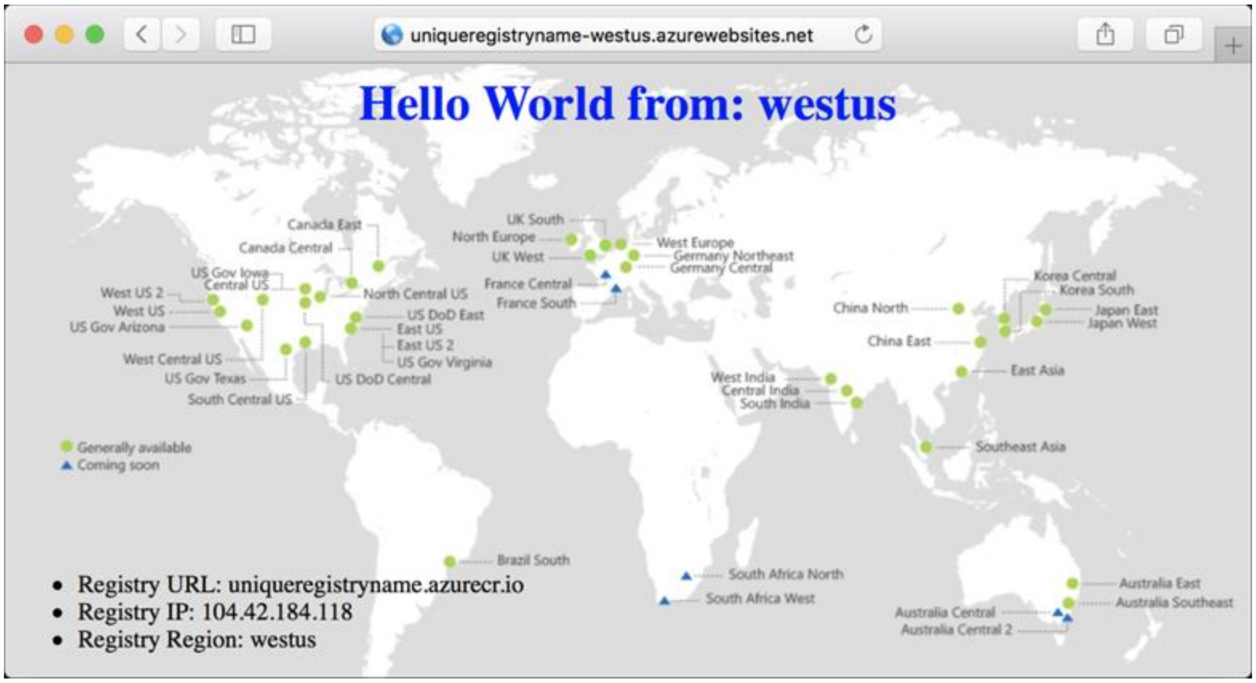
In the portal, select **App Services**, then the web app you provisioned in the previous step. In this example, the web app is named *uniqueregistryname-westus*.

Select the hyperlinked URL of the web app in the top-right of the **App Service** overview to view the running application in your browser.



Resource group (change)
myResourceGroup
Status
Running
Location
West US
Subscription (change)
Visual Studio Enterprise with MSDN
Subscription ID
<Subscription ID>
URL
http://uniqueregistryname-westus.azurewebsites.net
App Service plan/pricing tier
plan-westus (Standard: 1 Small)
FTP/deployment username
No FTP/deployment user set
FTP hostname
ftp://waws-prod-bay-081.ftp.azurewebsites.windows.net
FTPS hostname
ftps://waws-prod-bay-081.ftp.azurewebsites.windows.net

Once the Docker image is deployed from your geo-replicated container registry, the site displays an image representing the Azure region hosting the container registry.



Hello World from: westus

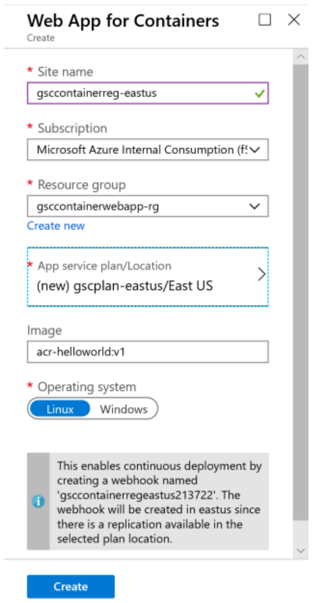
- Registry URL: uniqueregistryname.azurecr.io
- Registry IP: 104.42.184.118
- Registry Region: westus

Deploy second Web App for Containers instance

Use the procedure outlined in the previous section to deploy a second web app to the *East US* region. Under **Web App for Containers**, specify the following values:

Setting	Value
Site Name	A globally unique name for the web app. In this example, we use the format `<acrName>-eastus` to easily identify the registry and region the web app is deployed from.
Resource Group	**Use existing** > `myResourceGroup`
App service plan/Location	Create a new plan named `plan-eastus` in the **East US** region.
Image	`acr-helloworld:v1`
Operating system	Linux

Select **Create** to provision the web app to the *East US* region.


Web App for Containers
Create

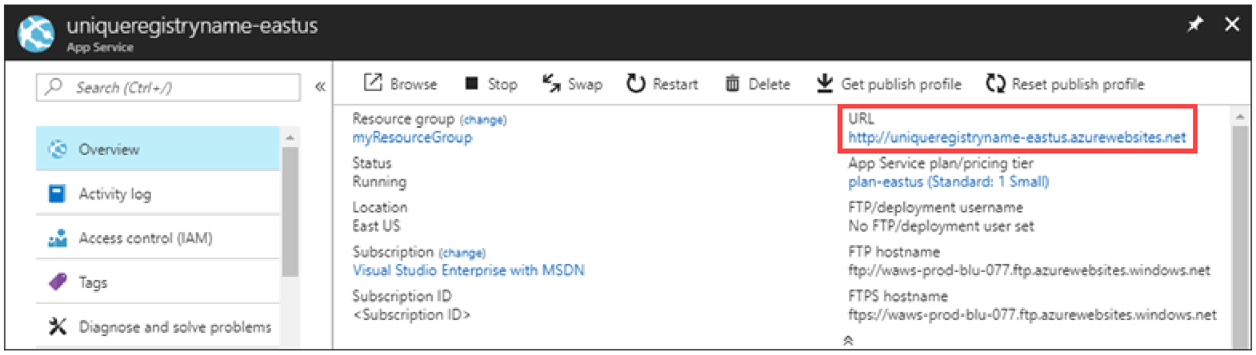
* Site name: gsccontainerreg-eastus
* Subscription: Microsoft Azure Internal Consumption (f!
* Resource group: gsccontainerwebapp-rg
Create new
* App service plan/Location: (new) gscplan-eastus/East US
Image: acr-helloworld:v1
* Operating system: [✓ Linux] Windows

This enables continuous deployment by creating a webhook named 'gsccontainerregeastus213722'. The webhook will be created in eastus since there is a replication available in the selected plan location.

Create

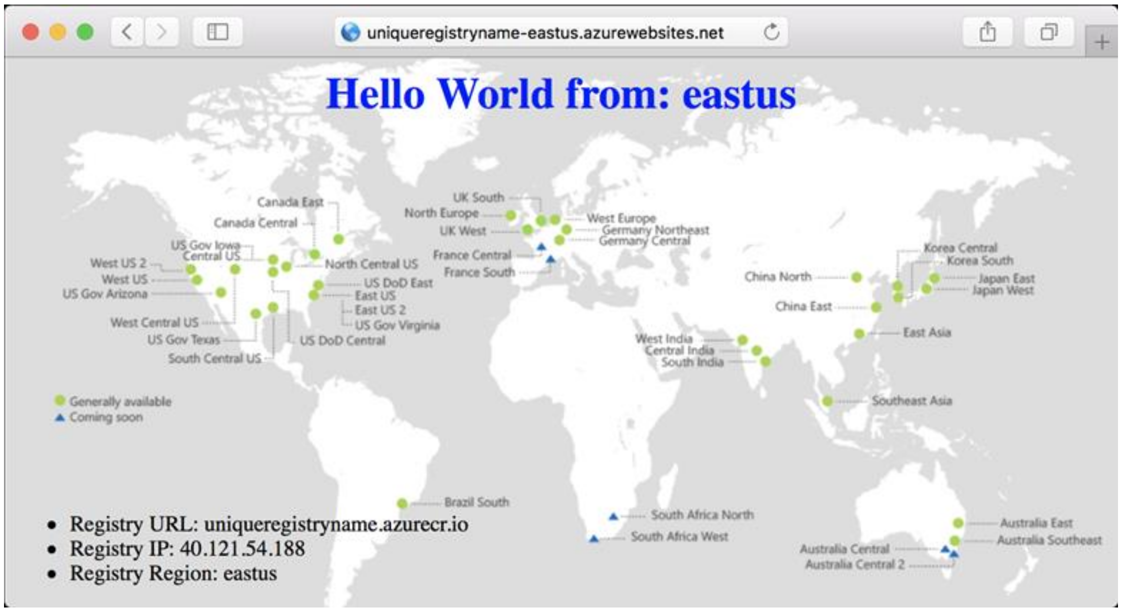
View the deployed web app

As before, you can view the running application by navigating to its URL in your browser. In the portal, select **App Services**, then the web app you provisioned in the previous step. In this example, the web app is named *uniqueregistryname-eastus*. Select the hyperlinked URL of the web app in the top-right of the **App Service overview** to view the running application in your browser.



Resource group (change)
myResourceGroup
Status
Running
Location
East US
Subscription (change)
Visual Studio Enterprise with MSDN
Subscription ID
<Subscription ID>
URL
http://uniqueregistryname-eastus.azurewebsites.net
App Service plan/pricing tier
plan-eastus (Standard: 1 Small)
FTP/deployment username
No FTP/deployment user set
FTP hostname
ftp://waws-prod-blu-077.ftp.azurewebsites.windows.net
FTPS hostname
ftps://waws-prod-blu-077.ftp.azurewebsites.windows.net

Once the Docker image is deployed from your geo-replicated container registry, the site displays an image representing the Azure region hosting the container registry.



Hello World from: eastus

- Registry URL: uniqueregistryname.azurecr.io
- Registry IP: 40.121.54.188
- Registry Region: eastus

Summary

In this exercise, you deployed two Web App for Containers instances from a geo-replicated Azure container registry.

Push an updated container image to a geo-replicated container registry for regional web app deployments

In this exercise, you first modify the application, then build a new container image and push it to your geo-replicated registry. Finally, you view the change, deployed automatically by Azure Container Registry webhooks, in both Web App instances.

In this exercise:

- Modify the web application HTML
- Build and tag the Docker image
- Push the change to Azure Container Registry
- View the updated app in two different regions

Modify the web application

In this step, make a change to the web application that will be highly visible once you push the updated container image to Azure Container Registry.

Find the `AcrHelloworld/Views/Home/Index.cshtml` file in the application source you cloned from GitHub in the first exercise and open it in your favorite text editor. Add the following line below the existing `<h1>` line:

```
<h1>MODIFIED</h1>
```

Your modified `Index.cshtml` should look similar to:

```
@{
    ViewData["Title"] = "Azure Container Registry :: Geo-replication";
}
<style>
    body {
        background-image: url('images/azure-regions.png');
        background-size: cover;
    }
    .footer {
        position: fixed;
        bottom: 0px;
        width: 100%;
    }
</style>

<h1 style="text-align:center;color:blue">Hello World from:  @ViewData["REGION"]</h1>
<h1>MODIFIED</h1>
<div class="footer">
    <ul>
        <li>Registry URL: @ViewData["REGISTRYURL"]</li>
        <li>Registry IP: @ViewData["REGISTRYIP"]</li>
        <li>Registry Region: @ViewData["REGION"]</li>
    </ul>
</div>
```

Rebuild the image

Now that you've updated the web application, rebuild its container image. As before, use the fully qualified image name, including the login server's fully qualified domain name (FQDN), for the tag:

```
docker build . -f ./AcrHelloworld/Dockerfile -t <acrName>.azurecr.io/acr-
helloworld:v1
```

Push image to Azure Container Registry

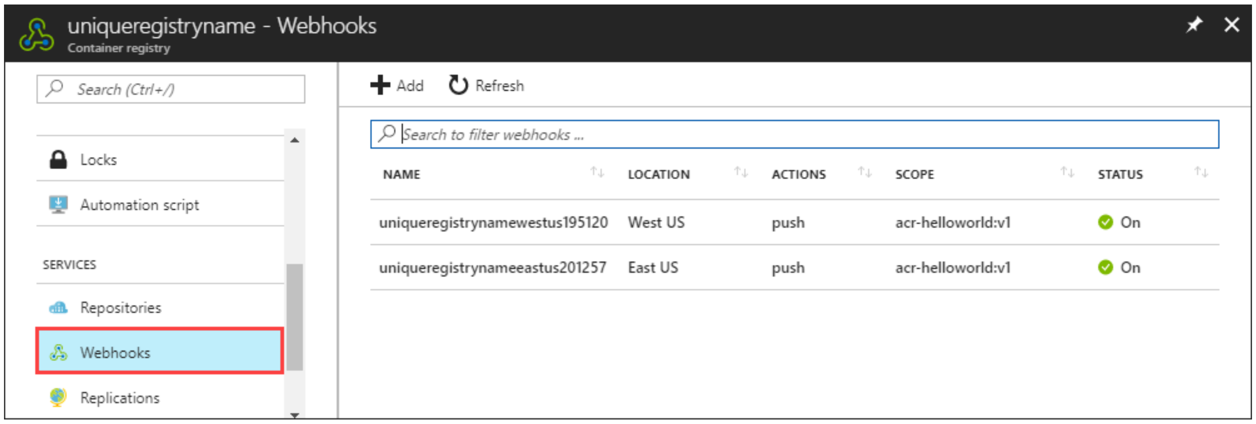
Next, push the updated *acr-helloworld* container image to your geo-replicated registry. Here, you're executing a single `docker push` command to deploy the updated image to the registry replicas in both the *West US* and *East US* regions.

```
docker push <acrName>.azurecr.io/acr-helloworld:v1
```

View the webhook logs

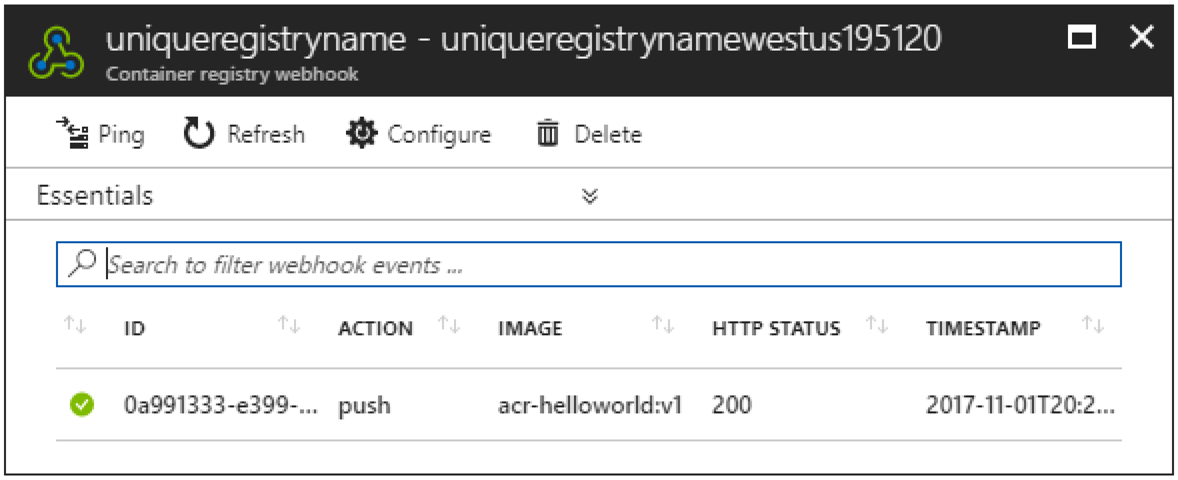
While the image is being replicated, you can see the Azure Container Registry webhooks being triggered.

To see the regional webhooks that were created when you deployed the container to *Web Apps for Containers* in a previous exercise, navigate to your container registry in the Azure portal, then select **Webhooks** under **SERVICES**.



NAME	LOCATION	ACTIONS	SCOPE	STATUS
uniqueregistrynamewestus195120	West US	push	acr-helloworld:v1	✓ On
uniqueregistrynameeastus201257	East US	push	acr-helloworld:v1	✓ On

Select each Webhook to see the history of its calls and responses. You should see a row for the **push** action in the logs of both Webhooks. Here, the log for the Webhook located in the *West US* region shows the **push** action triggered by the `docker push` in the previous step:

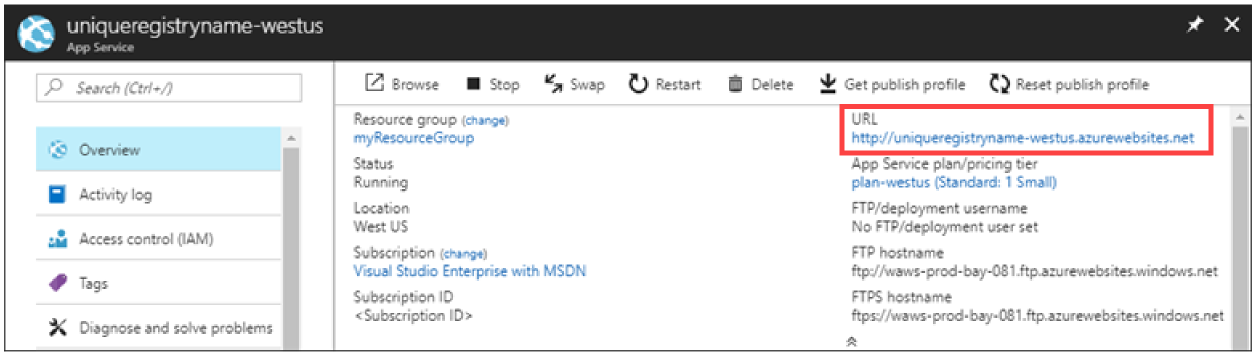

uniqueregistryname - uniqueregistrynamewestus195120
Container registry webhook
Ping
Refresh
Configure
Delete
Essentials
Search to filter webhook events ...

	ID	ACTION	IMAGE	HTTP STATUS	TIMESTAMP
✓	0a991333-e399-...	push	acr-helloworld:v1	200	2017-11-01T20:2...

View the updated web app

The Webhooks notify Web Apps that a new image has been pushed to the registry, which automatically deploys the updated container to the two regional web apps.

Verify that the application has been updated in both deployments by navigating to both regional Web App deployments in your web browser. As a reminder, you can find the URL for the deployed web app in the top-right of each App Service overview tab.


uniqueregistryname-westus
App Service

Resource group (change)
myResourceGroup

Status
Running

Location
West US

Subscription (change)
Visual Studio Enterprise with MSDN

Subscription ID
<Subscription ID>

URL
http://uniqueregistryname-westus.azurewebsites.net

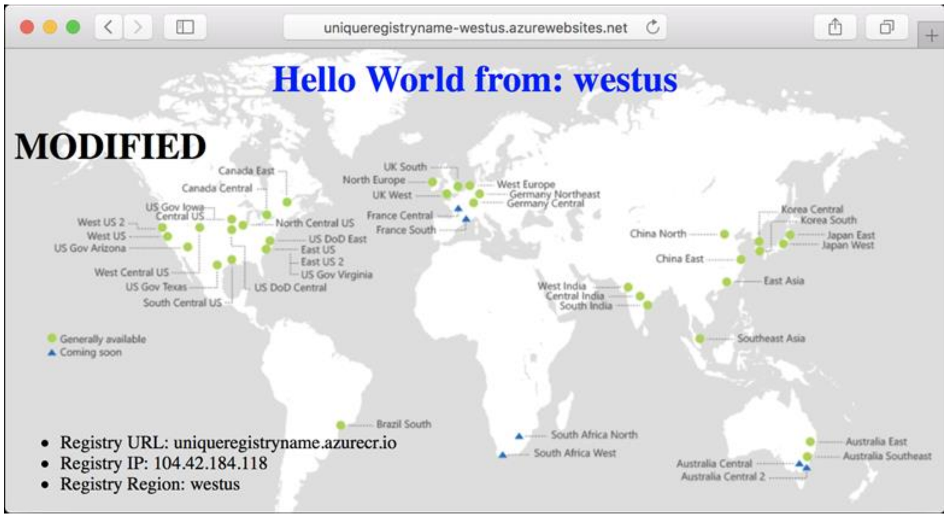
App Service plan/pricing tier
plan-westus (Standard: 1 Small)

FTP/deployment username
No FTP/deployment user set

FTP hostname
ftp://waws-prod-bay-081.ftp.azurewebsites.windows.net

FTPS hostname
ftps://waws-prod-bay-081.ftp.azurewebsites.windows.net

To see the updated application, select the link in the App Service overview. Here's an example view of the app running in *West US*:

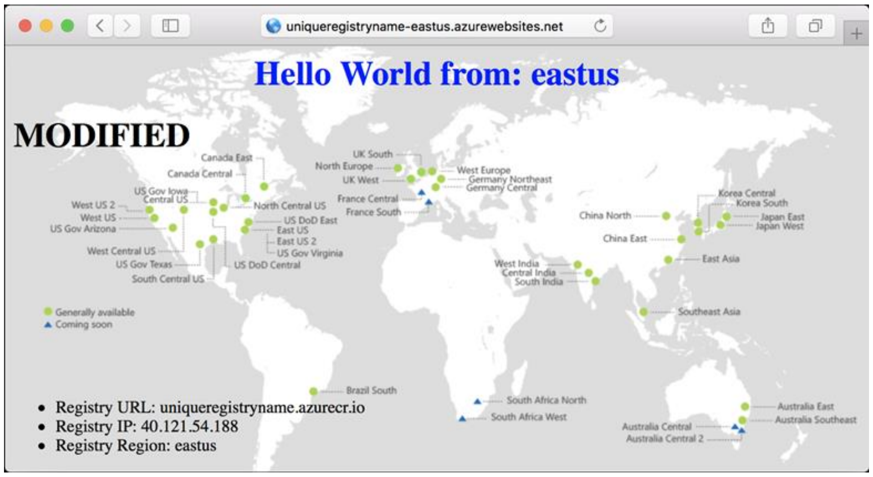

uniqueregistryname-westus.azurewebsites.net

Hello World from: westus

MODIFIED

- Registry URL: uniqueregistryname.azurecr.io
- Registry IP: 104.42.184.118
- Registry Region: westus

Verify that the updated container image was also deployed to the *East US* deployment by viewing it in your browser.


uniqueregistryname-eastus.azurewebsites.net

Hello World from: eastus

MODIFIED

- Registry URL: uniqueregistryname.azurecr.io
- Registry IP: 40.121.54.188
- Registry Region: eastus

With a single `docker push`, you've automatically updated the web application running in both regional Web App deployments. And, Azure Container Registry served the container images from the repositories located closest to each deployment.

Summary

In this exercise, you updated and pushed a new version of the web application container to your geo-replicated registry. Webhooks in Azure Container Registry notified Web Apps for Containers of the update, which triggered a local pull from the nearest registry replica.