

Configure a single public IP address for outbound and inbound traffic to a container group

Article • 01/13/2023

Setting up a [container group](#) with an external-facing IP address allows external clients to use the IP address to access a container in the group. For example, a browser can access a web app running in a container. However, currently a container group uses a different IP address for outbound traffic. This egress IP address isn't exposed programmatically, which makes container group monitoring and configuration of client firewall rules more complex.

This article provides steps to configure a container group in a [virtual network](#) integrated with [Azure Firewall](#). By setting up a user-defined route to the container group and firewall rules, you can route and identify traffic to and from the container group. Container group ingress and egress use the public IP address of the firewall. A single egress IP address can be used by multiple container groups deployed in the virtual network's subnet delegated to Azure Container Instances.

In this article, you use the Azure CLI to create the resources for this scenario:

- Container groups deployed on a delegated subnet [in the virtual network](#)
- An Azure firewall deployed in the network with a static public IP address
- A user-defined route on the container groups' subnet
- A NAT rule for firewall ingress and an application rule for egress

You then validate ingress and egress from example container groups through the firewall.

If you don't have an [Azure subscription](#), create an [Azure free account](#) [↗](#) before you begin.

Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#). For more information, see [Quickstart for Bash in Azure Cloud Shell](#).



- If you prefer to run CLI reference commands locally, [install](#) the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see [How to run the Azure CLI in a Docker container](#).
 - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see [Sign in with the Azure CLI](#).
 - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
 - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).

Launch Azure Cloud Shell

The Azure Cloud Shell is a free interactive shell that you can use to run the steps in this article. It has common Azure tools preinstalled and configured to use with your account.

To open the Cloud Shell, just select **Try it** from the upper right corner of a code block. You can also launch Cloud Shell in a separate browser tab by going to <https://shell.azure.com> [↗](#).

When Cloud Shell opens, verify that **Bash** is selected for your environment. Subsequent sessions will use Azure CLI in a Bash environment, Select **Copy** to copy the blocks of code, paste it into the Cloud Shell, and press **Enter** to run it.

Sign in to Azure

Cloud Shell is automatically authenticated under the initial account signed-in with. Use the following script to sign in using a different subscription, replacing `<Subscription ID>` with your Azure Subscription ID. If you don't have an [Azure subscription](#), create an [Azure free account](#) [↗](#) before you begin.

Azure CLI

```
subscription="<subscriptionId>" # add subscription here

az account set -s $subscription # ...or use 'az login'
```

For more information, see [set active subscription](#) or [log in interactively](#)

 **Note**

To download the complete script, go to [full script](#).

Get started

This tutorial makes use of a randomized variable. If you are using an existing resource group, modify the value of this variable appropriately.

Azure CLI

```
resourceGroup=resourceGroup$RANDOM
```

Azure resource group: If you don't have an Azure resource group already, create a resource group with the [az group create](#) command. Modify the location value as appropriate.

Azure CLI

```
az group create --name $resourceGroup --location eastus
```

Deploy ACI in a virtual network


In a typical case, you might already have an Azure virtual network in which to deploy a container group. For demonstration purposes, the following commands create a virtual network and subnet when the container group is created. The subnet is delegated to Azure Container Instances.

The container group runs a small web app from the `aci-helloworld` image. As shown in other articles in the documentation, this image packages a small web app written in Node.js that serves a static HTML page.

Create the container group with the [az container create](#) command:

Azure CLI

```
az container create \
  --name appcontainer \
  --resource-group $resourceGroup \
  --image mcr.microsoft.com/azuredocs/aci-helloworld \
  --vnet aci-vnet \
  --vnet-address-prefix 10.0.0.0/16 \
  --subnet aci-subnet \
  --subnet-address-prefix 10.0.0.0/24
```

 **Tip**

Adjust the value of `--subnet address-prefix` for the IP address space you need in your subnet. The smallest supported subnet is /29, which provides eight IP addresses. Some IP addresses are reserved for use by Azure.

For use in a later step, get the private IP address of the container group by running the `[az container show][az-container-show]` command:

Azure CLI

```
aciPrivateIp="$(az container show --name appcontainer \
  --resource-group $resourceGroup \
```

```
--query ipAddress.ip --output tsv)"
```

Deploy Azure Firewall in network

In the following sections, use the Azure CLI to deploy an Azure firewall in the virtual network. For background, see [Tutorial: Deploy and configure Azure Firewall using the Azure portal](#).

First, use the `az network vnet subnet create` to add a subnet named AzureFirewallSubnet for the firewall. AzureFirewallSubnet is the *required* name of this subnet.

Azure CLI

```
az network vnet subnet create \
  --name AzureFirewallSubnet \
  --resource-group $resourceGroup \
  --vnet-name aci-vnet \
  --address-prefix 10.0.1.0/26
```

Use the following [Azure CLI commands](#) to create a firewall in the subnet.

If not already installed, add the firewall extension to the Azure CLI using the `az extension add` command:

Azure CLI

```
az extension add --name azure-firewall
```

Create the firewall resources using the `az network firewall create` command:

Azure CLI

```
az network firewall create \
  --name myFirewall \
  --resource-group $resourceGroup \
  --location eastus

az network public-ip create \
  --name fw-pip \
  --resource-group $resourceGroup \
  --location eastus \
  --allocation-method static \
  --sku standard

az network firewall ip-config create \
  --firewall-name myFirewall \
  --name FW-config \
  --public-ip-address fw-pip \
  --resource-group $resourceGroup \
  --vnet-name aci-vnet
```

Update the firewall configuration using the `az network firewall update` command:

Azure CLI

```
az network firewall update \
  --name myFirewall \
  --resource-group $resourceGroup
```

Get the firewall's private IP address using the `az network firewall ip-config list` command. This private IP address is used in a later command.

Azure CLI

```
fwPrivateIp="$(az network firewall ip-config list \
  --resource-group $resourceGroup \
  --firewall-name myFirewall \
  --query "[].privateIpAddress" --output tsv)"
```

Get the firewall's public IP address using the `az network public-ip show` command. This public IP address is used in a later command.

Azure CLI

```
fwPublicIp="$(az network public-ip show \
  --name fw-pip \
  --resource-group $resourceGroup \
  --query ipAddress --output tsv)"
```

Define user-defined route on ACI subnet

Define a use-defined route on the ACI subnet, to divert traffic to the Azure firewall. For more information, see [Route network traffic](#).

Create route table

First, run the following [az network route-table create](#) command to create the route table. Create the route table in the same region as the virtual network.

Azure CLI

```
az network route-table create \
  --name Firewall-rt-table \
  --resource-group $resourceGroup \
  --location eastus \
  --disable-bgp-route-propagation true
```

Create route

Run [az network-route-table route create](#) to create a route in the route table. To route traffic to the firewall, set the next hop type to `VirtualAppliance`, and pass the firewall's private IP address as the next hop address.

Azure CLI

```
az network route-table route create \
  --resource-group $resourceGroup \
  --name DG-Route \
  --route-table-name Firewall-rt-table \
  --address-prefix 0.0.0.0/0 \
  --next-hop-type VirtualAppliance \
  --next-hop-ip-address $fwPrivateIp
```

Associate route table to ACI subnet

Run the [az network vnet subnet update](#) command to associate the route table with the subnet delegated to Azure Container Instances.

Azure CLI

```
az network vnet subnet update \
  --name aci-subnet \
  --resource-group $resourceGroup \
  --vnet-name aci-vnet \
  --address-prefixes 10.0.0.0/24 \
  --route-table Firewall-rt-table
```

Configure rules on firewall

By default, Azure Firewall denies (blocks) inbound and outbound traffic.

Configure NAT rule on firewall to ACI subnet

Create a [NAT rule](#) on the firewall to translate and filter inbound internet traffic to the application container you started previously in the network. For details, see [Filter inbound Internet traffic with Azure Firewall DNAT](#)

Create a NAT rule and collection by using the [az network firewall nat-rule create](#) command:

Azure CLI

```
az network firewall nat-rule create \  
  --firewall-name myFirewall \  
  --collection-name myNATCollection \  
  --action dnat \  
  --name myRule \  
  --protocols TCP \  
  --source-addresses '*' \  
  --destination-addresses $fwPublicIp \  
  --destination-ports 80 \  
  --resource-group $resourceGroup \  
  --translated-address $aciPrivateIp \  
  --translated-port 80 \  
  --priority 200
```

Add NAT rules as needed to filter traffic to other IP addresses in the subnet. For example, other container groups in the subnet could expose IP addresses for inbound traffic, or other internal IP addresses could be assigned to the container group after a restart.

Create outbound application rule on the firewall

Run the following [az network firewall application-rule create](#) command to create an outbound rule on the firewall. This sample rule allows access from the subnet delegated to Azure Container Instances to the FQDN `checkip.dyndns.org`. HTTP access to the site is used in a later step to confirm the egress IP address from Azure Container Instances.

Azure CLI

```
az network firewall application-rule create \  
  --collection-name myAppCollection \  
  --firewall-name myFirewall \  
  --name Allow-CheckIP \  
  --protocols Http=80 Https=443 \  
  --resource-group $resourceGroup \  
  --target-fqdns checkip.dyndns.org \  
  --source-addresses 10.0.0.0/24 \  
  --priority 200 \  
  --action Allow
```

Test container group access through the firewall

The following sections verify that the subnet delegated to Azure Container Instances is properly configured behind the Azure firewall. The previous steps routed both incoming traffic to the subnet and outgoing traffic from the subnet through the firewall.

Test ingress to a container group

Test inbound access to the `appcontainer` running in the virtual network by browsing to the firewall's public IP address. Previously, you stored the public IP address in variable `$FW_PUBLIC_IP`:

Azure CLI

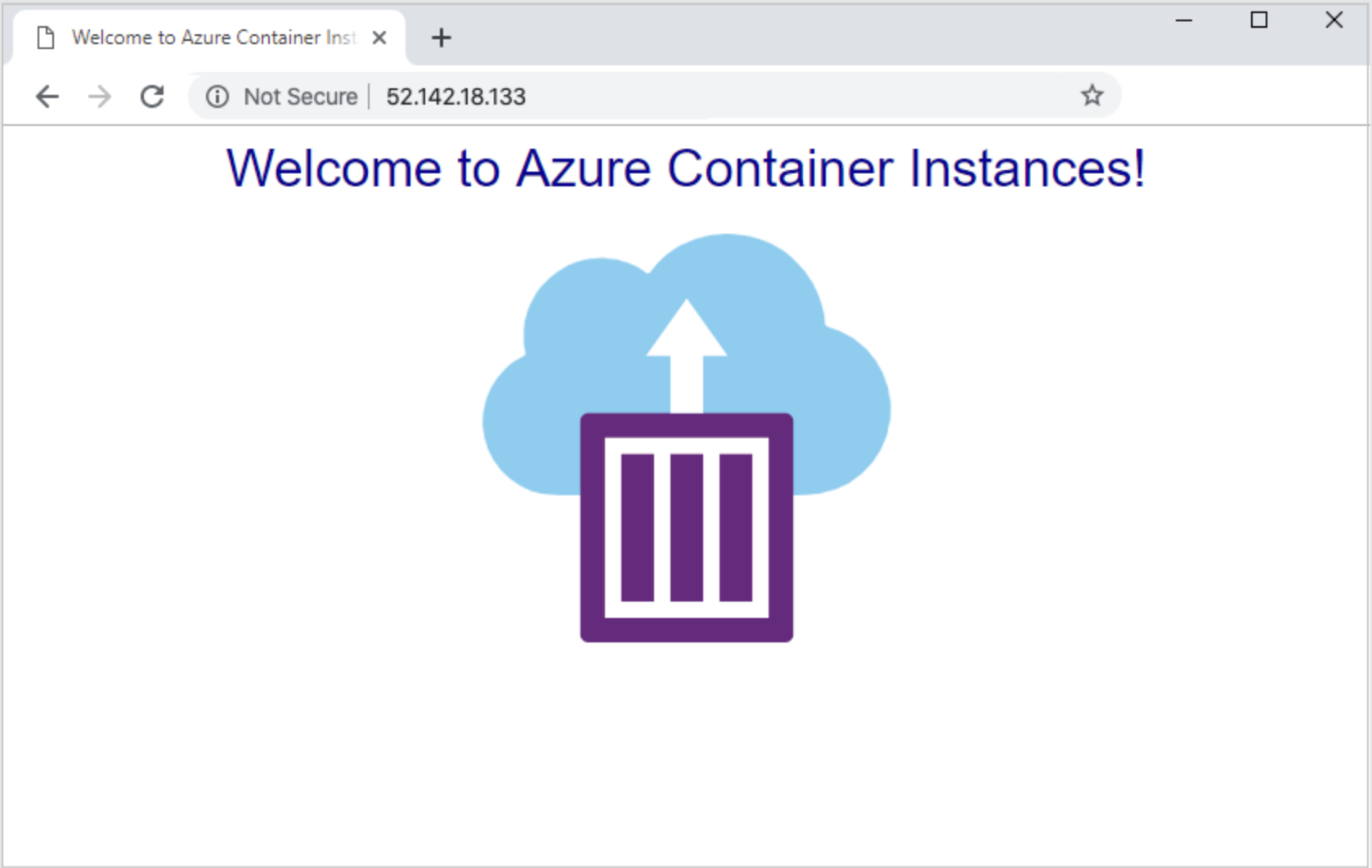
```
echo $fwPublicIp
```

Output is similar to:

Console

```
52.142.18.133
```

If the NAT rule on the firewall is configured properly, you see the following when you enter the firewall's public IP address in your browser:



Test egress from a container group

Deploy the following sample container into the virtual network. When it runs, it sends a single HTTP request to `http://checkip.dyndns.org`, which displays the IP address of the sender (the egress IP address). If the application rule on the firewall is configured properly, the firewall's public IP address is returned.

Azure CLI

```
az container create \
  --resource-group $resourceGroup \
  --name testegress \
  --image mcr.microsoft.com/azuredocs/aci-tutorial-sidecar \
  --command-line "curl -s http://checkip.dyndns.org" \
  --restart-policy OnFailure \
  --vnet aci-vnet \
  --subnet aci-subnet
```

View the container logs to confirm the IP address is the same as the public IP address of the firewall.

Azure CLI

```
az container logs \
  --sed 's/$RESOURCE_GROUP_NAME/$resourceGroup/g' \
  --name testegress
```

Output is similar to:

Console

```
<html><head><title>Current IP Check</title></head><body>Current IP Address: 52.142.18.133</body>
</html>
```

Clean up resources

When no longer needed, you can use `az group delete` to remove the resource group and all related resources as follows. The `--no-wait` parameter returns control to the prompt without waiting for the operation to complete. The `--yes` parameter confirms that you wish to delete the resources without an additional prompt to do so.

Azure CLI

```
az group delete --name $resourceGroup --yes --no-wait
```

Next steps

In this article, you set up container groups in a virtual network behind an Azure firewall. You configured a user-defined route and NAT and application rules on the firewall. By using this configuration, you set up a single, static IP address for ingress and egress from Azure Container Instances.

For more information about managing traffic and protecting Azure resources, see the [Azure Firewall](#) documentation.