Blog post

Sunday, September 4, 2022 10:34 PM

Azure Container Apps (ACA) Networking - a 101 intro

Before the Agenda

Note: If you're new to ACA **and** Kubernetes, feel free to glide through this section and go straight to the next section.

When first diving into Container Apps, it helps to understand that it's an abstracted form of Kubernetes under the hood. With that in mind, it helps to translate some ACA terms to the more well-known Kubernetes terms.

ACA Concept	Kubernetes Concept
Container App Environment	Kubernetes Cluster
Container App	Kubernetes Service and Deployment
Container App Instances	Kubernetes Pods
Container Environment Public/Private Endpoint	Layer-7 Load Balancer Endpoint/Service

By association of the above terms, we can assert the following things which also apply to basic Kubernetes clusters:

- A Container App Environment can host Container Apps which can communicate to one another within that environment through DNS resolution.
- Container Apps load balances to one or multiple endpoints that correspond to the instances (pods) of the Container App.
- Container App Instances host one or more containers, all sharing ports at localhost.
- The IP endpoint associated with your Container App Environment corresponds to a Layer 7 Load Balancer and requires traffic with a host header in order for the appropriate Container App to receive the request.

Agenda

We'll be touching base on the networking considerations for the following ACA concepts:

- Container App Environments
  - Vnet Integration
- HTTP Ingress
- Service Discovery
  - DNS Resolution
- Container App Revisions
- Custom Domains

Container App Environments

App Environments host your apps - but of course.

Through DNS resolution, apps can communicate through HTTP/s with one another if the destination app is configured to accept HTTP/s ingress. HTTP/80 and HTTPS/443 are the only acceptable protocols that apps can accept traffic in.

App Environments can have a public endpoint or private endpoint. You have the option or obligation to integrate with a Vnet, respectively. If integrating with a Vnet, zone redundancy is also an option.

- Ref:
  - Deploying a public environment: Provide an external virtual network to an Azure Container Apps environment | Microsoft Docs
  - Deploying a private environment: Provide an internal virtual network to an Azure Container Apps environment | Microsoft Docs

-Vnet Integration

If Vnet integration is desired, a Container App Environment will require exclusive access to a subnet with a minimum size of /23. After the creation fo your first Container App, in addition to the ACA and ACA Environment resources created, an MC_ Resource Group (RG) will also be created for you.

- This RG will contain:
  - Reserved subnet IPs for pods (app instances)
  - Public Load Balancer with:
    - Public IPs for outbound internet traffic.
    - Public frontend IP for inbound internet traffic if environment is public-facing.
  - Internal Load Balancer with private IP for inbound Vnet traffic if the environment is internal -only

Container App Ingress

When publishing an app, you have the option of enabling HTTP/s ingress to receive requests from other Apps in that Environment.

Furthermore, you have the option of allowing ingress from outside the environment. Depending on the endpoint configured on the environment, **external traffic** is classified as:

- For public endpoint environments - traffic from the internet. If there's Vnet integration, this includes Vnet resources that leverage the public endpoint. Note that there is no private ingress for Vnet integrated public environments.
- For private endpoint environments - traffic from the Vnet.

Reference ***properties.configuration.ingress.external*** property below:

```
kind: containerapp
location: uksouth
```

```
name: hello
resourceGroup: aca
type: Microsoft.App/containerApps
tags:
 tagname: value
properties:
 managedEnvironmentId: /subscriptions/redacted/resourceGroups/aca/providers/Microsoft.App/managedEnvironments/managedEnvironme nt-aca-aks-vnet-private-uksouth
 configuration:
  activeRevisionsMode: Multiple
  ingress:
   external: true
   allowInsecure: false
   targetPort: 80
   traffic:
    - latestRevision: true
      weight: 30
   transport: Auto
```

Translating the yaml above - the presence of ***ingress*** asserts that an app accepts HTTP/s ingress within the environment. The ***external*** property confirms whether the app will accept traffic from the internet or the Vnet for public or private endpoint environments, respectively. The absence of ***ingress*** asserts no ingress is allowed from anywhere for this particular app.

For more details refer to: HTTP Ingress

Note: You can have a public environment, and an app with ingress disabled. This effectively isolates it completely. For a workload in which this is applicable, please refer to Background Processing. The reason

Depending on whether external ingress is allowed, your Container App will receive a different FQDN. We'll go into more detail on that in the next section.

Service Discovery

Your Container Apps will be reachable through an FQDN:

- If external traffic is enabled for ingress on the app, the FQDN follows this scheme:
  - <appName>.<environmentID>.<regionName>.azurecontainerapps.io
- If external traffic is disabled for ingress, the FQDN follows this scheme:
  - <appName>.internal.<environmentID>.<regionName>.azurecontainerapps.io

The external FQDN is:

- load balanced through the endpoint of the environment - its exposed to the internet or VNet.
- resolved by public Azure DNS (for external/public environment) or private Azure DNS zones (for internal/private environment).
  - For more details, see the DNS Resolution section.

The internal FQDN is:

- resolved and load balanced internally within the environment - the endpoint is not exposed.

The environmentID is generated upon creation of the App Environment and cannot be specified by the user. Thus it's important to confirm the environmentID prior to creating your apps for the purpose of specifying appropriate Container App env variables which will allow for intra environment communication through the use of an app's fully qualified domain name (FQDN).

The following command will retrieve the environmentID after the creation of the environment:

az containerapp env show --name <environmentName> --resource-group <resourceGroupName> -o tsv --query properties.defaultDomain

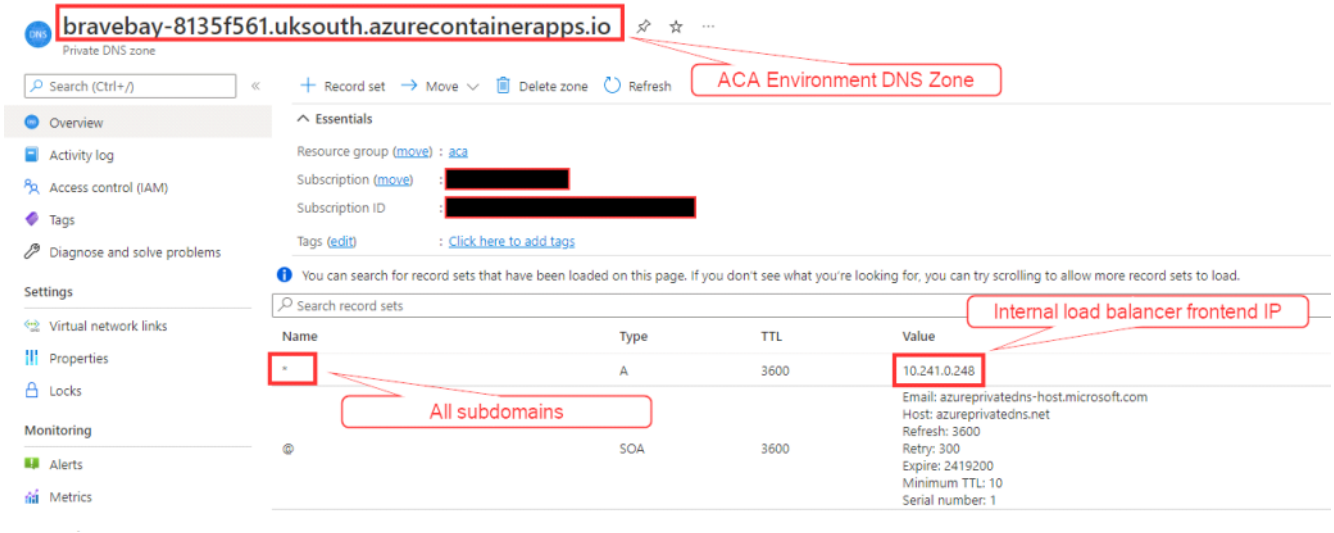
Ref: Connect applications in Azure Container Apps

-DNS Resolution

As previously discussed:

- Apps hosted in an external/public environment can have their FQDN resolved by Public Azure DNS.
- Apps hosted in internal/private environments require Azure Private DNS zones configurations for resolution of App FQDNs.

Below you'll see an example of an Azure Private DNS Zone created for environment **.bravebay-8135f561.uksouth.azurecontainerapps.io*.



The internal load balancer IP can be found in the MC_ resource group created for your internal environment. The above zone asserts that all subdomains for this zone should be sent to IP 10.241.0.248. When the DNS zone is linked to the integrated Vnet, resources within that Vnet will be able to resolve and reach your Container App.

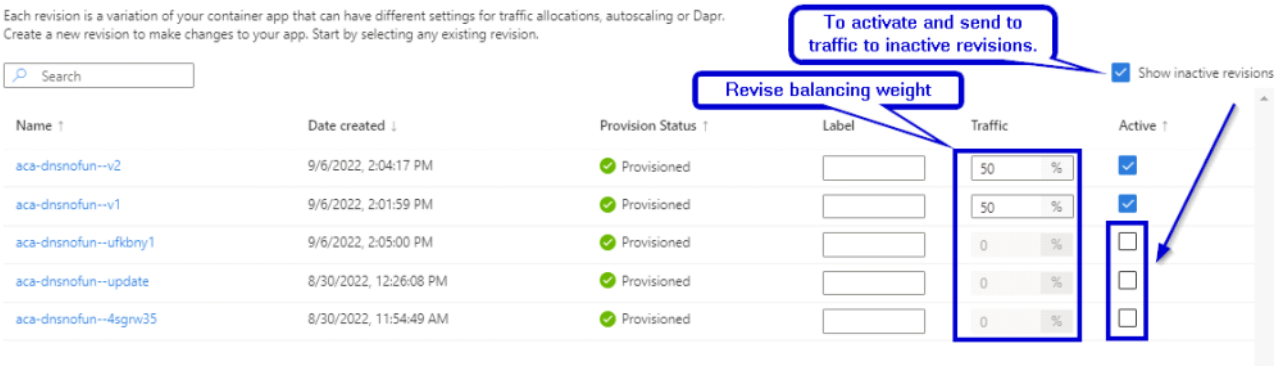
- For more info on private Azure DNS Zones, please see: [What is an Azure DNS private zone | Microsoft Docs]

Load Balancing between Container App Revisions

If your Container app is set to ***ActiveRevisionsMode:Multiple***, you have the option of load balancing between revisions.

Within the portal, this process is straight forward. Within the Container App blade, you:

- Navigate to *Revision Management*
  - Revision management
- Revise the revision mode to Multiple
  - Choose revision mode
- Revise the weight of traffic for each revision



For declarative deployments, yaml can be provided, e.g. *az containerapp update --yaml sample.yml --name <containerAppName> --resource-group >rgName>*

Below you can see an example of a Container App yaml declaration. These deployments don't necessarily act in an idempotent fashion when it comes to revision traffic weights.

- For a general reference of yaml deployments in ACA see: [Container Apps ARM template API specification | Microsoft Docs]

The example above asserts the following depending on whether the ***template*** for a revision results in the creation of a new revision, the update of an existing revision, or whether the template is not included at all:

- If the deployment includes non-existing template/revision:
  - New revision is created.
  - The traffic weight assigned to *latestRevision* is assigned to the revision created, if *latestRevision=true* - otherwise, the revision is created, but no traffic is routed to it.
- If the deployment includes pre-existing template/revision:
  - Revision is updated if necessary.
  - The traffic weight assigned to *latestRevision* is assigned to the ***last*** revision created, regardless of whether that revision is included in the template or not.
  - In order to change the traffic weight for a revision that's not the latest, it must be explicitly stated additionally in *traffic*.
- If the deployment does not include template:
  - The traffic weights are updated as declared.

Note: If a revision's traffic weight is explicitly declared in the yaml by name, and it's also the latest revision, the traffic values of the explicit declaration and the *latestRevision* will be summed.

Integrating Custom Domains

Pre-requisites:

1. Every domain needs a domain cert separately.
2. Certificates are for individual container app and bound to container app only
3. SNI Certs needed
4. Ingress must be enabled on container app

Add a custom domain and cert

Note: If using new certificate, you must have an existing SNI domain certificate to upload it.

1. Check if ingress is enabled by navigating to ACA on Azure portal.
   - If not enabled, enabled HTTP Ingress option under ACA settings and give target port and save.
2. Under settings section, select "Add custom domain".
3. In Add custom domain window, enter the domain name e.g. contoso.com and then associate the appropriate record for adding custom domain, apex custom domain needs A record so add the A record associated with the domain or if custom domain is a sub-domain that needs a CNAME record, e.g., mycontainerapp.contoso.com, associate CNAME record if it is a sub-domain.
4. After adding the existing or created DNS records, click validate to start domain validation process.
5. Once validation succeeds, select "Next" and click on "bind certificate + Add" for binding the SSL cert with the container app.
6. In case of a new cert created on local machine, pull out the cert by browsing to the folder where the cert is and upload it or add it as a file.
7. Once adding the SSL cert is done, custom domain added will be listed under the "custom domains".

Managing certificates:

1. Can be done via container app environment or through individual container apps.
2. Navigate to the "Add certificate" to add or renew the certificate.
3. Remove the old cert clicking the trash button.
4. To renew cert, click "renew certificate" to add the new renewed cert to the certs.

Ref: Custom domain names and certificates in Azure Container Apps | Microsoft Docs

In conclusion, we've gone over some of the ACA networking concepts necessary for ensuring your apps adhere to general operational excellence and reliability paradigms. Godspeed.