# Module 5 - Web Apps and Linux

This module will introduce you to App Service on Linux and Web App for Containers. The concepts you learn in this module will enable you to host web apps on Linux.

## Prerequisites

The following prerequisites are necessary before participating in this Boot Camp.

* Basic understanding of Docker containers and images
* Basic understanding of Linux
* Understanding of resource groups and App Service Plans
* Basic understanding of App Service infrastructure and its components like Front End, Controller, etc.
* An Azure subscription

## Goal

After completing this Boot Camp, you will have:

* An understanding of App Service on Linux and Web App for Containers.
* An understanding of managing Web Apps hosted on App Service on Linux and Web App for Containers.

# Hosting Web Apps on Linux

App Service is a fully managed PaaS offering optimized for hosting web applications. Initially we only supported Windows as the OS for hosting Web Apps. We recently added support for Linux. We leverage Docker containers to provide support for hosting Web Apps natively on Linux.

Customers can use either of the following two options to host Web Apps on Linux.

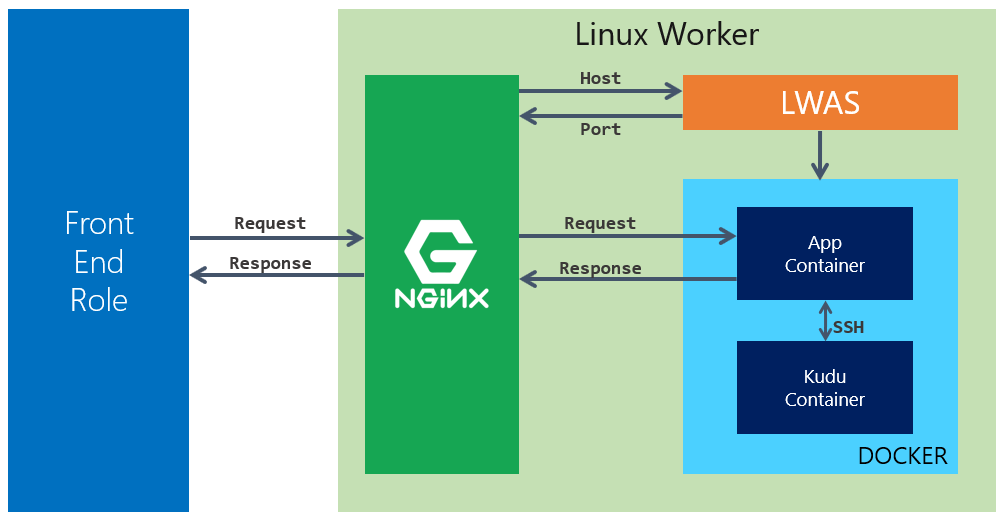
* App Service on Linux
* Web App for Containers

Both options use Docker containers. App Service on Linux uses Docker images maintained by Microsoft and Web App for Containers allows customers to use their own Docker images.

Not all App Service features are available to Linux customers. If a feature is not currently enabled for Linux, it will be grayed out in the portal. As our Linux offerings mature over time, more features will be added.

# Architecture of Linux Stamps

Here is a simple representation of the container architecture on the Linux worker.



Every Linux worker runs Nginx. Nginx is the proxy between the front-end and the Docker containers running on the worker.

LWAS is the Linux version of DWAS. It is responsible for starting and stopping the containers, and it mounts the storage as a CIFS volume to the Docker containers.

## App Service on Linux

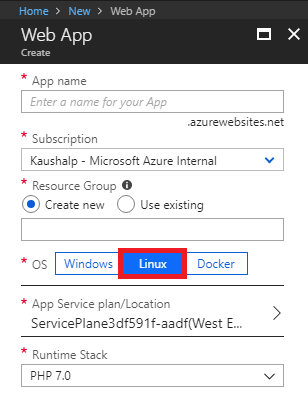
App Service on Linux offers many built-in Docker images for various languages. As of this writing, the following languages and versions are supported.

|  |  |
| --- | --- |
| Language | Supported Versions |
| Node.js | 4.4, 4.5, 4.8, 6.2, 6.6, 6.9, 6.10, 6.11, 8.0, 8.1, 8.2, 8.8, 8.9, 9.4 |
| Java\* | 8.0 |
| PHP | 5.6, 7.0, 7.2 |
| .NET Core | 1.0, 1.1, 2.0 |
| Ruby | 2.3 |
| Apache Tomcat | 8.5, 9.0 |

|  |  |
| --- | --- |
| C:\Users\james\AppData\Local\Temp\SNAGHTML53be59c.PNG | Internally, we refer to the Docker images provided in App Service on Linux as *blessed images*. |

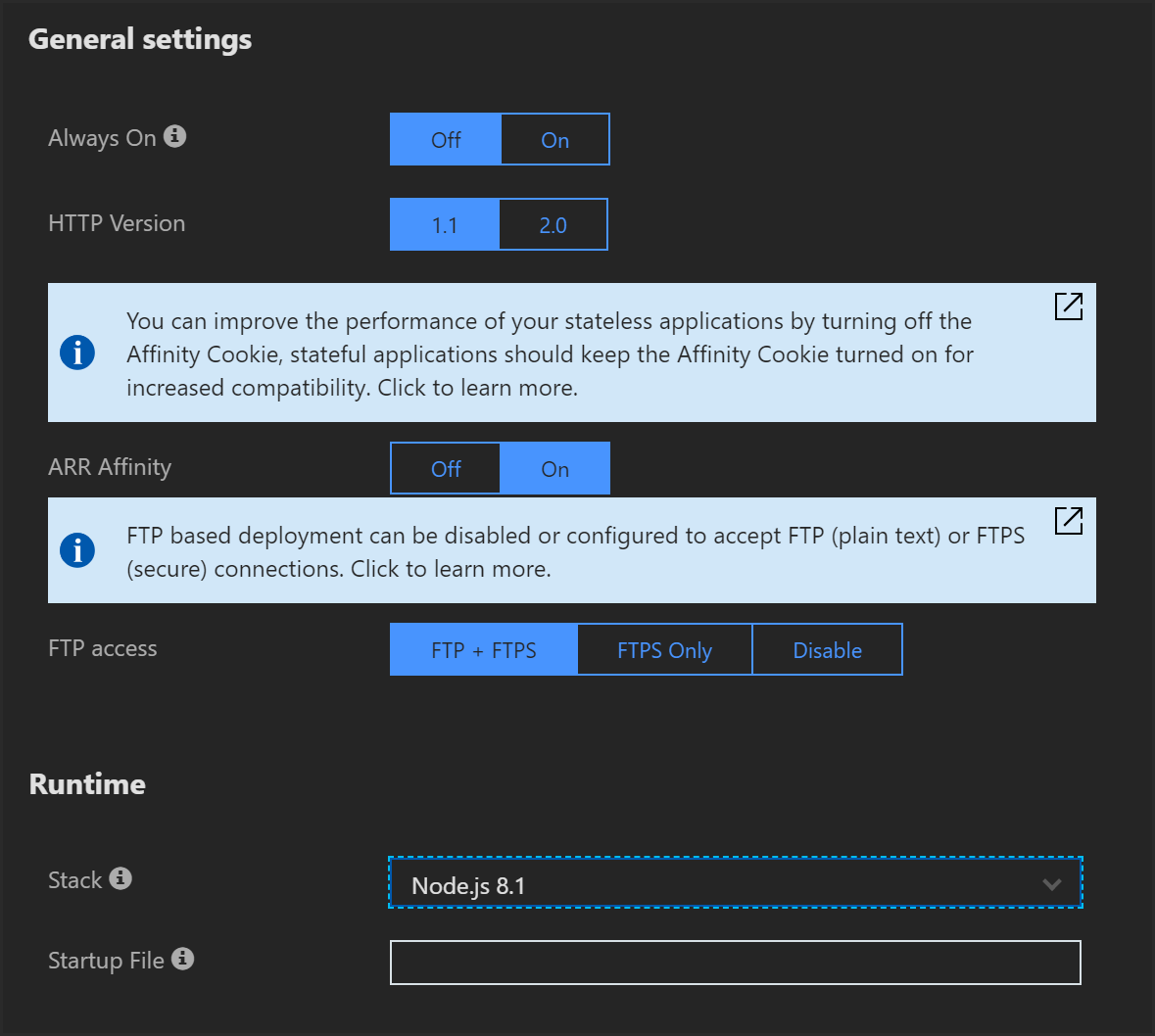
We make our blessed images available to customers in both GitHub and in Docker Hub. You can find our Docker Hub repos at https://hub.docker.com/u/appsvc/. If you view the details of any of the Docker repositories there, you'll see a link to the **Source Repository** in GitHub.

To use our built-in images, set the **OS** to **Linux** as shown below.



After you choose Linux as the OS, you'll need to use the **Runtime Stack** dropdown to select the language and version you want to use for your Web App. The Runtime Stack dropdown correlates directly to the blessed images offered in App Service on Linux.

You can change the runtime stack after creating a Web App using the **Runtime** section in the Application Settings blade as shown in the figure below.



In addition to the **Stack** option here, you can also specify a startup file. The startup file is used for:

* Specifying the PM2 configuration file or the script file for Node.js apps.
* Specifying the compiled DLL file for .NET Core apps. The format is *dotnet <myapp>.dll*.
* Specifying the Ruby script that should initialize your app for Rails apps.

## Web App for Containers

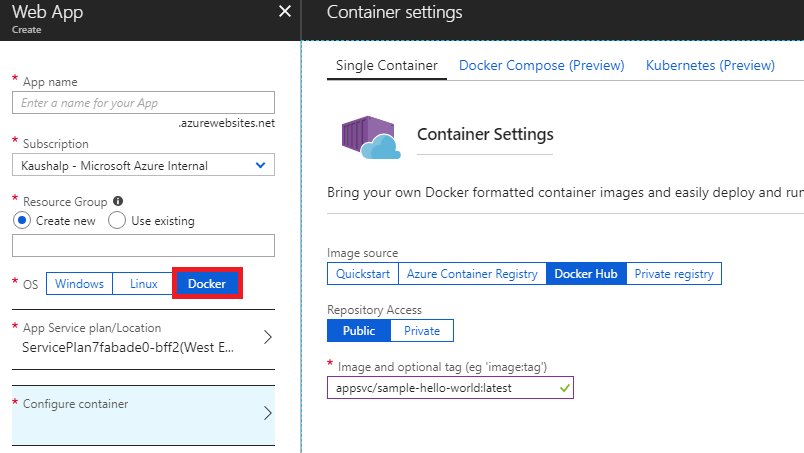
Web App for Containers allows customers to use their own Docker images or Docker images from any Docker registry. This is a great option if the customer's application requires a feature or component that is not available in our built-in images. Customers can host Docker images on Docker Hub, Azure Container Registry, or any other Docker registry.

|  |  |
| --- | --- |
| C:\Users\james\AppData\Local\Temp\SNAGHTML310f382.PNG | For information on building a custom image, see https://docs.microsoft.com/en-us/azure/app-service/containers/tutorial-custom-docker-image. |

To use your own image, select **Docker** as shown below. Under **Configure Container** you will need to provide the **image** and the **tag** (optional).

It's important to realize that not all Docker images are suitable for Web App for Containers. App Service is designed to host Web Apps that listen for HTTP requests and send HTTP responses. Therefore, if a Docker image isn't intended to meet those requirements, it won't work in Web App for Containers.

For example, a Docker image consisting of a daemon that monitors a directory for new files and processes those files will not work in Web App for Containers. In fact, such an image won't even successfully start in Web App for Containers because we require that a container respond to an HTTP ping for it to be considered a successful start.



# Configuring Web Apps in Linux

Basic configuration of settings such as custom domains, SSL certificates, app settings, etc., is the same for Linux sites and Windows sites. However, there are some additional considerations for Linux sites.

## App Service Storage - Web App for Containers Only

Content for a Windows Web App is always mapped to an Azure Storage volume. However, a Linux site can consist of content that is deployed as part of the Docker image. If the Linux Web App doesn't require persistence of any additional files, it's not necessary to have a mount point pointing to an Azure Storage volume. Therefore, by default, Web App for Containers apps do not use Azure Storage for storing any content.

|  |  |
| --- | --- |
| C:\Users\james\AppData\Local\Temp\SNAGHTML53be59c.PNG | We decided not to mount an Azure Storage volume for Linux sites to prevent problems caused by Azure Storage failover. |

In some situations, a customer might want to persist files to Azure Storage. Setting an app setting named WEBSITES\_ENABLE\_APP\_SERVICE\_STORAGE with a value of **True** will cause App Service to mount Azure Storage to /home.

Actually, it's a little more complicated than that. The default for **WEBSITES\_ENABLE\_APP\_SERVICE\_STORAGE** is true, so when you create a new Web App for Containers app, we add the app setting with a value of **false**. If you decide that you want to have persistent storage, you can delete the app setting and it will have the same effect as changing the value to **true**.

|  |  |
| --- | --- |
| C:\Users\james\AppData\Local\Temp\SNAGHTML53be59c.PNG | Keep in mind that when storage persistence is enabled, only the /home directory is persisted. |

## Installing from SSH - All Linux Apps

It's possible to SSH into a Web App running on Linux and install packages. However, it's important to understand that doing so is a bad idea because anything that you install will not persist between restarts of your app.

All our Linux Web App offerings are Docker-based, so a cold start of the app will reinitialize everything to what's included in the Docker container. Anything that's been added since container startup will be gone after a restart.

The only exception to the above is a Web App for Containers app that's persisting the /home directory, but in that case, only the /home directory is persisted.

## Deploying from Visual Studio - All Linux Apps

Just as with Windows sites, you can use Web Deploy to deploy a Linux Web App. However, because Linux apps are Docker container-based apps and because the Web App and Kudu run in separate containers, you need to make sure that your deployment endpoint isn't pointing to the Kudu container.

To ensure that your deployment succeeds with Web Deploy, you can add an app setting with a name of **WEBSITES\_WEBDEPLOY\_USE\_SCM** and set the value to **false**.

## Increasing the Start Time Limit - Web App for Containers Only

When you first browse to a Web App for Containers site, App Service will start the container. Not all Docker containers start within the same amount of time. A simple container might start within a few seconds, but if a container is complex, it can take several minutes for it to start.

We will wait 230 seconds for a container to start before we decide that it failed to start. However, if your container takes longer than 230 seconds to start, you can increase the time period that we wait up to a limit of 1,800 seconds. To do so, add an app setting named **WEBSITES\_CONTAINER\_START\_TIME\_LIMIT** and set the value to the number of seconds you want App Service to wait for the container to start. (Don't include commas. If you want us to wait 1,800 seconds, set the value to **1800**.)

|  |  |
| --- | --- |
| C:\Users\james\AppData\Local\Temp\SNAGHTML53be59c.PNG | A container startup is considered successful only after the Docker container starts and the container responds to an HTTP ping. |

## Exposing Ports - Web App for Containers Only

App Service will use Docker's *inspect* command to try and determine what port your Docker image needs to expose the outside world. However, in some cases, we can't determine the port. If that's the case, your Docker container may fail to start because it won't respond to our HTTP pings.

You can add an app setting to tell App Service what port to expose in your container. Add the **WEBSITES\_PORT** app setting and set the value to the port you want to expose.

|  |  |
| --- | --- |
| C:\Users\james\AppData\Local\Temp\SNAGHTML53be59c.PNG | This app setting used to be called **PORT** instead of **WEBSITES\_PORT**. As of this writing, we will honor either of these app settings, but at some point, we will deprecate the PORT app setting. |

## SSL and Linux

This guidance applies to all Linux Web Apps, but it's more likely to be appropriate for a Web App for Containers customer. Just as with our Windows offering, SSL to a Linux Web App is terminated on the front-end. That means that all requests to the Web App's worker are going to be non-SSL. For that reason, customers should not include SSL support in their app.

Linux customers are often used to configuring things using config files, so they may not realize that doing something like configuring Apache to recognize SSL requests isn't going to work.

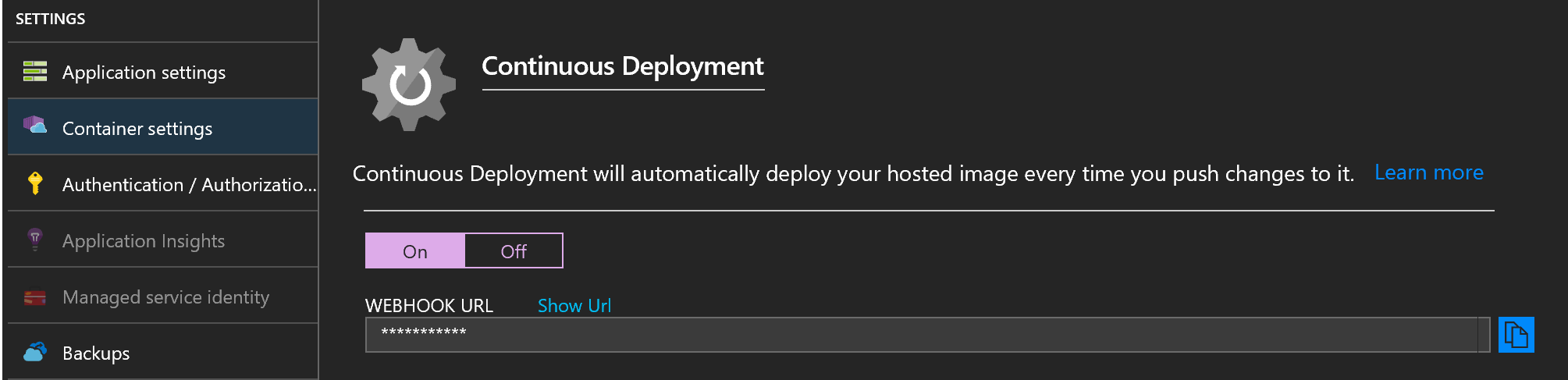
## Environment Variables

If you need to set an environment variable inside of your Docker container, you can use an app setting. When the container is started, App Service will inject the app setting into the container as an environment variable.

|  |  |
| --- | --- |
| C:\Users\james\AppData\Local\Temp\SNAGHTML53be59c.PNG | You won't see your environment variables in the Bash console in Kudu. The Bash console runs in a different context (the Kudu container) than your app. If you want to see your environment variables, use SSH to access your app. |

# Continuous Deployment

Customers can enable continuous deployment so that deployment happens automatically when a change is pushed to an image. To enable continuous deployment, turn it on in the portal and ensure that your registry is configured with the webhook URL shown in the Azure portal.



When a new Docker image is pushed to your registry, the registry will use a webhook to call the webhook URL. This hits an SCM endpoint at /docker/hook in App Service and tells App Service to do a pull on the new image. At that point, App Service will perform an overlapped recycle of the Web App once the new container is running.

|  |  |
| --- | --- |
| C:\Users\james\AppData\Local\Temp\SNAGHTML53be59c.PNG | The image:tag combination must remain the same in order to continuous integration to work. If you change the tag on an image, you’ll need to change the tag in the Azure portal in order for the new image to be pulled. |

# Multi-Container Deployments

Customers can also use either Docker Compose or Kubernetes to deploy a multi-container application in Web App for Containers. To deploy a multi-container, you must create a YAML file that defines the deployment using either Docker Compose or Kubernetes.

Multi-container deployments use the same Docker images as a single-container deployment. A configuration file in YAML format is used to define the Docker images that should be included in the deployment and various options for building the deployment.

In a multi-container deployment, one container is considered the *web container*, the container that represents the website that users see when they browse to the Web App. We use the following logic to identify the web container.

1. If your YAML file has only one container in it, we'll use that container as the web container.
2. If you have set the WEBSITES\_WEB\_CONTAINER\_NAME app setting, we will use that as the web container.
3. We will pick the first container in your YAML file that exposes either port 80 or port 8080.
4. If we make it this far and we haven't picked the web container, we'll use the first container in the list of containers in your YAML file.

There are a couple of requirements that must be met for a multi-container deployment to succeed; each Docker image specified in the deployment must use the same Docker registry, and all containers must start successfully or the startup of the Web App will fail.

Let’s have a look at both Docker Compose and Kubernetes and how you can use them in App Service.

## Docker Compose

In Docker Compose, the YAML configuration file is referred to as a *compose file*. The compose file defines one or more *services*, and each service consists of a Docker image and various settings. Consider the following sample compose file from our online documentation.

|  |
| --- |
| version: '3.3'  services:  db:  image: mysql:5.7  volumes:  - db\_data:/var/lib/mysql  restart: always  environment:  MYSQL\_ROOT\_PASSWORD: somewordpress  MYSQL\_DATABASE: wordpress  MYSQL\_USER: wordpress  MYSQL\_PASSWORD: wordpress  wordpress:  image: wordpress:latest  ports:  - "8000:80"  restart: always  environment:  WORDPRESS\_DB\_HOST: db:3306  WORDPRESS\_DB\_USER: wordpress  WORDPRESS\_DB\_PASSWORD: wordpress  volumes:  db\_data: |

|  |  |
| --- | --- |
| C:\Users\james\AppData\Local\Temp\SNAGHTML53be59c.PNG | Indentation is important in your YAML files. The number of spaces you indent doesn’t matter, but it must be at least one space, and you should always use the same number of spaces for the same indentation level. Do not use tabs. Use only spaces. |

This compose file defines two services; db and wordpress. The db service uses the mysql:5.7 Docker image, and the wordpress service uses the wordpress:latest Docker image. (Both of these Docker images will be pulled from the Docker Hub registry.) Notice also that the wordpress service maps port 80, so this will be the web container in the deployment.

|  |  |
| --- | --- |
| C:\Users\james\AppData\Local\Temp\SNAGHTML53be59c.PNG | We don’t support all Docker Compose configuration options in App Service. To see what we do and don’t support, see https://docs.microsoft.com/en-us/azure/app-service/containers/tutorial-multi-container-app#docker-compose-configuration-options. |

## Kubernetes (Kube or K8s)

Kubernetes (sometimes referred to as Kube or K8s) is an open-source system that is designed to manage multiple containers. Kubernetes orchestrates container startup and manages other aspects of containers. In App Service, we don’t use many of the features of Kubernetes.

It’s important to understand that Kubernetes is not an alternative to Docker. Kubernetes requires a runtime to host the containers that it manages, and in App Service, that runtime is Docker.

|  |  |
| --- | --- |
| C:\Users\james\AppData\Local\Temp\SNAGHTML53be59c.PNG | Docker has its own orchestration feature called *Docker Swarm*. We don’t currently support Docker Swarm in App Service. |

In App Service, we support the creation of a Kubernetes *pod*, a group of containers that share storage and a network. To define a Kubernetes pod, you use a Kubernetes config file. Kubernetes supports both YAML and JSON for the config file, but in App Service, we require that you use YAML.

Here’s a sample Kubernetes config file. Note that the *kind* key must have a value of *Pod* as that’s the only Kubernetes object we support in App Service.

|  |
| --- |
| apiVersion: v1  kind: Pod  metadata:  name: wordpress  spec:  containers:  - image: redis:3-alpine  name: redis  - image: microsoft/multicontainerwordpress  name: wordpress  ports:  - containerPort: 80  volumeMounts:  - name: appservice-storage  mountPath: /var/www/html  subPath: /site/wwwroot  volumes:  - name: appservice-storage  hostConfig:  path: ${WEBAPP\_STORAGE\_HOME} |

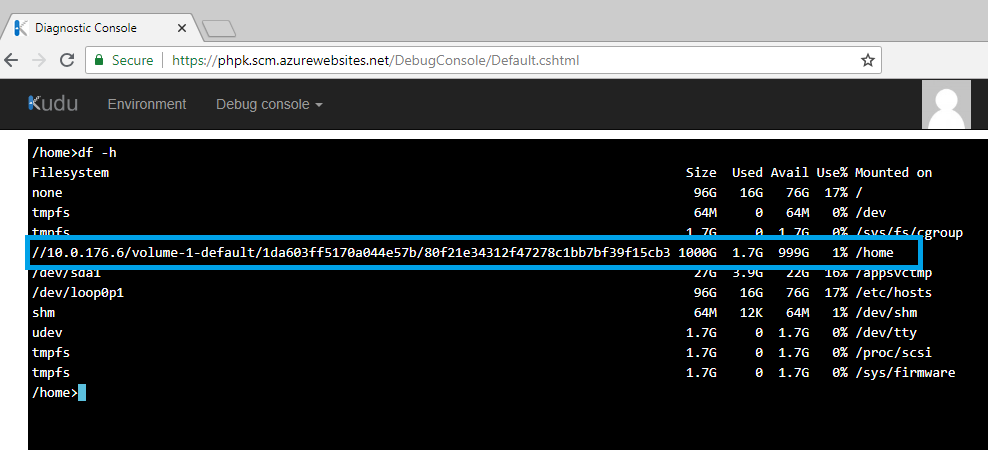
Note that **${WEBAPP\_STORAGE\_HOME}** is used when you need to refer to the shared storage location made available when App Service Storage is enabled.

We don’t support all Kubernetes options in the config file. For the official list of what we support, see https://docs.microsoft.com/en-us/azure/app-service/containers/tutorial-multi-container-app#use-a-kubernetes-configuration-optional.

# Kudu

On Linux workers, Kudu runs inside a separate container from the Web App. Kudu can be accessed by browsing to https://<sitename>.scm.azurewebsites.net. In Kudu, you can click on Debug Console to access two different command line consoles; Bash and SSH.

When you access the Bash console, you are accessing the Kudu container and not the Web App’s container. However, if you are using App Service on Linux or if you are using Web App for Containers with App Service Storage enabled, you will see the /home folder for your Web App as shown below.



If you click SSH on the Debug Console menu, you will connect to the Web App’s container via SSH. For this to successfully connect, your Web App must be running. If the Web App is stopped, SSH will not be able to connect.

App Service on Linux images are configured for SSH for you. If you use Web App for Containers, you must ensure that your image is build so that it supports SSH. For information on how to do that, see <https://aka.ms/ThingsYouShouldKnow/Linux#CustomSSH>.

## SSH with Other Clients

Many users prefer to use their own SSH client to access their app. Some people like to use an app like Putty. Still others prefer to open a console on their computer and SSH directly to their app. Both of these options are possible by opening a TCP tunnel to your Web App.

You can find all of the information necessary to SSH using your favorite method by going to <https://aka.ms/ThingsYouShouldKnow/AppServiceSSH>.

|  |  |
| --- | --- |
| C:\Users\james\AppData\Local\Temp\SNAGHTML53be59c.PNG | Opening a TCP tunnel also allows you to use SFTP and to remotely debug your Node.js apps running in App Service. |