Creating and Deploying Tomcat based web app docker image into Azure App Service

Version 0.0

Author : surendra@nichesolv.com

Introduction

This document capture the steps for creating tomcat (9.0 with JDK 8.0) sample java webapp (Spring Pet Clinic WAR https://github.com/spring-petclinic/spring-framework-petclinic) docker image and deploying it to Azure Container Registry (ACR) and configuring Azure App Service to run pull this image from ACR and run it a as Web App.

References

1. https://docs.microsoft.com/en-us/azure/app-service/tutorial-custom-container?pivots=container-linux
2. https://github.com/Azure/tomcat-container-quickstart
3. https://docs.microsoft.com/en-us/azure/app-service/configure-custom-container?pivots=container-linux#enable-ssh

Prerequisites

- Azure Account with Active Subscription - for ACR, AppService and other resources
- Docker (e.g.Installed on Ubuntu VM) - for building images
- Azure CLI (e.g. Azure CLI on Ubuntu) - for executing steps from CLI

Clone and download quickstart repo

```
git clone https://github.com/Azure/tomcat-container-quickstart.git
cd tomcat-container-quickstart
```

(Optional) Examine the Docker file

```
FROM mcr.microsoft.com/java/jdk:8-zulu-alpine
ARG APP_FILE=ROOT.war
ARG TOMCAT_VERSION=9.0.38
ARG SERVER_XML=server.xml
ARG LOGGING_PROPERTIES=logging.properties
ARG CATALINA_PROPERTIES=catalina.properties
ARG TOMCAT_MAJOR=9
```

```
# As provided here, only the access log gets written to this location.
# Mount to a persistent volume to preserve access logs.
# Modify this value to specify a different log directory.
# e.g. /home/logs in Azure App Service
ENV LOG_ROOT=/tomcat_logs

ENV PATH /usr/local/tomcat/bin:$PATH

# Make Java 8 obey container resource limits, improve performance
ENV JAVA_OPTS "-XX:+UnlockExperimentalVMOptions
-XX:+UseCGroupMemoryLimitForHeap -XX:+UseG1GC -Djava.awt.headless=true"

# Cleanup & Install
RUN apk add --update openssh-server bash openrc \
        && rm -rf /var/cache/apk/* \
        # Remove unnecessary services
        && rm -f /etc/init.d/hwdrivers \
                 /etc/init.d/hwclock \
                 /etc/init.d/mtab \
                 /etc/init.d/bootmisc \
                 /etc/init.d/modules \
                 /etc/init.d/modules-load \
                 /etc/init.d/modloop \
        # Install Tomcat
        && wget -O /tmp/apache-tomcat-${TOMCAT_VERSION}.tar.gz
https://archive.apache.org/dist/tomcat/tomcat-$TOMCAT_MAJOR/v${TOMCAT_VERSI
ON}/bin/apache-tomcat-${TOMCAT_VERSION}.tar.gz --no-verbose \
        && mkdir /usr/local/tomcat \
        && tar xvzf /tmp/apache-tomcat-${TOMCAT_VERSION}.tar.gz -C
/usr/local/tomcat --strip-components 1 \
        && rm -f /tmp/apache-tomcat-${TOMCAT_VERSION}.tar.gz \
         # Remove the sample webapps provided by Tomcat
        && rm -rf /usr/local/tomcat/webapps/*

COPY $APP_FILE /usr/local/tomcat/webapps/ROOT.war

RUN mkdir /usr/local/tomcat/webapps/ROOT \
    && cd /usr/local/tomcat/webapps/ROOT \
    && unzip ../ROOT.war \
    && rm -f ../ROOT.war

COPY ${SERVER_XML} /usr/local/tomcat/conf/server.xml
COPY ${LOGGING_PROPERTIES} /usr/local/tomcat/conf/logging.properties
COPY ${CATALINA_PROPERTIES} /usr/local/tomcat/conf/catalina.properties

# Copy the startup script
COPY startup.sh /startup.sh
RUN chmod a+x /startup.sh

EXPOSE 8080

CMD ["/startup.sh"]
```

Build and test the image locally

Use the following command to build the image

```
docker build . -t pet-store-sample
```

Use the following command to run the image locally

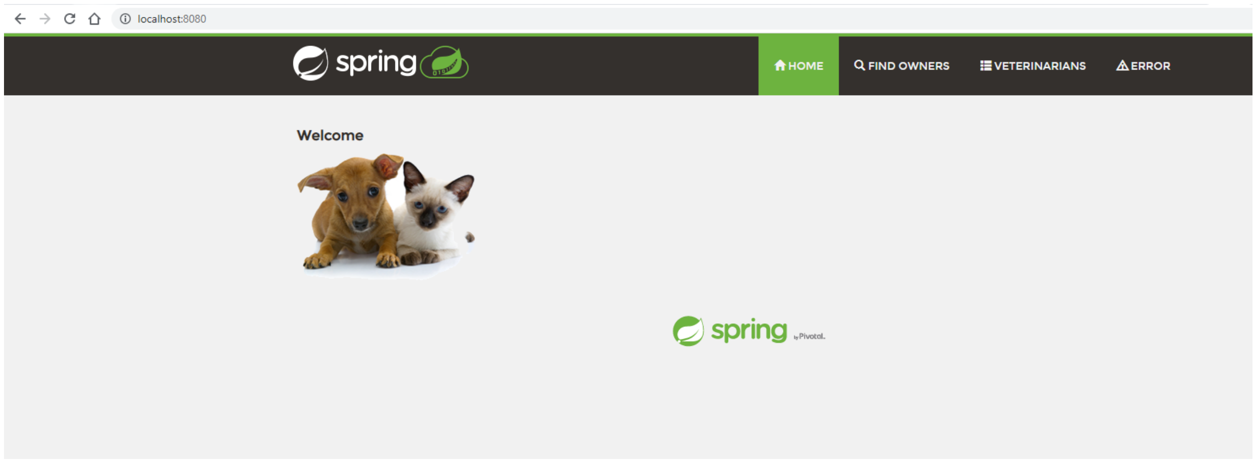
```
docker run -it -p 8080:8080 pet-store-sample
```

You will see tomcat server logs in the console. (You can use Ctrl + C to terminate the server)

Launch another console window and use curl command to verify that server is running

```
curl http://localhost:8080
```

If you are able to launch browser you will see the home page



Login to Azure

In this section and the following sections, we prepare resources in Azure to push the image and then deploy a container to Azure App Service. We start by creating a resource group in which you want to collect all the resources. First login to your Azure Account and Select the right subscription

```
az login
```

Use az account show to confirm you are connected to right subscription, if not use
az account set --subscription 'subscription name or id' to select the right subscription.

Create a resource group

Run the az group create command to create a resource group:

```
az group create --name docker-sample --location eastus
```

We can change the --location value to specify the required region.

Push the image to Azure Container Registry

In this section, we push the image to Azure Container Registry from which App Service can pull and deploy it.

1. Run the az acr create command to create an Azure Container Registry:

```
az acr create --name <registry-name> --resource-group myResourceGroup --sku
Basic --admin-enabled true
```

Replace <registry-name> with a suitable name for your registry. The name must contain only letters, numbers, and must be unique across all of Azure.
For e.g.

```
az acr create --name nichesuren --resource-group docker-sample --sku Basic
--admin-enabled true
```

2. Run the az acr show command to retrieve credentials for the registry:

```
az acr credential show --resource-group myResourceGroup --name
<registry-name>
```

The JSON output of this command provides two passwords along with the registry's user name.
For e.g.

```
az acr credential show --resource-group docker-sample  --name nichesuren
```

Will produce

```
{
  "passwords": [
    {
      "name": "password",
      "value": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
    },
    {
      "name": "password2",
      "value": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
    }
  ],
  "username": "nichesuren"
}
```

3. Use the docker login command to sign in to the container registry:

```
docker login <registry-name>.azurecr.io --username <registry-username>
```

For e.g

```
docker login nichesuren.azurecr.io --username nichesuren
```

4. When the login is successful, we tag the local Docker image to the registry:

```
docker tag <custom-image>
<registry-name>.azurecr.io/appsvc-tutorial-custom-image:latest
```

For e.g

```
docker tag pet-store-sample nichesuren.azurecr.io/pet-store-sample:latest
```

5. Use the docker push command to push the image to the registry:

```
docker push <registry-name>.azurecr.io/<custom-image>:latest
```

For e.g.

```
docker push nichesuren.azurecr.io/pet-store-sample:latest
```

Uploading the image the first time will take a few minutes because it includes the base image. Subsequent uploads are typically faster.

6. Use the az acr repository list command to verify that the push was successful:

```
az acr repository list -n <registry-name>
```

For e.g.

```
az acr repository list -n nichesuren
```

Will produce output like this

```
[
  "pet-store-sample"
]
```

Configure App Service to deploy the image from the registry

To deploy a container to Azure App Service, we first create a web app on App Service, then connect the web app to the container registry. When the web app starts, App Service automatically pulls the image from the registry.

1. Create an App Service plan using the az appservice plan create command:

```
az appservice plan create --name myAppServicePlan --resource-group
myResourceGroup --is-linux
```

An App Service plan corresponds to the virtual machine that hosts the web app. By default, the previous command uses an inexpensive B1 pricing tier that is free for the first month. We can control the tier with the --sku parameter.

For e.g

```
az appservice plan create --name dockerAppServicePlan --resource-group
docker-sample --is-linux
```

2. Create the web app with the az webpp create command:

```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan
--name <app-name> --deployment-container-image-name
<registry-name>.azurecr.io/appsvc-tutorial-custom-image:latest
```

Replace <app-name> with a name for the web app, which must be unique across all of Azure. Also replace <registry-name> with the name of your registry from the previous section.

For e.g.

```
az webapp create --resource-group docker-sample --plan dockerAppServicePlan
--name surenPetSample --deployment-container-image-name
nichesuren.azurecr.io/pet-store-sample:latest
```

3. Use az webapp config appsettings set to set the WEBSITES_PORT environment variable as expected by the app code:

```
az webapp config appsettings set --resource-group myResourceGroup --name
<app-name> --settings WEBSITES_PORT=8000
```

For e.g

```
az webapp config appsettings set --resource-group docker-sample --name
surenPetSample --settings WEBSITES_PORT=8080
```

4. Enable the system-assigned managed identity for the web app by using the az webapp identity assign command:

```
az webapp identity assign --resource-group myResourceGroup --name
<app-name> --query principalId --output tsv
```

Replace <app-name> with the name we used in the previous step. The output of the command (filtered by the --query and --output arguments) is the service principal of the assigned identity, which we will use shortly.

Managed identity allows us to grant permissions to the web app to access other Azure resources without needing any specific credentials.

For e.g

```
az webapp identity assign --resource-group docker-sample --name
surenPetSample --query principalId --output tsv
```

5. Retrieve the subscription ID with the az account show command, which we need in the next step:

```
az account show --query id --output tsv
```

6. Grant the managed identity permission to access the container registry:

```
az role assignment create --assignee <principal-id> --scope
/subscriptions/<subscription-id>/resourceGroups/myResourceGroup/providers/M
icrosoft.ContainerRegistry/registries/<registry-name> --role "AcrPull"
```

Replace the following values:

<principal-id> with the service principal ID from the az webapp identity assign command.
<registry-name> with the name of our container registry.
<subscription-id> with the subscription ID retrieved from the az account show command.
For e.g.

```
az role assignment create --assignee xxxxxx-352d-4706-xxxx-71f0ab0d0654
--scope
/subscriptions/xxxxxx-b478-40d1-xxxx-bb52812d8b3e/resourceGroups/docker-sam
ple/providers/
Microsoft.ContainerRegistry/registries/nichesuren --role "AcrPull"
```

7. Configure your app to use the managed identity to pull from Azure Container Registry.

```
az resource update --ids
/subscriptions/<subscription-id>/resourceGroups/myResourceGroup/providers/M
icrosoft.Web/sites/<app-name>/config/web --set
properties.acrUseManagedIdentityCreds=True
```

Replace the following values:

<subscription-id> with the subscription ID retrieved from the az account show command.
<app-name> with the name of our web app.

For e.g

```
az resource update --ids
/subscriptions/xxxxxx-b478-40d1-xxxx-bb52812d8b3e/resourceGroups/docker-sam
ple/providers/Microsoft.Web/sites/surenPetSample/config/web --set prope
rties.acrUseManagedIdentityCreds=True
```

Deploy the image and test the app

1. Use the az webapp config container set command to specify the container registry and the image to deploy for the web app:

```
az webapp config container set --name <app-name> --resource-group
myResourceGroup --docker-custom-image-name
<registry-name>.azurecr.io/appsvc-tutorial-custom-image:latest
--docker-registry-server-url https://<registry-name>.azurecr.io
```

Replace <app-name> with the name of the web app and replace <registry-name> in two places with the name of the registry.

When using a registry other than Docker Hub (as this example shows),
--docker-registry-server-url must be formatted as https:// followed by the fully qualified domain name of the registry.
The message, "No credential was provided to access Azure Container Registry. Trying to look up..." indicates that Azure is using the app's managed identity to authenticate with the container registry rather than asking for a username and password.
If you encounter the error, "AttributeError: 'NoneType' object has no attribute 'reserved'", ensure that your <app-name> is correct.

For e.g.

```
az webapp config container set --name surenPetSample
--resource-group docker-sample --docker-custom-image-name
nichesuren.azurecr.io/pet-store-sample:latest --docker-registry-server-url
https://nichesuren.azurecr.io
```

We can retrieve the web app's container settings at any time with the command

```
 az webapp config container show --name <app-name> --resource-group
myResourceGroup
```

For e.g.

```
az webapp config container show --name surenPetSample --resource-group
docker-sample
```

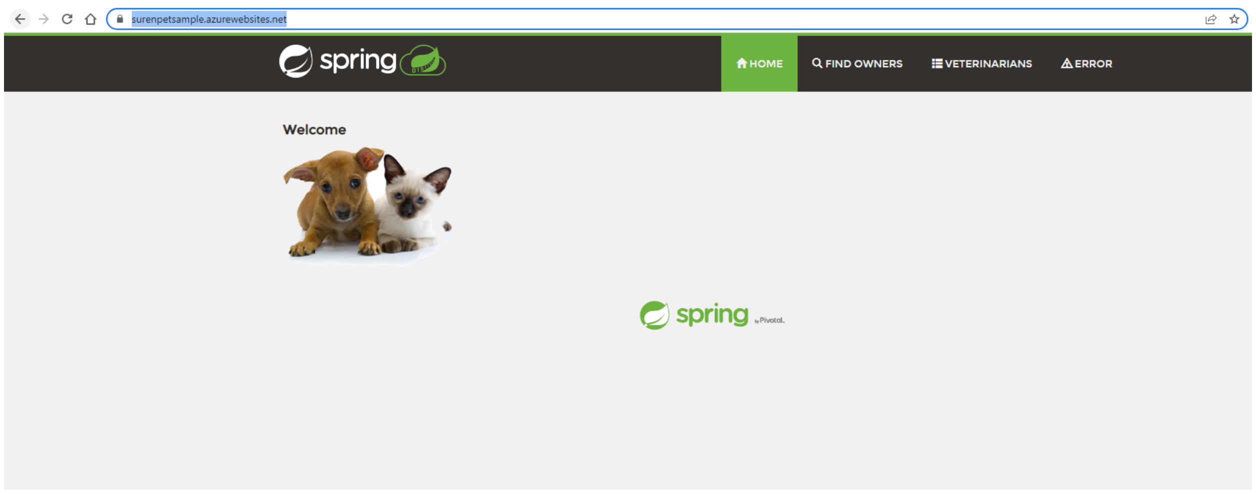
Will display

```
[
  {
    "name": "WEBSITES_ENABLE_APP_SERVICE_STORAGE",
    "slotSetting": false,
    "value": "false"
  },
  {
    "name": "DOCKER_REGISTRY_SERVER_URL",
    "slotSetting": false,
    "value": "https://nichesuren.azurecr.io"
  },
  {
    "name": "DOCKER_REGISTRY_SERVER_USERNAME",
    "slotSetting": false,
    "value": "nichesuren"
  },
  {
    "name": "DOCKER_REGISTRY_SERVER_PASSWORD",
    "slotSetting": false,
    "value": null
  },
  {
    "name": "DOCKER_CUSTOM_IMAGE_NAME",
    "value": "DOCKER|nichesuren.azurecr.io/pet-store-sample:latest"
  }
]
```

2. When the az webapp config container set command completes, the web app should be running in the container on App Service.

   To test the app, browse to https://<app-name>.azurewebsites.net, replacing <app-name> with the name of your web app. On first access, it might take some time for the app to respond because App Service must pull the entire image from the registry. If the browser times out, just refresh the page. Once the initial image is pulled, subsequent tests will run much faster.

```
https://surenpetsample.azurewebsites.net/
```



Access diagnostic logs

1. Turn on container logging:

```
az webapp log config --name <app-name> --resource-group myResourceGroup
--docker-container-logging filesystem
```

For e.g

```
az webapp log config --name surenPetSample --resource-group docker-sample
--docker-container-logging filesystem
```

2. Enable the log stream:

```
az webapp log tail --name <app-name> --resource-group myResourceGroup
```

For e.g

```
az webapp log tail --name surenPetSample --resource-group
docker-sample
```

Wait for some time to see the logs

You can also inspect the log files from the browser at `https://<app-name>.scm.azurewebsites.net/api/logs/docker.`

3. To stop log streaming at any time, type Ctrl+C.

Configure continuous deployment

Our App Service app now can pull the container image securely from our private container registry. However, it doesn't know when that image is updated in our registry. Each time we push the updated image to the registry, we must manually trigger an image pull by restarting the App Service app. In this step, we enable CI/CD, so that the App Service gets notified of a new image and triggers a pull automatically.

1. Enable CI/CD in App Service.

```
az webapp deployment container config --enable-cd true --name <app-name>
--resource-group myResourceGroup --query CI_CD_URL --output tsv
```

CI_CD_URL is a URL that App Service generates for us. Our registry should use this URL to notify App Service that an image push occurred. It doesn't actually create the webhook for us.

For e.g

```
az webapp deployment container config --enable-cd tru
e --name surenPetSample --resource-group docker-sample --query CI_CD_URL
--output tsv
```

Would produce

```
https://$surenPetSample:xxxxxxoPkX09Bmgsk1pG488w1wS3aFrWNgwujz1KyvJ8PaMT8Df
rZFhD3e5w@surenpetsample.scm.azurewebsites.net/docker/hook
```

2. Create a webhook in our container registry using the CI_CD_URL we got from the last step.

```
az acr webhook create --name appserviceCD --registry <registry-name> --uri
'<ci-cd-url>' --actions push --scope appsvc-tutorial-custom-image:latest
```

For e.g

```
az acr webhook create --name appserviceCD --registry
nichesuren --uri
'https://$surenPetSample:xxxxxxPkX09Bmgsk1pG488w1wS3aFrWNgwujz1KyvJ8PaMT8Df
rZFhD3e5w@surenpetsample.scm.azurewebsites.net/docker/hook' --actions push
--scope pet-store-sample:latest
```

3. To test if our webhook is configured properly, ping the webhook and see if we get a 200 OK response.

```
eventId=$(az acr webhook ping --name appserviceCD --registry
<registry-name> --query id --output tsv)
```

```
az acr webhook list-events --name appserviceCD --registry <registry-name>
--query "[?id=='$eventId'].eventResponseMessage"
```

For e.g

```
eventId=$(az acr webhook ping --name appserviceCD --registry nichesuren
--query id --output tsv)
```

```
az acr webhook list-events --name appserviceCD --registry nichesuren
--query "[?id=='$eventId'].eventResponseMessage"
```

Will produce

```
[
  {
    "content":
"{\"OperationId\":\"04b39378-9e16-41dc-bfb3-3524ae61eb78\",\"TrackingUrl\":
\"https://surenpetsample.scm.azurewebsites.net/api/logstream?filter=op:04b3
9378-9e16-41dc-bfb3-3524ae61eb78,volatile:false\"}",
    "headers": {
      "Content-Length": "191",
      "Date": "Mon, 11 Apr 2022 14:17:14 GMT",
      "Server": "Microsoft-IIS/10.0"
    },
    "reasonPhrase": "OK",
    "statusCode": "200",
    "version": "1.1"
  }
]
```

Modify the app code and redeploy

1. Modify the web application and generate a new WAR file
2. Rebuild the docker image

```
docker build . -t pet-store-sample
```

3. Update the image's tag to latest:

```
docker tag <custom-image> <registry-name>.azurecr.io/<custom-image>:latest
```

For e.g

```
docker tag pet-store-sample nichesuren.azurecr.io/pet-store-sample:latest
```

4. Use the docker push command to push the image to the registry:

```
docker push <registry-name>.azurecr.io/<custom-image>:latest
```

For e.g.

```
docker push nichesuren.azurecr.io/pet-store-sample:latest
```

5. When the image push is complete, the webhook notifies the App Service about the push, and App Service tries to pull in the updated image. Wait a few minutes, and then verify that the update has been deployed by browsing to https://<app-name>.azurewebsites.net.

Enable SSH

SSH enables secure communication between a container and a client. In order for a custom container to support SSH, we must add it into your Docker image itself.

Configure the container for SSH

1. Add an sshd_config file to the repository, like the following example.

```
Port                 2222
ListenAddress              0.0.0.0
LoginGraceTime             180
X11Forwarding              yes
Ciphers aes128-cbc,3des-cbc,aes256-cbc,aes128-ctr,aes192-ctr,aes256-ctr
MACs hmac-sha1,hmac-sha1-96
StrictModes                yes
SyslogFacility             DAEMON
PasswordAuthentication  yes
PermitEmptyPasswords    no
PermitRootLogin   yes
Subsystem sftp internal-sftp
```

2. Add an ssh_setup script file to create the SSH keys using ssh-keygen to the repository.

```
#!/bin/sh

ssh-keygen -A

#prepare run dir
if [ ! -d "/var/run/sshd" ]; then
    mkdir -p /var/run/sshd
fi
```

3. In your Dockerfile, add the following commands:

```
# Install OpenSSH and set the password for root to "Docker!". In this
example, "apk add" is the install instruction for an Alpine Linux-based
image.
RUN apk add openssh \
     && echo "root:Docker!" | chpasswd

# Copy the sshd_config file to the /etc/ssh/ directory
COPY sshd_config /etc/ssh/

# Copy and configure the ssh_setup file
RUN mkdir -p /tmp
COPY ssh_setup.sh /tmp
RUN chmod +x /tmp/ssh_setup.sh \
    && (sleep 1;/tmp/ssh_setup.sh 2>&1 > /dev/null)

# Open port 2222 for SSH access
EXPOSE 80 2222
```

4. In the start-up script for your container, start the SSH server.

```
#!/bin/sh

# Create a log directory for each hostname in case a shared file system is
used among multiple hosts or containers.
# As provided in the default configuration, only the access log gets
written to this directory.
LOG_DIR="${LOG_ROOT}/$(hostname)"
mkdir -p "${LOG_DIR}"
export JAVA_OPTS="${JAVA_OPTS} -Dlog.base=${LOG_DIR}"
/usr/sbin/sshd
/usr/local/tomcat/bin/catalina.sh run
```

Open SSH connection to container

Browse to https://<app-name>.scm.azurewebsites.net/webssh/host and sign in with your Azure account. Replace <app-name> with the name of your web app.

When you sign in, you're redirected to an informational page for the web app. Select SSH at the top of the page to open the shell and use commands.

For example, you can examine the processes running within it using the top command.

Clean up resources

The resources we created in the above steps might incur ongoing costs. To clean up the resources, we only need to delete the resource group that contains them:

```
az group delete --name myResourceGroup
```