# Transact-BOHA Web Application Framework

## Introduction

This document will provide you with the instructions to implement Transact BOHA web application.

The Transact BOHA application consists of the components listed below

* SQL Server Cloud Database
* Azure AD B2C
* ASP.NET Core Web API
* Angular 13 Web Application

In addition to these components, we will build a Kubernetes controller, an image repository, and an operating environment (server farm). Operations management, redundancy, scaling up and down and automatic restart are managed by Kubernetes and native Azure features. The application code for the Web Application and the Web API is located at the git address specified below:

<https://symphonize.git.beanstalkapp.com/transact.git>

## Prerequisites

The required components are a Windows and/or Mac desktop and an Azure account.

The document assumes that your desktop machine is already on the latest version of the following components:

1. Node.js
2. Angular CLI
3. Docker Desktop
4. NPM and Angular packages
5. Azure CLI

Node.js: <https://nodejs.org/en/download/>

Angular CLI: npm install -g @angular/cli

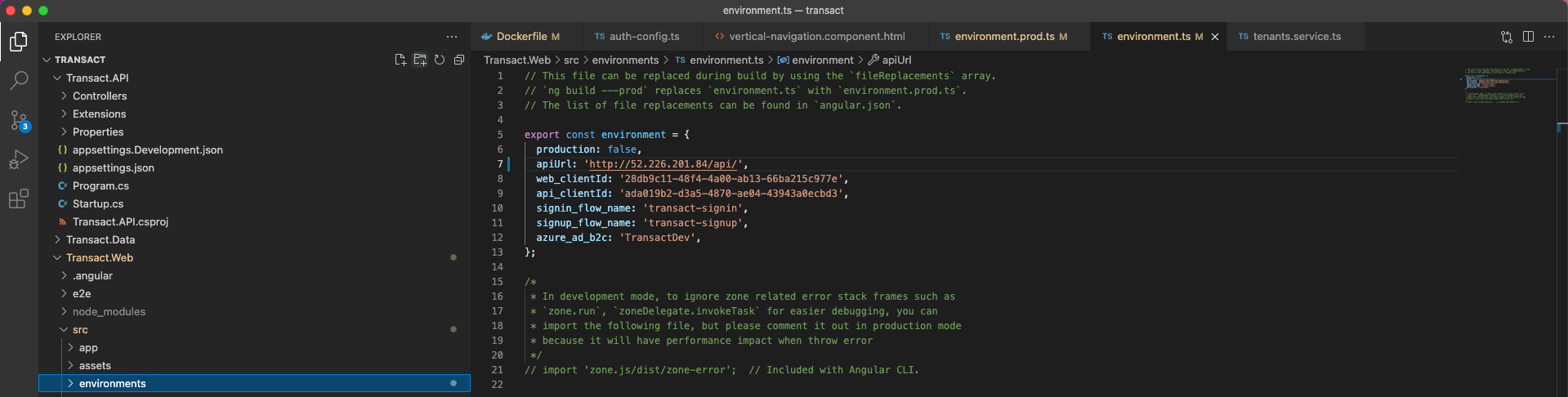
Docker Desktop: <https://docs.docker.com/desktop/mac/install/>

<https://docs.docker.com/desktop/windows/install/>

Azure CLI: <https://docs.microsoft.com/en-us/cli/azure/install-azure-cli>

In addition to the above components, the document assumes that you were able to succesfully install the Transact.API webapi to Azure Kubenetes Service. This will give you access to the ip address that you need to use in the environment.ts and environment.prod.ts files in the folder Transact.Web/src

Please check the screenshot below.



## Step 1 – One Time Infrastructure Build

This step has been implemented as part of the Transact.API installation. So, we don’t need to repeat this step one more time for the Transact.Web app

## Step 2 – WebApp Deployment

WebApp deployment consists of using an application repository (git) to construct a Docker image (application + operating system), testing the build then moving the image to Azure where rollout and operations will be managed by Kubernetes.

1. Create Application Directory

On you desktop machine, cd to the desired location for the application code.

|  |
| --- |
| % mkdir transact  % cd transact |
| transact % |

1. Download the application project from the git repo

The clone/download is the repo with our deliverables.

|  |
| --- |
| % git clone https://symphonize.git.beanstalkapp.com/transact.git |
| transact % git clone https://symphonize.git.beanstalkapp.com/transact.git  Cloning into 'transact'...  remote: Enumerating objects: 1458, done.  remote: Counting objects: 100% (1458/1458), done.  remote: Compressing objects: 100% (1355/1355), done.  remote: Total 1458 (delta 492), reused 772 (delta 72)  Receiving objects: 100% (1458/1458), 7.96 MiB | 4.31 MiB/s, done.  Resolving deltas: 100% (492/492), done.  transact % |

1. Log in to the appropriate Azure account (output details shown above)

|  |
| --- |
| % az login |

1. Log in to the Azure Container Registry (ACR)

|  |
| --- |
| az acr login -n bohaACR |
| Login Succeeded |

1. Build the Docker Image

This step takes a file called “Dockerfile” and the source from the current directory and delivers an image (OS+app) to the Desktop Docker.

|  |
| --- |
| % docker build . -t transact.web:v1 |
| [+] Building 54.2s (17/17) FINISHED  => [internal] load build definition from Dockerfile 0.0s  => => transferring dockerfile: 551B 0.0s  => [internal] load .dockerignore 0.0s  => => transferring context: 2B 0.0s  => [internal] load metadata for docker.io/library/nginx:alpine 0.5s  => [internal] load metadata for docker.io/library/node:16 0.5s  => [ui-build 1/6] FROM docker.io/library/node:16@sha256:4b0b5c3db44f567d5d25c80a6fe33a981d911cdae20b39d2395be268aea2cb97 0.0s  => [stage-1 1/5] FROM docker.io/library/nginx:alpine@sha256:eb05700fe7baa6890b74278e39b66b2ed1326831f9ec3ed4bdc6361a4ac2f333 0.0s  => [internal] load build context 1.8s  => => transferring context: 3.59MB 1.7s  => CACHED [ui-build 2/6] WORKDIR /usr/src/app 0.0s  => [ui-build 3/6] COPY . . 7.8s  => [ui-build 4/6] RUN npm install @angular/cli 3.9s  => [ui-build 5/6] RUN npm install 1.7s  => [ui-build 6/6] RUN npm run build 36.6s  => CACHED [stage-1 2/5] WORKDIR /usr/share/nginx/html 0.0s  => CACHED [stage-1 3/5] RUN rm -rf ./\* 0.0s  => CACHED [stage-1 4/5] COPY --from=ui-build /usr/src/app/transact-boha/ . 0.0s  => CACHED [stage-1 5/5] COPY --from=ui-build /usr/src/app/nginx-custom.conf /etc/nginx/conf.d/default.conf 0.0s  => exporting to image 0.0s  => => exporting layers 0.0s  => => writing image sha256:51cc82dd7ff961cd9a08a349ac8425d8d6d733f690ea51a8470ce0d10b6bfbb5 0.0s  => => naming to docker.io/library/transact.web:v1 0.0s  Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them |

1. Desktop Test Image

At this point you can follow Docker Instructions[[1]](#footnote-2) to start the image, connect to it and run the app. Keep in mind Azure services will not be present on your build machine.

1. Tag Image

Tagging the image makes it accessible by the repository (ACR). There is no response.

|  |
| --- |
| % docker tag transact.web:v1 bohaacr.azurecr.io/transact.web:v1 |
| No output indicates successful tagging |

1. Copy Image to Azure

This step copies the tagged image into a repository connected to the Azure Container Registry (ACR)

|  |
| --- |
| % docker push bohaacr.azurecr.io/transact.web:v1 |
| The push refers to repository [bohaacr.azurecr.io/transact.web]  eb04852bc3d3: Pushed  bdea3691fdfc: Pushed  1f718f016c56: Pushed  5f70bf18a086: Pushed  26421a1e4098: Pushed  e81892a9e06a: Pushed  69178c32f0f7: Pushed  39a29773c0aa: Pushed  1b65d8f10357: Pushed  07d3c46c9599: Pushed  v1: digest: sha256:f61fe1ed901d2da60b7f47a9e5d72d70790f0ee52fd2a6a3dddfc0bc958ead0b size: 2399 |

1. Get Registry Credentials

This step updates the deployment server’s (your desktop) credentials from Azure. This allows Kubernetes access to oversee the build and deploy environment from your machine.

|  |
| --- |
| % az aks get-credentials -n bohaAKS -g Transact |
| Merged "bohaAKS" as current context in /Users/ashok/.kube/config |

1. Locate Kubernetes context

Kubernetes stores environment information, including which cluster to use, in a file called “context”. This command will list the contexts. Adjusting which context Kubernetes controls which cluster you will use in delivering the application to Azure.

|  |
| --- |
| % kubectl config get-contexts |
| CURRENT NAME CLUSTER AUTHINFO NAMESPACE  \* bohaAKS bohaAKS clusterUser\_Transact\_bohaAKS |

1. Adjust Kubernetes context

This command instructs Kubernetes to use the cluster called “bohaAKS”.

|  |
| --- |
| % kubectl config use-context bohaAKS |
| Switched to context "bohaAKS". |

1. Define and Activate the Application

In this step we provide kubectl with a yaml file[[2]](#footnote-3) that defines the deployment including service connections, infrastructure requirements and other operating parameters.

|  |
| --- |
| % kubectl apply -f manifest.yaml |
| deployment.apps/transact-web created  service/transact-web created |
|  |

1. Identify External IP Address[[3]](#footnote-4)

We can request kubectl to provide us with the IP address that has been associated with our service. In the output below, the IP address is xx.xx.xxx.xxx

|  |
| --- |
| % kubectl get svc -w |
| NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE  kubernetes ClusterIP 10.0.0.1 <none> 443/TCP 32m  transact-webi LoadBalancer 10.0.248.216 52.226.201.84 80:32210/TCP 10m |

1. Access

The application is accessible at the IP address which could be seen in the output above.

1. Conclusion

This application is secure, fault tolerant and performant. It can be updated from repo to live in minutes. It can be sensitive to business cycles and add compute resources to meet increased demand or remove resources when demand subsides. Cycling servers is automatic and unobtrusive. This is state of the art computing technology that is a powerful tool to support your business goals.

1. <https://docs.docker.com/desktop/windows/>

   https://hub.docker.com/editions/community/docker-ce-desktop-mac [↑](#footnote-ref-2)
2. Commands read in yaml files that contain situation or environment specific parameters. Be sure the view the contents of the file before submitting it to be processed.  [↑](#footnote-ref-3)
3. Actual IP addresses redacted [↑](#footnote-ref-4)