Host Azure DevOps Build containers on AKS

Jonathan Gardner
Dec 18, 2018 · 3 min read ★

Photo by Hal Gatewood on Unsplash

I don’t like waiting in lines, lines of any kind. Hatred of lines is one of my many character flaws. It is this hatred for waiting in lines that drove me to look to find a faster way to run my code through a build pipeline in Azure DevOps. I have been doing quite a bit of work with Kubernetes of late and thought it would be an ideal location. A build server on Kubernetes would allow me to control the build host configuration and a near zero queue time waiting for my builds to fail and show me where I messed up. This article walks through setting up an Azure DevOps agent on Azure Kubernetes Service (AKS).

> *NOTE: This article assumes you have a pretty good handle on Kubernetes basics. If not, links to more Kubernetes information can be found throughout the article.*

Build Agent

Since we should start at the beginning, let's talk about build agents. While it is possible to build a Dockerfile that downloads the agent and configures all of the necessary tools, I am lazy, so I like to start with the base image that Microsoft has already created and published to Docker Hub. I then add some tools that I regularly use in my builds like Terraform and Vuejs. Addition of these tools is reflected in the Dockerfile below.

```
# Base Image
FROM microsoft/vsts-agent

# Update packages and install new ones
RUN sudo apt-get update \
  && sudo apt-get upgrade -y \
  && sudo apt install apt-utils unzip -y

# Install Terraform
RUN curl -O https://releases.hashicorp.com/terraform/0.11.7/terraform_0.11.7_linux_amd64.zip \
  && unzip terraform_0.11.7_linux_amd64.zip -d /usr/local/bin/ \
  && export PATH="$PATH:/usr/local/bin"

# Insntall NPM Package
RUN sudo npm install -g eslint @vue/cli @vue/eslint-config-standard

# Set env variables
ENV VSTS_AGENT='$(hostname)-agent'
ENV VSTS_WORK='/var/vsts/$VSTS_AGENT'

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

Build this image and post it to your container registry of choice. I use Azure Container Registry, but this could easily be Docker Hub or even a self-hosted registry.

Deploy to Kubernetes

Since I run all of my services on Azure, I am using the Azure Kubernetes Service to host my cluster. This deployment includes the deployment, a service to connect to it, an

ingress point, and Let's Encrypt to secure all the things.

> *NOTE: The configuration mentioned in this article is specific to AKS. If deploying this to any other k8s cluster type the DNS/Ingress information will need to be modified.*

The Azure DevOps build agent takes 3 arguments to get connected: the Azure DevOps account name, a personal access token for that account, and a build agent pool name. To keep this information out of my Git repo, I have used Kubernetes Secrets to store these items and then call them in the deployment. My Kubernetes deployment is below.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: vstslinuxbuild
spec:
  replicas: 3
  selector:
    matchLabels:
      app: vstslinuxbuild
  template:
    metadata:
      labels:
        app: vstslinuxbuild
    spec:
      containers:
      - name: vstslinuxbuild
        image: <my vsts build agent image>
        ports:
        - containerPort: 443
        env:
          - name: VSTS_ACCOUNT
            valueFrom:
              secretKeyRef:
                name: vsts
                key: account
          - name: VSTS_TOKEN
            valueFrom:
              secretKeyRef:
                name: vsts
                key: token
          - name: VSTS_POOL
            valueFrom:
              secretKeyRef:
                name: vsts
```

```
                key: pool
        volumeMounts:
          - name: docker-graph-storage
            mountPath: /var/lib/docker
      volumes:
        - name: docker-graph-storage
          emptyDir: {}
```

vsts-linux-build-deployment.yaml hosted with ♡ by GitHub view raw

This deployment referenced the container registry and image and created 3 pods with the environment variables created by the secrets.

We can access these pods individually, but we need a way to access them as a single service, enter Kubernetes Services. The service defined by the yaml file below allows other resources to connect to the 3 replicas with one name: vstslinuxbuld.

```
apiVersion: v1
kind: Service
metadata:
  name: vstslinuxbuild
spec:
  ports:
  - port: 80
    name: web
    protocol: TCP
    targetPort: 80
  - port: 8080
    name: web2
    protocol: TCP
    targetPort: 8080
  - port: 443
    name: secureweb
    protocol: TCP
    targetPort: 443
  selector:
    app: vstslinuxbuild
  type: ClusterIP
```

vsts-linux-build-service.yaml hosted with ♡ by GitHub view raw

While services internal to the AKS cluster can get to the newly created service, external sources can't. External access restriction poses a problem for us to use Azure DevOps to connect to the build agent. External access is also where some of the AKS specific

configurations come into play. This configuration takes advantage of the HTTP application routing in AKS.

Using Let's Encrypt to issue certificates automatically takes a few steps. The first is to create a Kubernetes Cluster Issuer. The code below is used to create the Cluster Issuer.

```yaml
apiVersion: certmanager.k8s.io/v1alpha1
kind: ClusterIssuer
metadata:
  name: letsencrypt-staging
spec:
  acme:
    server: https://acme-staging-v02.api.letsencrypt.org/directory
    email: <<youremailhere@example.com>>
    privateKeySecretRef:
      name: letsencrypt-staging
    http01: {}
```

cluster-issuer.yaml hosted with ♡ by GitHub view raw

With or Cluster Issuer in place, create a Certificate for use by the Ingress Controller.

```yaml
apiVersion: certmanager.k8s.io/v1alpha1
kind: Certificate
metadata:
  name: tls-secret
spec:
  secretName: tls-secret
  dnsNames:
  - <<your dns name>>
  acme:
    config:
    - http01:
        ingressClass: nginx
      domains:
      - << your dns name>>
  issuerRef:
    name: letsencrypt-staging
    kind: ClusterIssuer
```

certificates.yaml hosted with ♡ by GitHub view raw

The final item needed is to set up the Ingress Controller.

```yaml
apiVersion: extensions/v1beta1
```

```yaml
kind: Ingress
metadata:
  name: vstslinuxbuild
  annotations:
    kubernetes.io/ingress.class: nginx
    certmanager.k8s.io/cluster-issuer: letsencrypt-prod
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  tls:
  - hosts:
    - <<your dns name>>
    secretName: tls-secret
  rules:
  - host: <<your dns name>>
    http:
      paths:
      - path: /
        backend:
          serviceName: vstslinuxbuild
          servicePort: 80
      - path: /
        backend:
          serviceName: vstslinuxbuild
          servicePort: 443
```

vsts-linux-build-ingress.yaml hosted with ♡ by GitHub view raw

If everything goes right, the pods running the Azure DevOps agent, will deploy to the cluster and connect automatically be advertised as available in the Agent Pools.

> *NOTE: Jonathan is a Senior Program Manager on the AzureCAT team at Microsoft. The views and optioned expressed on this site are soley those of the original authors and other contributors. These views and opinions do not necessarily represent those of Microsoft.*

Docker Kubernetes Azure Devops Azure Kubernetes Service Azure