Dennis Zielke

Follow 298 Followers About

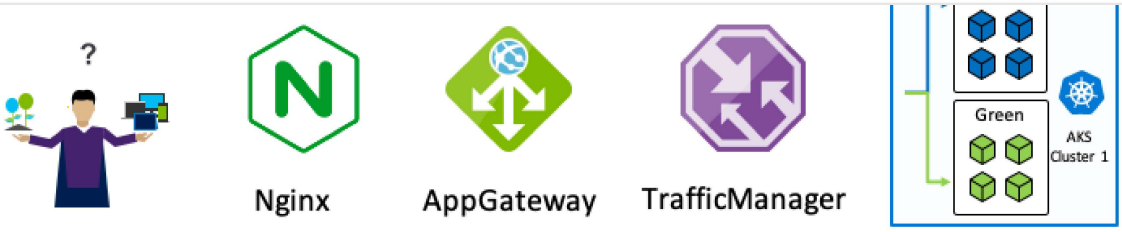
Continuous Kubernetes blue-green deployments on Azure using Nginx, AppGateway or TrafficManager — part 2

Dennis Zielke Jul 7, 2020 · 9 min read

This is part two of my series on advanced deployment practices. If you have been following part 1 we finished with a working continuous deployment pipeline and some rudimentary automated rollback mechanism using helm. Unfortunately we were always replacing the existing helm release, which means that if there is something wrong with the new version, customers will always be impacted and possibly experiencing errors, while we are rolling back to the previous working version. This is obviously not ideal.

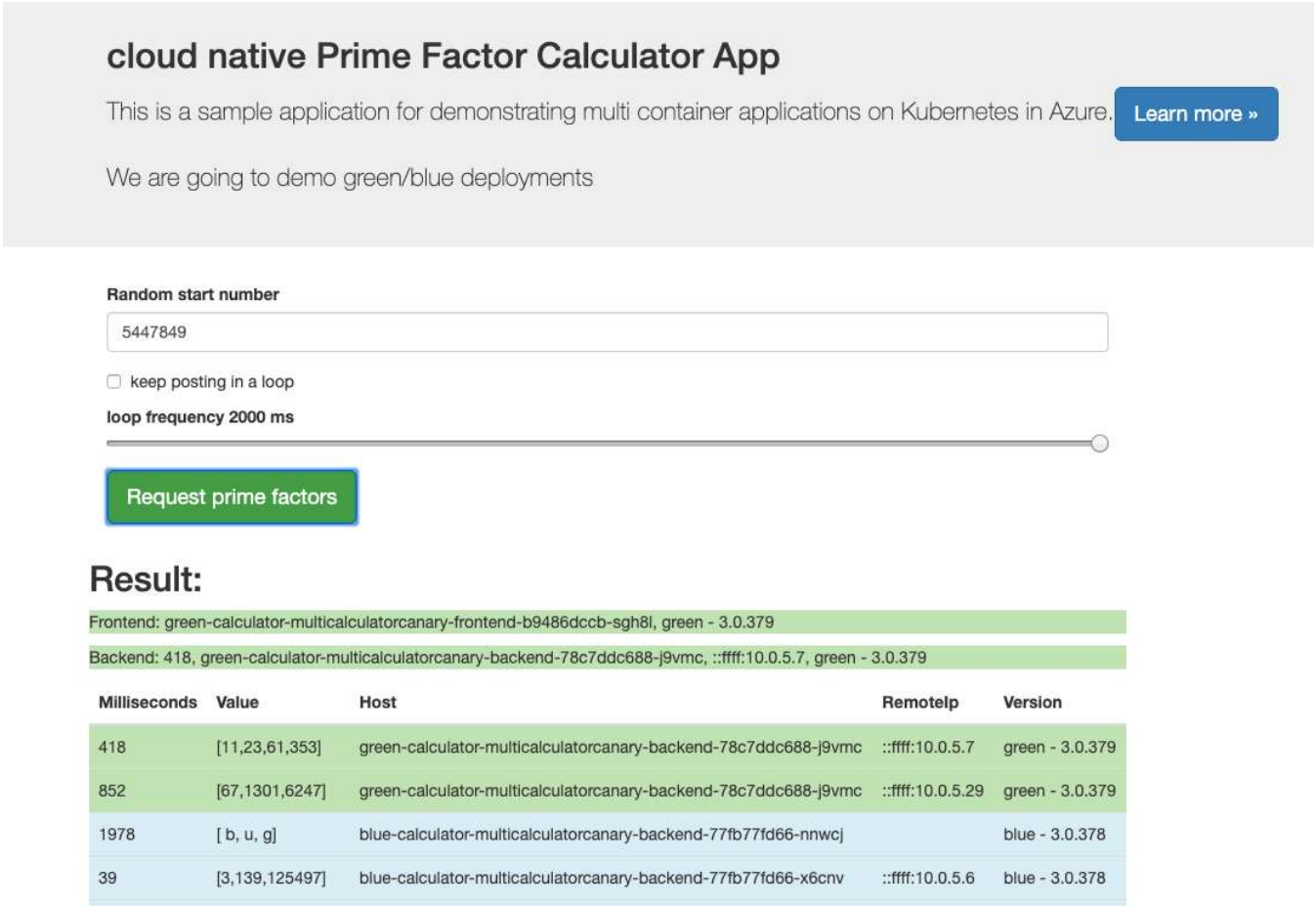
Here now enters the practice of blue/green deployments which means that instead of replacing the previous version (here we refer to this version as *blue*), we bring up the new version (here referred to as the *green* version) next to the existing version, but not expose it to the actual users right away. On the condition of having successfully validated that the green version works correctly, we will promote this version to the public version by changing the routing configuration without downtime. If something is wrong with the green version we can revert back without users every noticing interruptions.

Sound easy right? Maybe you have asked yourself the following questions:



Do you want to know how it can be done?

In this post I want to introduce you into different flavours of implementing blue/green deployments on Azure using Nginx ingress controller (any other ingress controller like Ambassador, Traefik or Kong will also work), Azure Application Gateway and Azure Traffic Manager to compare their capabilities, help you make your choice to tools and provide you with samples to get you starting on adopting more advanced deployment practices in your own apps.



Users will see the blue version replaced with the green version after it has been validated — without downtime

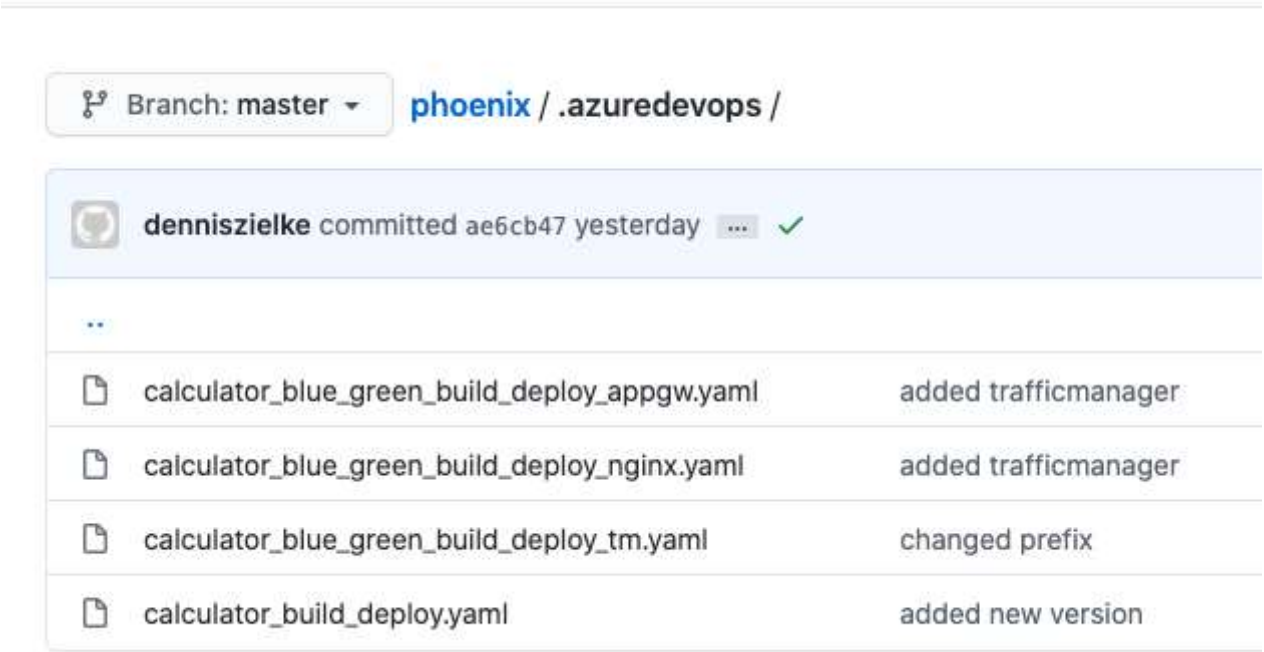
Same as before we are aiming to achieve a couple of design considerations that we typically see in enterprise environments:

without overhead.

2. We want to see, measure and compare telemetry and metrics for each deployment and also evaluate the blue and green deployments side by side.
3. In case of a failure we want to roll back our application automatically and have the ability to start over with the next release.

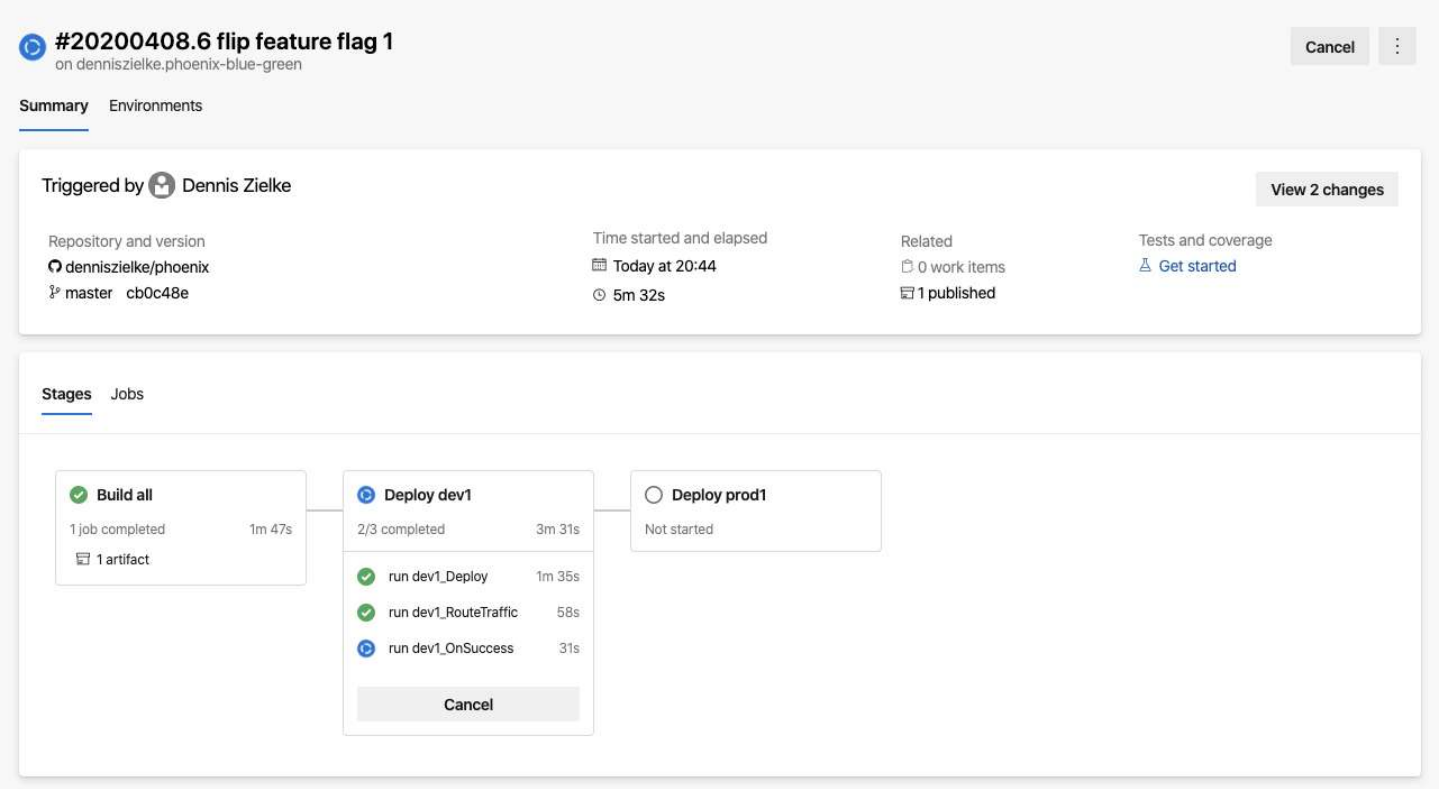
Let's get started

Again we will continue with the phoenix sample application. The easiest way to deploy everything in your own azure subscription is to open up an azure shell , clone your repo there and follow these instructions to deploy at least one environment which should bring up an instance of AKS, KeyVault, Application Insights, Azure Container Registry, Application Gateway and Traffic Manager. You can check the last post on how to configure azure devops project and permissions with the newly created service principals.



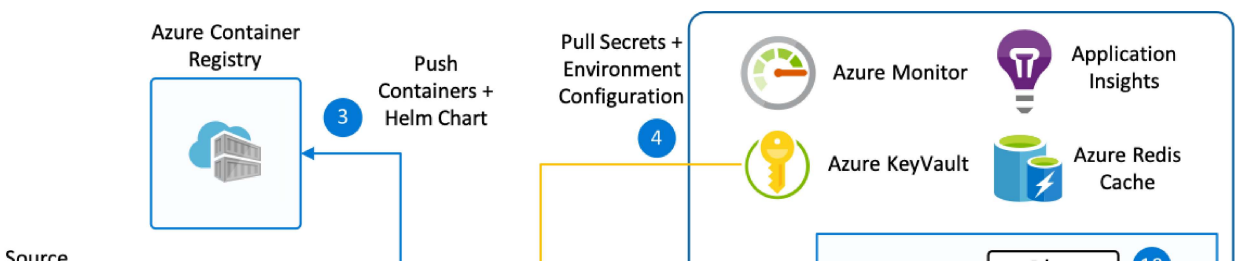
In the phoenix repository you will find an azure devops build and deployment pipeline template.

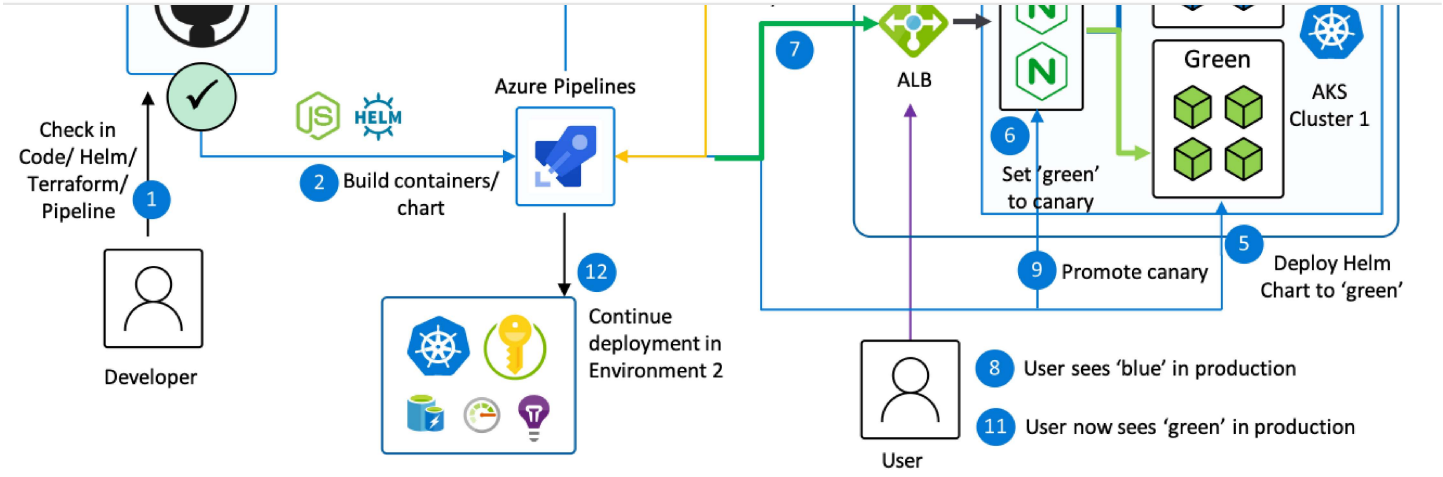
configure the *AZURE_CONTAINER_REGISTRY_NAME* for your deployment and the *AZURE_KEYVAULT_NAME* for each environment secret store in the pipeline after you have imported it into your azure devops project.



The imported pipeline will allow you to see how the blue/green deployment progresses

We will start with the Nginx deployment, which will leverage a single azure load balancer in front of our Nginx ingress controller in the AKS cluster. The idea is to deploy our application multiple times in different namespaces leveraging the canary feature, which allows us to hook up two ingress objects to the same dns name and route them to different backend Kubernetes services depending on the existence of a custom HTTP header in the request.





We are using the canary feature to route the traffic between the blue or the green version.

To make the automation work, we also need an extra helm variable called "canary" which will be used to define which of the two deployments is currently the canary. Before deploying a new version we will read back all deployed helm charts from the cluster, determine if there is already a canary deployment or not and perform a new deployment in either the blue or the green slot — which ever is currently not used or in a older version than the other slot.

scripts > deploy_multicalculator_blue_green_nginx.sh

```
82  if [ "$GREEN_VERSION" -gt "$BLUE_VERSION" ]; then
83      echo "Green is higher than blue - deploying blue"
84      SLOT="blue"
85  else
86      echo "Blue is higher than green - deploying green"
87      SLOT="green"
88  fi
89
90  if [ "$GREEN_VERSION" == "0" ]; then
91  if [ "$BLUE_VERSION" == "0" ]; then
92      NOCANARY="true"
93  fi
94  fi
95
96  DEPLOY_NAMESPACE=$SLOT-$KUBERNETES_NAMESPACE
97  RELEASE=$SLOT-calculator
98  kubectl create ns $DEPLOY_NAMESPACE
99
100 if [ "$NOCANARY" == "true" ]; then
101 helm upgrade $RELEASE $AZURE_CONTAINER_REGISTRY_NAME/multicalculatorcanary --namespace $DEPLOY_NAMESPACE --install --set replicaCount=4
    --set image.frontendTag=$BUILD_BUILDNUMBER --set image.backendTag=$BUILD_BUILDNUMBER --set image.repository=$AZURE_CONTAINER_REGISTRY_URL
    --set dependencies.useAppInsights=true --set dependencies.appInsightsSecretValue=$APPINSIGHTS_KEY --set dependencies.useAzureRedis=true
    --set dependencies.redisHostValue=$REDIS_HOST --set dependencies.redisKeyValue=$REDIS_AUTH --set slot=$SLOT --set ingress.
    host=$INGRESS_FQDN --set ingress.class=nginx --wait --timeout 60s
102 else
103 helm upgrade $RELEASE $AZURE_CONTAINER_REGISTRY_NAME/multicalculatorcanary --namespace $DEPLOY_NAMESPACE --install --set replicaCount=2
    --set image.frontendTag=$BUILD_BUILDNUMBER --set image.backendTag=$BUILD_BUILDNUMBER --set image.repository=$AZURE_CONTAINER_REGISTRY_URL
    --set dependencies.useAppInsights=true --set dependencies.appInsightsSecretValue=$APPINSIGHTS_KEY --set dependencies.useAzureRedis=true
    --set dependencies.redisHostValue=$REDIS_HOST --set dependencies.redisKeyValue=$REDIS_AUTH --set slot=$SLOT --set ingress.
    host=$INGRESS_FQDN --set ingress.class=nginx --set canary=true --set ingress.weigth=0  --wait --timeout 60s      You, a day ago • added tfm
104 fi
```

The deployed helm charts in the cluster will act as source of truth — no other state store is required.

header in the ingress object. At the same time we have defined the weight of the canary to be 0, which will ensure that no traffic will be routed there unless the header is set.

charts > multicalculatorcanary > templates > ingress.yaml > {map}

You, a few seconds ago | 2 authors (Dennis Zielke and others)

```
{{- if .Values.ingress.enabled -}}
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: {{ include "multicalculatorcanary.fullname" . }}
  labels:
    {{- include "multicalculatorcanary.labels" . | nindent 4 }}
  annotations:
    kubernetes.io/ingress.class: {{ .Values.ingress.class }}
  {{- if .Values.canary }}
    appgw.ingress.kubernetes.io/connection-draining: "true"
    appgw.ingress.kubernetes.io/connection-draining-timeout: "60"
    appgw.ingress.kubernetes.io/backend-path-prefix: "/"
    nginx.ingress.kubernetes.io/canary: "true"
    nginx.ingress.kubernetes.io/canary-by-header: "canary"
    nginx.ingress.kubernetes.io/canary-weight: "{{ .Values.ingress.weight }}"
  {{- with .Values.ingress.annotations }}
    {{- toYaml . | nindent 4 }}
  {{- end }}
```

The canary variable will signal if the deployment is a canary and ensure the right ingress annotations.

As you can see below in the initial version we only have the blue version 3.0.378 deployed in the blue-calculator namespace.

```
➜ ~ (⎈ |dztenix-472:default) helm list --all-namespaces
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS          CHART                           APP VERSION
blue-calculator blue-calculator 3               2020-04-08 18:41:58.31606889 +0000 UTC  deployed        multicalculatorcanary-0.1.378   3.0.378
nginx-ingress   nginx           1               2020-04-08 12:57:42.692218 +0200 CEST   deployed        nginx-ingress-1.36.0            0.30.0
```

Only the blue version is deployed.

Now in step 5 of our process we are deploying the green version 3.0.379 into the green-calculator namespace — but with the canary annotation in the ingress object.

```
➜ ~ (⎈ |dztenix-472:default) helm list --all-namespaces
NAME                NAMESPACE           REVISION        UPDATED                                 STATUS          CHART                                   APP VERSION
blue-calculator     blue-calculator     3               2020-04-08 18:41:58.31606889 +0000 UTC  deployed        multicalculatorcanary-0.1.373.0.378
green-calculator    green-calculator    1               2020-04-08 18:47:24.374216313 +0000 UTC deployed        multicalculatorcanary-0.1.373.0.379
nginx-ingress       nginx               1               2020-04-08 12:57:42.692218 +0200 CEST   deployed        nginx-ingress-1.36.0            0.30.0
```

Both blue and green are deployed in the same cluster

"canary: always" in the HTTP request we are ending up in the green version.

```
echo "Checking production curl http://$INGRESS_FQDN/ping without canary header"
curl -s -H "canary: never" -H "Host: $INGRESS_FQDN" http://$INGRESS_FQDN/ping
echo ""
echo "Checking staging slot curl http://$INGRESS_FQDN/ping  with header 'canary: always'"
curl -s -H "canary: always" -H "Host: $INGRESS_FQDN" http://$INGRESS_FQDN/ping
echo ""

Checking production curl http://51.138.7.236.xip.io/ping without canary header
{"response":"pong!","host":"blue-calculator-multicalculatorcanary-frontend-cfc6c5b99-7xnc5","version":"blue - 3.0.378"}
Checking staging slot curl http://51.138.7.236.xip.io/ping  with header 'canary: always'
{"response":"pong!","host":"green-calculator-multicalculatorcanary-frontend-b9486dccb-sgh8l","version":"green - 3.0.379"}
```

Compare the -H "canary:never" header value and you can see that we end up getting different versions from the /ping api

At this point we are in step 7 in our process diagram using the *routeTraffic* step in azure devops while using the custom headers to validate that the application works as expected. Only if that is the case we will allow the deployment to progress and end up in the *on success* step — if not we will execute the *on failure* step to clean up the canary and delete the helm deployment in the green slot.

Assuming everything worked out we will promote the canary by upgrading the helm deployment with canary=true and weight=100. This will basically override the existing ingress object for the same DNS name in the other namespace and ensure that from now on all traffic will be routed to the green deployment slot.

scripts > on_deploy_success_multicalculator_blue_green_nginx.sh

```
56  if [ "$CANARY_SLOT" != "none" ]; then
57  echo "Canary $CANARY_SLOT will be promoted to production"
58  DEPLOY_NAMESPACE=$CANARY_SLOT-$KUBERNETES_NAMESPACE
59  RELEASE=$CANARY_SLOT-calculator
60
61  helm upgrade $RELEASE $AZURE_CONTAINER_REGISTRY_NAME/multicalculatorcanary --namespace $DEPLOY_NAMESPACE --install --set replicaCount=2
    --set image.frontendTag=$BUILD_BUILDNUMBER --set image.backendTag=$BUILD_BUILDNUMBER --set image.repository=$AZURE_CONTAINER_REGISTRY_URL
    --set dependencies.useAppInsights=true --set dependencies.appInsightsSecretValue=$APPINSIGHTS_KEY --set dependencies.useAzureRedis=true
    --set dependencies.redisHostValue=$REDIS_HOST --set dependencies.redisKeyValue=$REDIS_AUTH --set slot=$CANARY_SLOT --set ingress.
    class=nginx --set ingress.host=$INGRESS_FQDN --set canary=true --set ingress.weigth=100 --wait --timeout 45s
62
63  if [ "$PRODUCTION_SLOT" != "none" ]; then
64  echo "Production $PRODUCTION_SLOT will be deleted"
65  DEPLOY_NAMESPACE=$PRODUCTION_SLOT-$KUBERNETES_NAMESPACE
66  RELEASE=$PRODUCTION_SLOT-calculator
67  helm delete $RELEASE --namespace $DEPLOY_NAMESPACE
68  fi
```

By setting canary=true and weight=100 we will override the production slot and force all traffic to the green slot.

Now we will simply delete the existing production slot and as a last step do one more upgrade of the existing green slot with canary=false to promote it to be the new production slot and ensure we can start over at the beginning when the next release with a new canary gets deployed.

```
curl -s -H "canary: always" -H "Host: $INGRESS_FQDN" http://$INGRESS_FQDN/ping
echo ""

Checking production curl http://51.138.7.236.xip.io/ping without canary header
{"response":"pong!","host":"blue-calculator-multicalculatorcanary-frontend-cfc6c5b99-7xnc5","version":"blue - 3.0.378"}
Checking staging slot curl http://51.138.7.236.xip.io/ping  with header 'canary: always'
{"response":"pong!","host":"blue-calculator-multicalculatorcanary-frontend-cfc6c5b99-gh9kf","version":"blue - 3.0.378"}
```

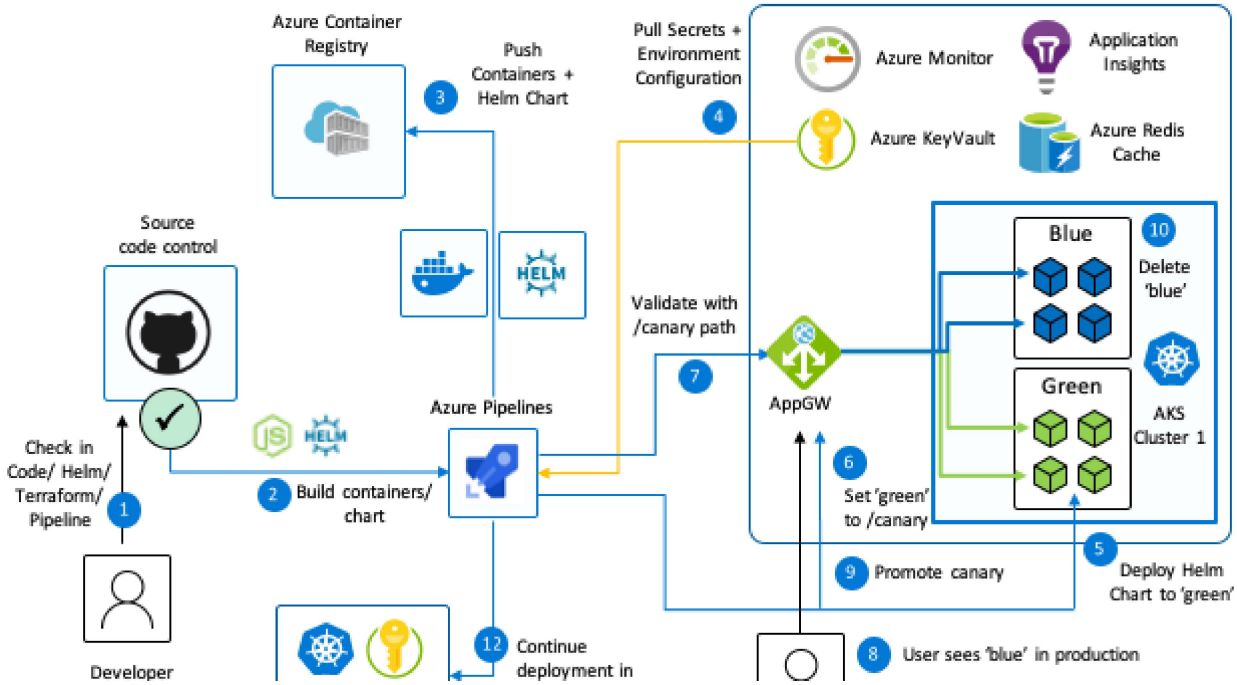
After upgrading the green slot it has now become the production slot for the next iteration.

Since we are using Nginx and Application Insights in our app that means we are getting very detailed metrics, logs and dashboards which will allow us to compare the performance and functionality of our new apps. With a little tuning we can easily set up a Grafana dashboard that allows us to compare both deployments side by side.

That is it for using Nginx!

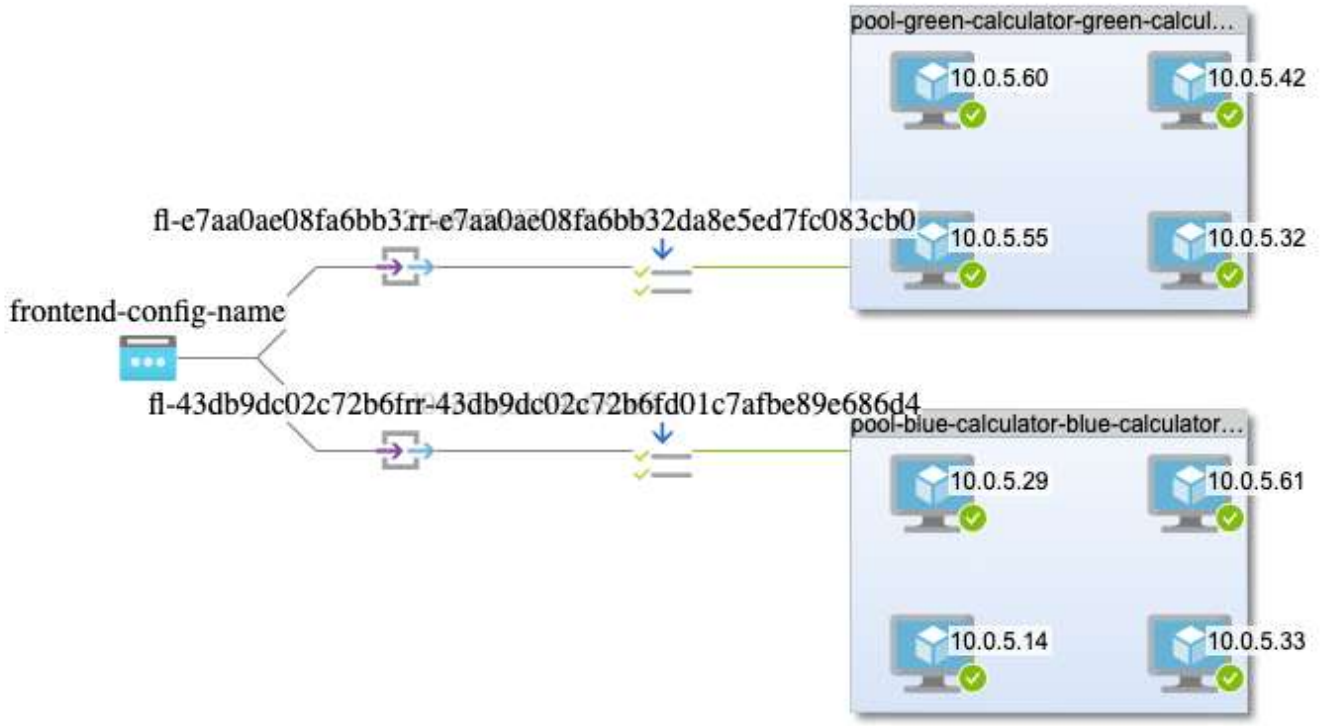
The next type of deployment will leverage the azure application gateway as ingress controller to achieve the same. As of today the AKS managed appgateway addon is officially supported but still cannot be automatically deployed via terraform. That means you have to manually activate the addon after you have deployed the cluster as described here.

The process will look like this and is implemented in the appgw azure pipeline template:



The process is similar to the Nginx deployment, only here we have to use different routes for each deployment.

In comparison to the Nginx ingress controller the AppGateway Ingress controller is running outside of the AKS cluster as its own dedicated managed service, but just as you would expect it will route traffic directly to the individual containers, which is why our application does not have to change to make it work.



Screenshot from the network diagram of the application gateway, where we can see the different backend pools.

The essential process will be the same and controlled by the helm deployment parameters, which will differ only in the ingress.class.

scripts > deploy_multicalulator_blue_green_appgw.sh

```
82  if [ "$GREEN_VERSION" -gt "$BLUE_VERSION" ]; then
83      echo "Green is higher than blue - deploying blue"
84      SLOT="blue"
85  else
86      echo "Blue is higher than green - deploying green"
87      SLOT="green"
88  fi
89
90  if [ "$GREEN_VERSION" == "0" ]; then
91  if [ "$BLUE_VERSION" == "0" ]; then
92      NOCANARY="true"
93  fi
94  fi
95
96  DEPLOY_NAMESPACE=$SLOT-$KUBERNETES_NAMESPACE
```

```
    --set image.frontendTag=$BUILD_BUILDNUMBER --set image.backendTag=$BUILD_BUILDNUMBER --set image.repository=$AZURE_CONTAINER_REGISTRY_URL
    --set dependencies.useAppInsights=true --set dependencies.appInsightsSecretValue=$APPINSIGHTS_KEY --set dependencies.useAzureRedis=true
    --set dependencies.redisHostValue=$REDIS_HOST --set dependencies.redisKeyValue=$REDIS_AUTH --set slot=$SLOT --set ingress.class=azure/
    application-gateway --set ingress.host=$APPGW_FQDN --wait --timeout 60s
102 else
103 helm upgrade $RELEASE $AZURE_CONTAINER_REGISTRY_NAME/multicalculatorcanary --namespace $DEPLOY_NAMESPACE --install --set replicaCount=2
    --set image.frontendTag=$BUILD_BUILDNUMBER --set image.backendTag=$BUILD_BUILDNUMBER --set image.repository=$AZURE_CONTAINER_REGISTRY_URL
    --set dependencies.useAppInsights=true --set dependencies.appInsightsSecretValue=$APPINSIGHTS_KEY --set dependencies.useAzureRedis=true
    --set dependencies.redisHostValue=$REDIS_HOST --set dependencies.redisKeyValue=$REDIS_AUTH --set slot=$SLOT --set ingress.class=azure/
    application-gateway --set ingress.host=$APPGW_FQDN --set canary=true --wait --timeout 60s      You, a day ago • added tfm
104 fi
```

Same as before we will use ingress objects to register the routing to the right version.

Since the Application Gateway does not support canary routing by HTTP header we are in this case exposing the canary under a different HTTP route named /canary. This allows us same as before validate the canary deployment after it has been deployed and perform a switch between the / — route and the /canary -route after we have confirmed that it works as expected.

charts > multicalculatorcanary > templates > ingress.yaml > {map}

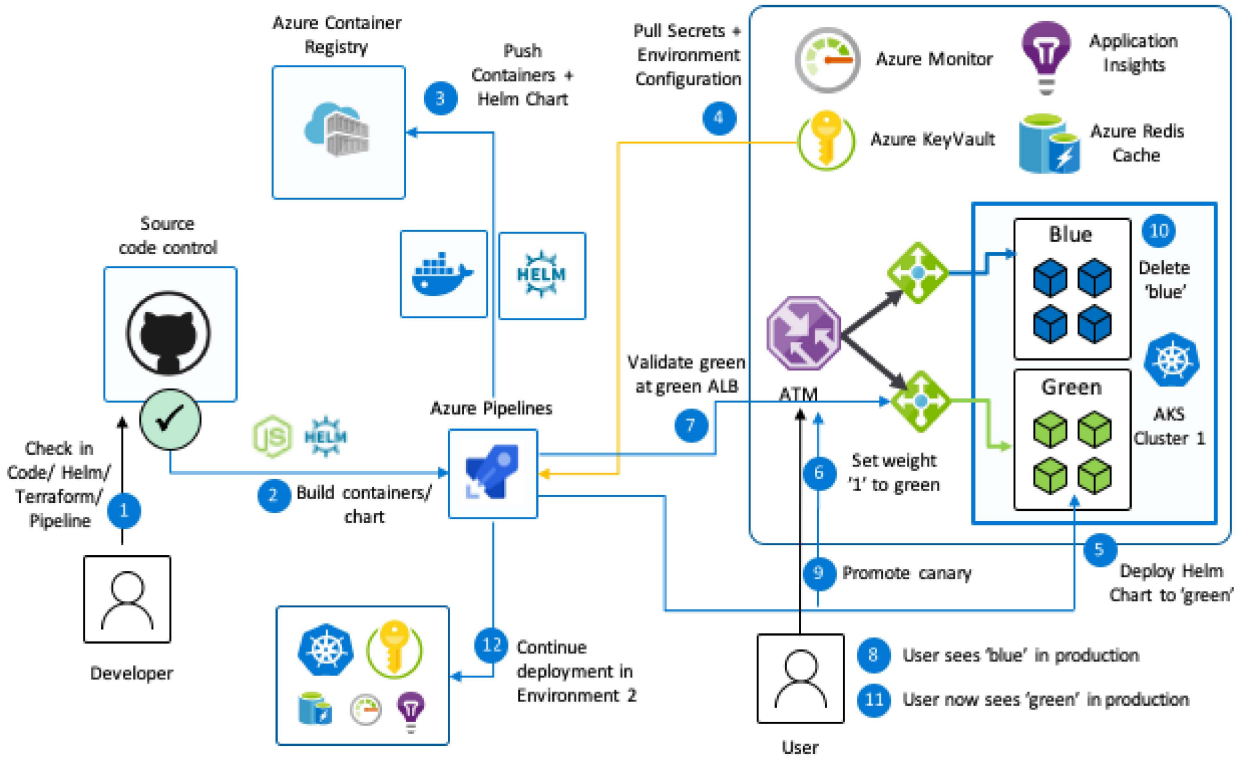
```
35        paths:
36    {{- if and (.Values.canary) (eq .Values.ingress.class "azure/application-gateway") }}
37          - path: /canary/*
38            backend:
39              serviceName: {{ include "multicalculatorcanary.fullname" . }}-frontend-svc
40              servicePort: {{ .Values.service.port }}
41    {{- else }}
42          - path:
43            backend:
44              serviceName: {{ include "multicalculatorcanary.fullname" . }}-frontend-svc
45              servicePort: {{ .Values.service.port }}
46    {{- end }}
47  {{- end }}
48
```

We are switching on the ingress class to publish our app under a different sub path for a canary route

In terms of metrics I have not found a way to compare the traffic monitoring statistics between our two deployment slots, which is a disadvantage, since you have to rely solely on the application insights metrics. However the nice thing about this is that the application gateway can also act as a router between multiple clusters — if they are part of the same VNET. Be aware that the managed Application Gateway Ingress Controller Addon does not work in shared mode, that means if you want one Application Gateway for multiple cluster you have to manually deploy and configure the ingress controller via helm. This makes it possible to perform blue/green deployments not only with our own applications running in different namespaces, but also with blue and green versions running in different clusters. If you are interested in

As final variation we want to take a look at blue/green deployments using Azure Traffic Manager, which is again an azure managed service that works based on DNS resolution. It has the advantage that it will also work on multiple clusters, which do not need to be part of the same VNET or even the same Azure region. However since it is DNS based it will also require is to bring up dedicated Load Balancer IPs for each deployment slot with dedicated DNS entries.

The overall process will look like this:



When using azure traffic manager for blue/green our deployments need to have dedicated DNS entries.

Our instance of azure traffic manager will be configured with the weighted traffic-routing method- which will defines how a HTTP request for our Traffic Manager DNS will get resolved to one of our AKS Load balancer DNS names. This will allow us to decide if the blue or the green version should be preferred by the traffic manager profile.

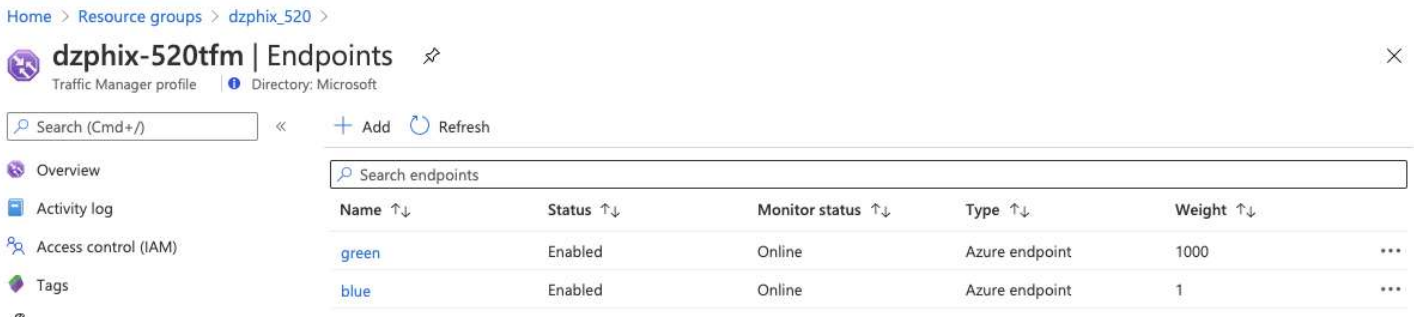
the endpoints and the routing weight distribution. Our existing shell scripts already have the right spots for these steps and during our terraform deployment we have already set up dedicated load balancer ips and dns entries for both blue and green deployment slots.

scripts > deploy_multicalulator_blue_green_tm.sh

```
105
106  WEIGHT=1
107  if [ "$NOCANARY" == "true" ]; then
108      WEIGHT=1000      You, a day ago • increased weights
109      helm upgrade $RELEASE $AZURE_CONTAINER_REGISTRY_NAME/multicalculatorcanary --namespace $DEPLOY_NAMESPACE --install --set replicaCount=4
         --set image.frontendTag=$BUILD_BUILDNUMBER --set image.backendTag=$BUILD_BUILDNUMBER --set image.
         repository=$AZURE_CONTAINER_REGISTRY_URL --set dependencies.useAppInsights=true --set dependencies.
         appInsightsSecretValue=$APPINSIGHTS_KEY --set dependencies.useAzureRedis=true --set dependencies.redisHostValue=$REDIS_HOST --set
         dependencies.redisKeyValue=$REDIS_AUTH --set slot=$SLOT --set ingress.enabled=false --set service.type=LoadBalancer --set service.
         dns=$DNS_LABEL --set service.ip=$IP --set canary=false --wait --timeout 60s
110  else
111      helm upgrade $RELEASE $AZURE_CONTAINER_REGISTRY_NAME/multicalculatorcanary --namespace $DEPLOY_NAMESPACE --install --set replicaCount=4
         --set image.frontendTag=$BUILD_BUILDNUMBER --set image.backendTag=$BUILD_BUILDNUMBER --set image.
         repository=$AZURE_CONTAINER_REGISTRY_URL --set dependencies.useAppInsights=true --set dependencies.
         appInsightsSecretValue=$APPINSIGHTS_KEY --set dependencies.useAzureRedis=true --set dependencies.redisHostValue=$REDIS_HOST --set
         dependencies.redisKeyValue=$REDIS_AUTH --set slot=$SLOT --set ingress.enabled=false --set service.type=LoadBalancer --set service.
         dns=$DNS_LABEL --set service.ip=$IP --set canary=true --wait --timeout 60s
112  fi
113
114  EXISTS=$(az network traffic-manager endpoint show --name $SLOT -g $AKS_GROUP --profile-name $TFM_NAME --type azureEndpoints --query name -o
     tsv)
115
116  if [ "$EXISTS" == $SLOT ]; then
117      az network traffic-manager endpoint update -g $AKS_GROUP --profile-name $TFM_NAME \
118      -n $SLOT --type azureEndpoints --target-resource-id $IP_RESOURCE_ID --endpoint-status enabled \
119      --weight $WEIGHT --custom-headers host=$DNS
120  else
121      az network traffic-manager endpoint create -g $AKS_GROUP --profile-name $TFM_NAME \
122      -n $SLOT --type azureEndpoints --target-resource-id $IP_RESOURCE_ID --endpoint-status enabled \
123      --weight $WEIGHT --custom-headers host=$DNS
124  fi
```

In our deployment steps we will configure the weight of the endpoints and connect them to our traffic manager profile.

The azure traffic manager will also support more than two public endpoints and allow us to register multiple applications, clusters and instances that can be connected behind the same profile.



The weight and the availability status of each endpoint will decide if traffic will be routed to the individual endpoints.

In terms of metrics the azure traffic manager brings even less metrics than the

slots.

For final comparison I wanted to summarise some of the key aspects of the different implementations and encourage you to try all of them using the samples provided.

	Nginx	Azure AppGW	Azure TrafficManager
Blue/ Green deployments	Yes	Yes	Yes
Fully managed Service	No, managed by you	Yes, as AKS addon for V2	Yes, configured by you
Canary deployment	Paths, Header, Cookies	Paths	No
Connection draining	Yes	Yes	No
Traffic Telemetry	Yes, Grafana	Some, Azure Monitor	No
Works on private IPs	Yes	Yes on V1, not fully private on V2	No
Targeting multiple cluster	No	Yes, but same VNET	Yes, multiple regions

Short summary on the key differences between the technologies used here

I hope you learned something today and I am curious to hear your feedback on how you have implemented your blue/green deployment practice, if my story and samples have helped you to do it and if there is something I have missed.

The end ;)

Azure Azure Devops Kubernetes Blue Green Deployment DevOps