Creating and Deploying a StreamSets Microservice on Kubernetes

Table of Contents

Introduction

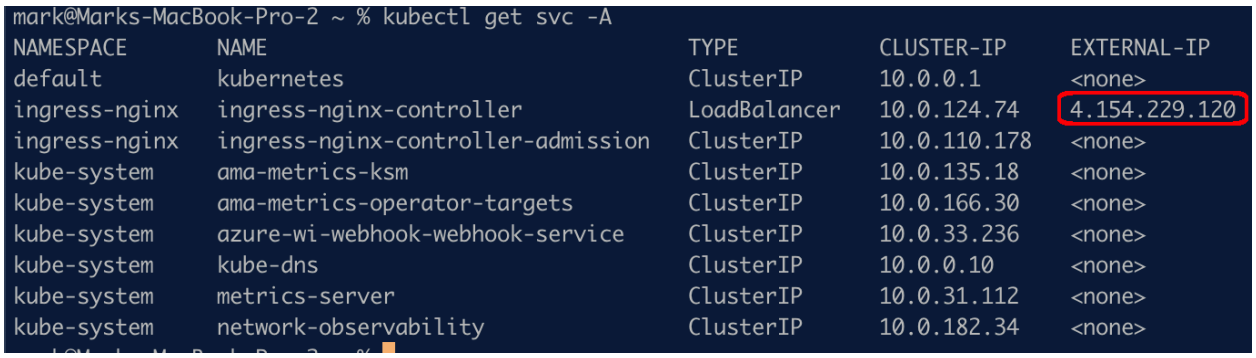
This document provides an example of creating and deploying a StreamSets Microservice Pipeline on Kubernetes with TLS. This example uses ingress-nginx as an ingress controller, but you could easily use your preferred ingress mechanism instead.

The project here contains the example's artifacts.

Deploy and configure the ingress controller

Deploy the ingress controller using the command here.

Get the external IP of the ingress controller:



```
mark@Marks-MacBook-Pro-2 ~ % kubectl get svc -A
NAMESPACE       NAME                                 TYPE           CLUSTER-IP     EXTERNAL-IP
default         kubernetes                           ClusterIP      10.0.0.1       <none>
ingress-nginx   ingress-nginx-controller             LoadBalancer   10.0.124.74    4.154.229.120
ingress-nginx   ingress-nginx-controller-admission   ClusterIP      10.0.110.178   <none>
kube-system     ama-metrics-ksm                      ClusterIP      10.0.135.18    <none>
kube-system     ama-metrics-operator-targets         ClusterIP      10.0.166.30    <none>
kube-system     azure-wi-webhook-webhook-service     ClusterIP      10.0.33.236    <none>
kube-system     kube-dns                             ClusterIP      10.0.0.10      <none>
kube-system     metrics-server                       ClusterIP      10.0.31.112    <none>
kube-system     network-observability                ClusterIP      10.0.182.34    <none>
```

Map a DNS name to the ingress controller's external IP. I'll use the DNS name `aks.onefoursix.com`

Obtain a TLS cert and key for the ingress controller's hostname

Obtain a TLS cert and key for the ingress controller's hostname. I'll use a wildcard cert and key for `*.onefoursix.com` generated using Lets Encrypt in the files `tls.crt` and `tls.key`.

Create a Namespace for the StreamSets components

I'll create a namespace named ns1:

```
$ kubectl create ns ns1
```

Store the tls cert and key in a tls secret

```
$ kubectl create secret tls streamsets-tls \
  --key ~/certs/tls.key --cert ~/certs/tls.crt
```

Create a StreamSets Kubernetes Environment

Create a StreamSets Kubernetes Environment and deploy a StreamSets Kubernetes Agent See the docs here.

Launch a StreamSets Kubernetes Deployment

Launch a StreamSets Kubernetes deployment in the namespace `ns1`. See the docs here.

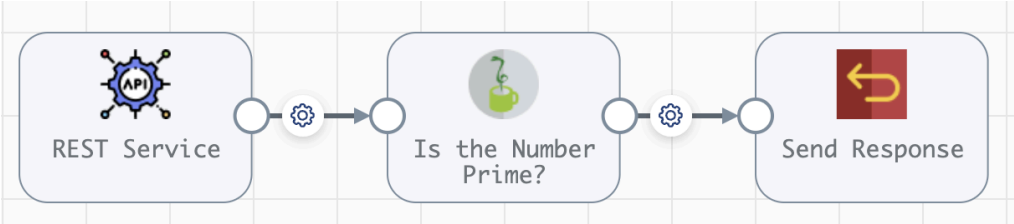
Include the Jython Evaluator stage library in the deployment configuration as the logic of the microservice is implemented in Jython.

This example requires StreamSets engine version 6.1.0 or higher.

Import the Microservice pipeline

Import the `IsPrime Microservice` pipeline from the archive here.

The pipeline looks like this:



The `IsPrime Microservice` takes a POST request payload like `{"number": 11}` like this:

```
$ curl -v https://aks.onefoursix.com/is-prime \
  -H "Content-Type:application/json" \
  -H "X-SDC-APPLICATION-ID:app1" \
  -d '{"number": 11}'
```

The microservice returns `true` or `false` if the number is prime or not. For example:

```
{"httpStatusCode":200,"data":[{"number":11,"is_prime":"true"}],"error":[]}
```

Create a Keystore for the Pipeline

Important note: You may encounter downstream errors trying to use keystores generated on macOS. If you are working on macOS I recommend you generate the keystore on Linux and then copy the keystore back to your Mac for the subsequent steps.

Create a Java keystore in pkcs12 format for the pipeline's REST Service connector. I'll use the same wildcard TLS cert and key as described in the steps above using a command like this:

```
$ openssl pkcs12 -export -in tls.crt -inkey tls.key -name is-prime
-out is-prime-keystore.p12
```

This gives me the keystore file `is-prime-keystore.p12`.

You will be prompted to enter and confirm a password for the keystore; store that value in a text file on your local machine named `is-prime-keystore-password.txt`

Store the keystore and password in secrets

Store the keystore and password in Kubernetes secrets:

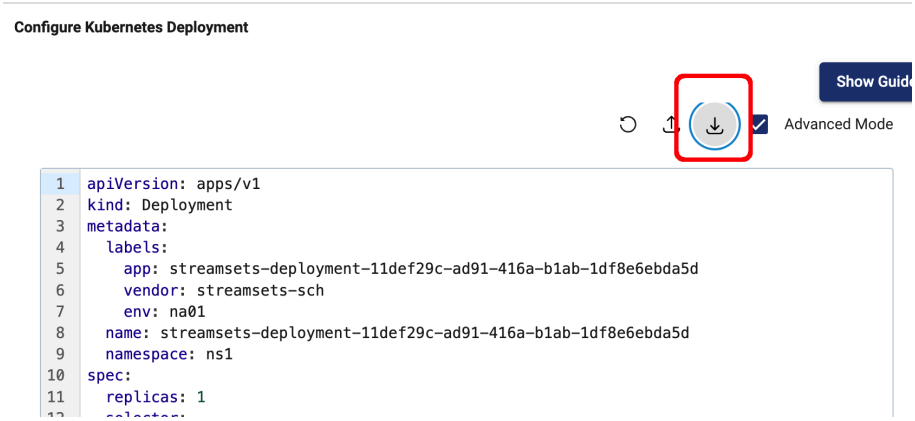
```
$ kubectl -n ns1 create secret generic is-prime-keystore
--from-file=is-prime-keystore.p12

$ kubectl -n ns1 create secret generic is-prime-keystore-password
--from-file=is-prime-keystore-password.txt
```

VolumeMount the keystore and password secrets into the StreamSets engine

In the StreamSets Kubernetes deployment's advanced mode add Volumes for the two secrets and VolumeMount the secrets into the Deployment. See the yaml file here for an example.

Tip: Don't try to edit the yaml within the StreamSets UI; instead, click the download button (see the screenshot below) to save the generated yaml to your local machine, edit the yaml using your favorite text editor, and then import the file back into the StreamSets UI.



Once you have imported the edited yaml back into the StreamSets UI and saved your changes, the StreamSets engine pod(s) will get redeployed, and once they are back online, you can confirm the Volume and VolumeMounts are working by seeing if the two files are present in engine's `/resources` directory, using a command like this:

```
$ kubectl exec -it <engine-pod-id> -- bash -c 'ls -l /resources'
```

You should see output like this:

```
mark@Marks-MacBook-Pro-2 ~ % k exec -it streamsets-deployment-11def29c-ad91-416a-b1a
b-1df8e6ebda5d2n5xx -- bash -c 'ls -l /resources'
total 12
-rw-r--r-- 1 root root   12 Mar  5 20:25 is-prime-keystore-password.txt
-rw-r--r-- 1 root root 4420 Mar  5 20:25 is-prime-keystore.p12
```

Create a Service

Create a Kubernetes ClusterIP Service to expose the microservice pipeline. I'll use port 8000 for the both the pipeline's and the services port

In order to get the value for the service's selector, describe one of the deployment's engine pod and grab its label using a command like this:

```
$ kubectl describe po <engine-pod-id> | grep Labels
```

You should see output like this:

```
mark@Marks-MacBook-Pro-2 ~ % kubectl describe po streamsets-deployment-11def29c-ad91-4
16a-b1ab-1df8e6ebda5d9mfhn | grep Labels
Labels:           app=streamsets-deployment-11def29c-ad91-416a-b1ab-1df8e6ebda5d
```

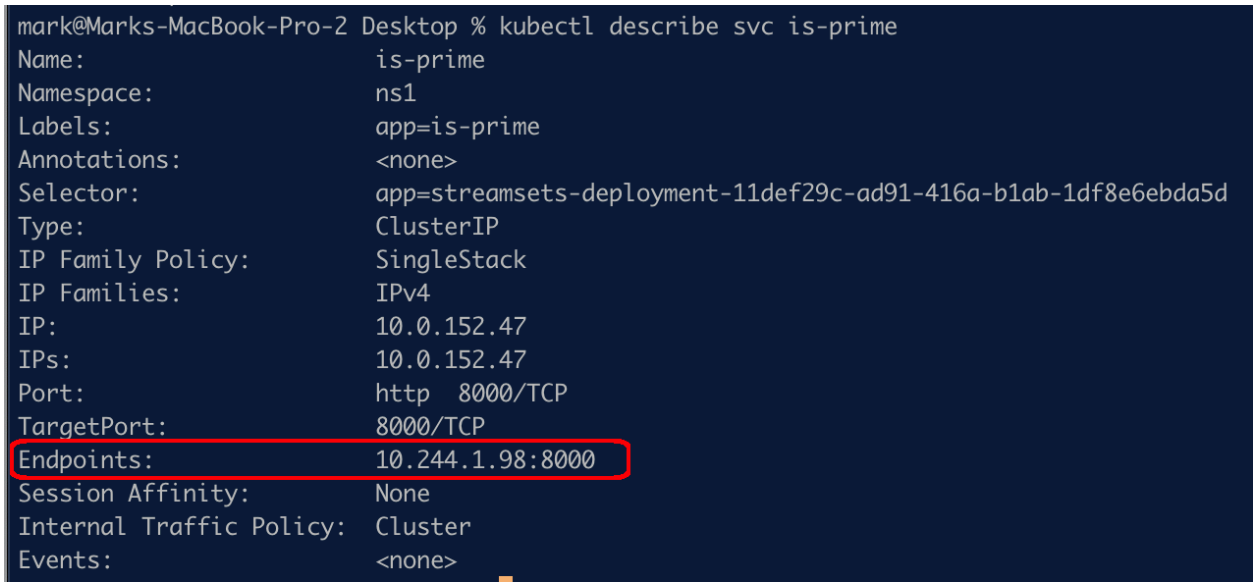
I created a file named `is-prime-service.yaml` on my local machine with this content, including the value of the app label in the selector:

```
apiVersion: v1
kind: Service
metadata:
  name: is-prime
  namespace: ns1
  labels:
    app: is-prime
spec:
  type: ClusterIP
  ports:
  - name: http
    port: 8000
    targetPort: 8000
    protocol: TCP
  selector:
    app: streamsets-deployment-11def29c-ad91-416a-b1ab-1df8e6ebda5d
```

I'll apply that yaml to create the service:

```
$ kubectl -n ns1 apply -f is-prime-service.yaml
```

Describe the service and make sure it has a valid backend:



→ That confirms the service's selector is correct.

Create an Ingress

We'll need an ingress to expose the ClusterIP service to external callers, and the syntax for the ingress depends on your choice of ingress controller or gateway. On OpenShift you could create a Route rather than an ingress. This example is for ingress-nginx.

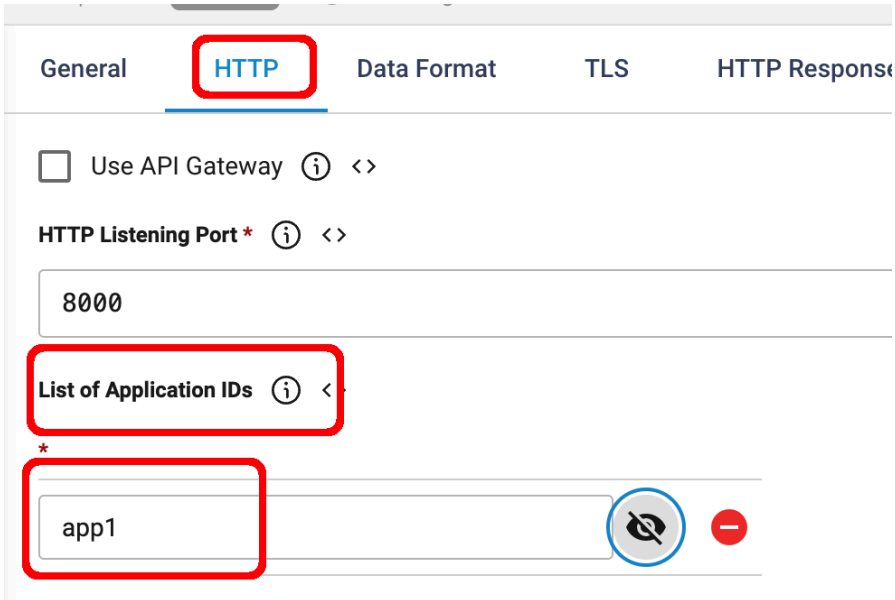
I created a file named `is-prime-ingress.yaml` on my local machine with this content:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: is-prime
  namespace: ns1
  annotations:
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /$2
    nginx.ingress.kubernetes.io/backend-protocol: "https"
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - aks.onefoursix.com
    secretName: streamsets-tls
  rules:
  - host: aks.onefoursix.com
    http:
      paths:
      - path: /is-prime(/|$)(.*)
        pathType: ImplementationSpecific
        backend:
          service:
            name: is-prime
            port:
              number: 8000
```

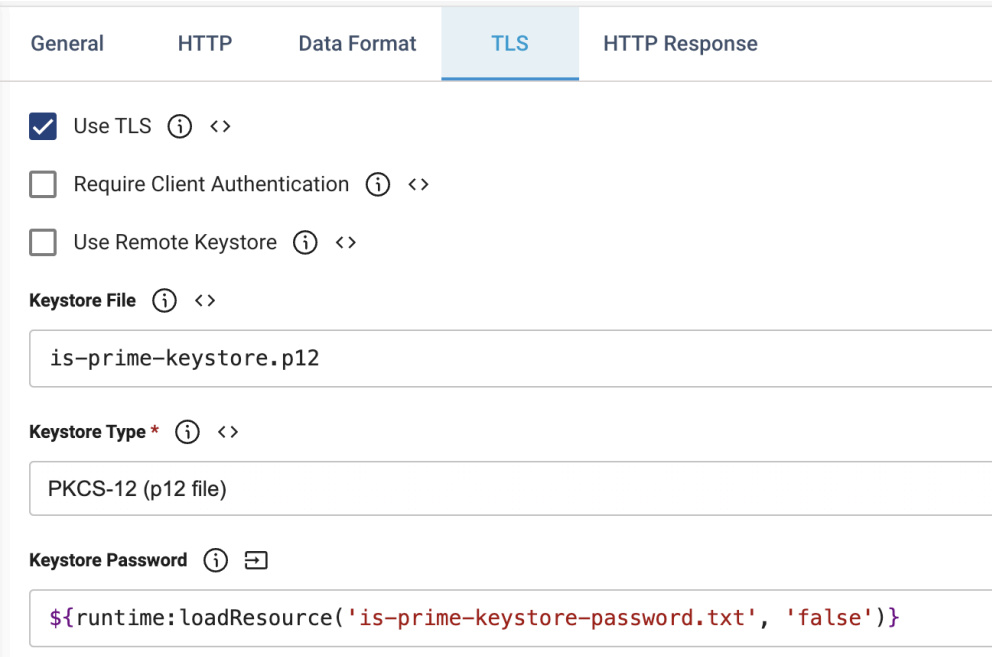
Describe the ingress and make sure it has a valid endpoint:

```
mark@Marks-MacBook-Pro-2 Desktop % kubectl describe ingress is-prime
Name:             is-prime
Labels:           <none>
Namespace:        ns1
Address:          4.154.229.120
Ingress Class:    nginx
Default backend:  <default>
TLS:
  streamsets-tls terminates aks.onefoursix.com
Rules:
  Host               Path  Backends
  ----               ----  --------
  aks.onefoursix.com
                     /is-prime(/|$)(.*)   is-prime:8000 (10.244.1.98:8000)
Annotations:         nginx.ingress.kubernetes.io/auth-tls-verify-client: off
                     nginx.ingress.kubernetes.io/backend-protocol: https
                     nginx.ingress.kubernetes.io/proxy-set-headers: X-Client-Cert $ssl_client_cert
                     nginx.ingress.kubernetes.io/rewrite-target: /$2
                     nginx.ingress.kubernetes.io/ssl-redirect: true
Events:
  Type    Reason  Age                From                      Message
  ----    ------  ----               ----                      -------
  Normal  Sync    58s (x2 over 82s)  nginx-ingress-controller  Scheduled for sync
```

Configure the REST Service's App ID



Configure the REST Service's TLS settings

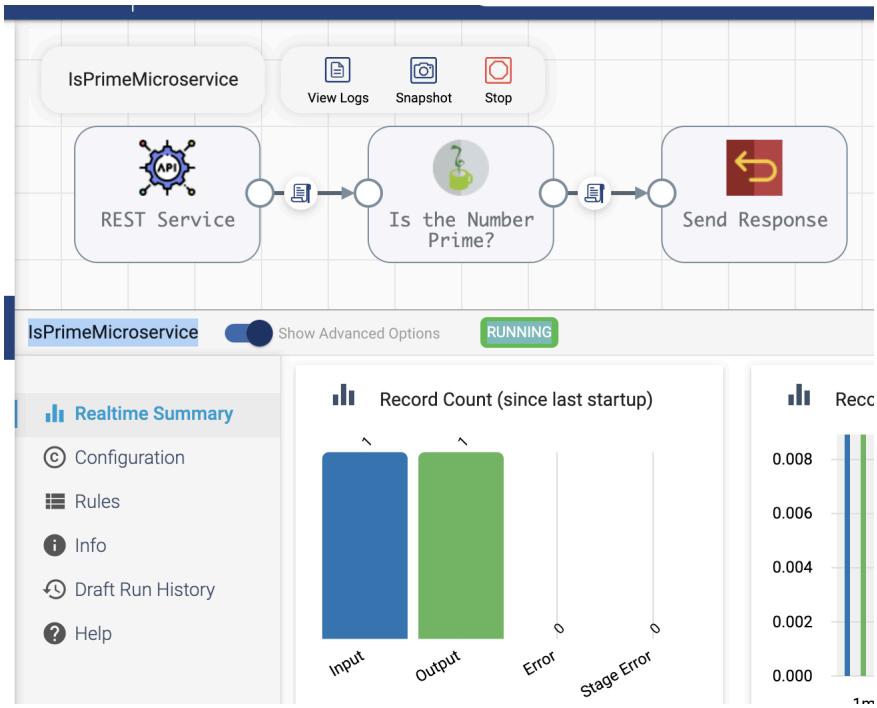


Start the Pipeline and Call the Service

Start the pipeline and call the service like this:

```
curl -v https://<ingress controller host name>/is-prime \
  -H "Content-Type:application/json" \
  -H "X-SDC-APPLICATION-ID:app1" \
  -d '{"number": 11}'
```

You should see the request is handled by the service:



And you should get a response like this:

```
{"httpStatusCode":200,"data":[{"number":11,"is_prime":"true"}],"error":[]}
```