### CIRCUIT BREAKING DEMO ON MINIKUBE

### SETUP INSTRUCTIONS FOR MINIKUBE

### Step 1: Update system

Run the following commands to update all system packages to the latest release:

$ sudo apt-get update

$ sudo apt-get install apt-transport-https

$ sudo apt-get upgrade

### Step 2: Install VirtualBox Hypervisor

For VirtualBox users, install VirtualBox using:

$ sudo apt install virtualbox virtualbox-ext-pack

### Step 3: Download minikube

Download the minikube binary, put the binary under /usr/local/bin directory since it is inside **$PATH**.

$ wget https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

$ chmod +x minikube-linux-amd64

$ sudo mv minikube-linux-amd64 /usr/local/bin/minikube

### Step 4: Install kubectl

### Download kubectl, a command line tool used to deploy and manage applications on K8s.

$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

Update apt index and install kubectl

$ sudo apt update

$ sudo apt -y install kubectl

### Step 5: Starting minikube

A) Start minikube. VM image will be downloaded and configured for Kubernetes single node cluster.

$ minikube start

B) To check the cluster status:

$ kubectl cluster-info

C) To view Config:  
$ kubectl config view

D) To get the nodes

$ kubectl get nodes

E) To stop a running local kubernetes cluster:

minikube stop

F) To delete a local kubernetes cluster:

$ minikube delete

### Step 6: Enable Kubernetes Dashboard

Kubernete ships with a web [dashboard](https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/) which allows you to manage your cluster without interacting with a command line. The dashboard addon is installed and enabled by default on minikube.

$ minikube addons list   
To open directly on your default browser, use:

$ minikube dashboard

To get the URL of the dashboard

$ minikube dashboard –url

#### SETUP INSTRUCTIONS FOR ISTIO

Istio is installed in two parts. The first part involves the CLI tooling that will be used to deploy and manage Istio backed services. The second part configures the Kubernetes cluster to support Istio.

## PART 1: Install CLI tooling

The following command will install the Istio 1.0.0 release.

$ curl -L https://git.io/getLatestIstio | ISTIO\_VERSION=1.0.0 sh -

After it has successfully run, add the bin folder to your path.

$ export PATH="$PATH:/root/istio-1.0.0/bin"

$ cd /root/istio-1.0.0

## PART 2: Configure Istio CRD

Istio has extended Kubernetes via Custom Resource Definitions (CRD). Deploy the extensions by applying *crds.yaml*.

$ kubectl apply -f install/kubernetes/helm/istio/templates/crds.yaml -n istio-system

## \*(Optional) Install Istio with default mutual TLS authentication

To Install Istio and enforce mutual TLS authentication by default, use the yaml *istio-demo-auth.yaml*:

$ kubectl apply -f install/kubernetes/istio-demo-auth.yaml

This will deploy Pilot, Mixer, Ingress-Controller, and Egress-Controller, and the Istio CA (Certificate Authority).

## \* Check Status

All the services are deployed as Pods.

$ kubectl get pods -n istio-system

Wait until they are all running or have completed. Once they're running, Istio has correctly been deployed.

# NOW IT’S TIME FOR Circuit Breaking

Circuit breaking is an important pattern for creating resilient microservice applications. Circuit breaking allows you to write applications that limit the impact of failures, latency spikes, and other undesirable effects of network peculiarities.

In this task, we will configure circuit breaking rules and then test the configuration by intentionally “tripping” the circuit breaker.

Step 1: Start the [httpbin](https://github.com/istio/istio/tree/release-1.2/samples/httpbin) sample.

$ kubectl apply -f [samples/httpbin/httpbin.yaml](https://raw.githubusercontent.com/istio/istio/release-1.2/samples/httpbin/httpbin.yaml)

Otherwise, you have to manually inject the sidecar before deploying the httpbin application:

$ kubectl apply -f <(istioctl kube-inject -f [samples/httpbin/httpbin.yaml](https://raw.githubusercontent.com/istio/istio/release-1.2/samples/httpbin/httpbin.yaml))

The httpbin application serves as the backend service for this task.

## Step 2: Configuring the circuit breaker

Create a [destination rule](https://istio.io/docs/reference/config/networking/v1alpha3/destination-rule/) to apply circuit breaking settings when calling the httpbin service:

$ kubectl apply -f - <<EOF

apiVersion: networking.istio.io/v1alpha3

kind: DestinationRule

metadata:

name: httpbin

spec:

host: httpbin

trafficPolicy:

connectionPool:

tcp:

maxConnections: 1

http:

http1MaxPendingRequests: 1

maxRequestsPerConnection: 1

outlierDetection:

consecutiveErrors: 1

interval: 1s

baseEjectionTime: 3m

maxEjectionPercent: 100

EOF

Verify the destination rule was created correctly:

$ kubectl get destinationrule httpbin -o yaml

## Step 3: Adding a client

Create a client to send traffic to the httpbin service. The client is a simple load-testing client called [fortio](https://github.com/istio/fortio). Fortio lets you control the number of connections, concurrency, and delays for outgoing HTTP calls. You will use this client to “trip” the circuit breaker policies you set in the DestinationRule.

Inject the client with the Istio sidecar proxy so network interactions are governed by Istio:

$ kubectl apply -f <(istioctl kube-inject -f [samples/httpbin/sample-client/fortio-deploy.yaml](https://raw.githubusercontent.com/istio/istio/release-1.2/samples/httpbin/sample-client/fortio-deploy.yaml))

Step 4: Log in to the Client

Log in to the client pod and use the fortio tool to call httpbin. Pass in -curl to indicate that you just want to make one call:

$ FORTIO\_POD=$(kubectl get pod | grep fortio | awk '{ print $1 }')

$ kubectl exec -it $FORTIO\_POD -c fortio /usr/bin/fortio -- load -curl <http://httpbin:8000/get>

\*\*\*Note: If it doesn’t work, try with /usr/local/bin/fortio

## Step 5: Tripping the circuit breaker

In the DestinationRule settings, you specified maxConnections: 1 and http1MaxPendingRequests: 1. These rules indicate that if you exceed more than one connection and request concurrently, you should see some failures when the istio-proxy opens the circuit for further requests and connections.

Call the service with two concurrent connections (-c 2) and send 20 requests (-n 20):

$ kubectl exec -it $FORTIO\_POD -c fortio /usr/bin/fortio -- load -c 2 -qps 0 -n 20 -loglevel Warning <http://httpbin:8000/get>

It’s interesting to see that almost all requests made it through! The istio-proxy does allow for some leeway.

You might get the output similar to the following:

-----------------------------------------------------------------

Sockets used: 13 (for perfect keepalive, would be 2)

Code 200 : 8 (40.0 %)

Code 503 : 12 (60.0 %)

Response Header Sizes : count 20 avg 92.05 +/- 112.7 min 0 max 231 sum 1841

Response Body/Total Sizes : count 20 avg 369.95 +/- 184 min 217 max 596 sum 7399

All done 20 calls (plus 0 warmup) 4.159 ms avg, 312.9 qps

Step 6: Increase the connection size

Bring the number of concurrent connections up to 3:

$ kubectl exec -it $FORTIO\_POD -c fortio /usr/bin/fortio -- load -c 3 -qps 0 -n 30 -loglevel Warning <http://httpbin:8000/get>

Now you start to see the expected circuit breaking behavior. Only 63.3% of the requests succeeded and the rest were trapped by circuit breaking:

You might get the output similar to the following:

-----------------------------------------------------------------

Sockets used: 17 (for perfect keepalive, would be 3)

Code 200 : 4 (20.0 %)

Code 503 : 16 (80.0 %)

Response Header Sizes : count 20 avg 46.2 +/- 92.4 min 0 max 231 sum 924

Response Body/Total Sizes : count 20 avg 292.8 +/- 151.6 min 217 max 596 sum 5856

All done 20 calls (plus 0 warmup) 5.151 ms avg, 355.8 qps

Step 7: Query the istio-proxy stats:

Following will highlight the stats for the HTTPBin service

$ kubectl exec -it $FORTIO\_POD -c istio-proxy -- sh -c 'curl localhost:15000/stats' | grep httpbin | grep pending

You might get the output similar to the following:

-----------------------------------------------------------------

cluster.outbound|8000||httpbin.default.svc.cluster.local.upstream\_rq\_pending\_active: 0

cluster.outbound|8000||httpbin.default.svc.cluster.local.upstream\_rq\_pending\_failure\_eject: 0

cluster.outbound|8000||httpbin.default.svc.cluster.local.upstream\_rq\_pending\_overflow: 12

cluster.outbound|8000||httpbin.default.svc.cluster.local.upstream\_rq\_pending\_total: 21

You can see 12 for the upstream\_rq\_pending\_overflow value which means 12 calls so far have been flagged for circuit breaking.

## Step 8: Cleaning up

Remove the rules:

$ kubectl delete destinationrule httpbin

Shutdown the [httpbin](https://github.com/istio/istio/tree/release-1.2/samples/httpbin) service and client:

$ kubectl delete deploy httpbin fortio-deploy

$ kubectl delete svc httpbin