Install IBM Cloud Kubernetes Service command-line utilities

You must already have created a cluster. Be sure you have an IBM Cloud account.

1. Install the IBM Cloud command-line interface (CLI).
2. Log in to the IBM Cloud CLI. If you have a federated account, include the `--sso` flag:
   `ibmcloud login --sso`
   If you have an an API key, use this command: `ibmcloud login --apikey <your_apikey>`
3. Install the IBM Cloud Kubernetes Service plug-in:
   `ibmcloud plugin install container-service`
4. Verify that the plug-in is installed properly:
   `ibmcloud plugin list`
   The service plug-in is displayed in the results as `container-service/kubernetes-service`.
5. Initialize the service plug-in and point the endpoint to your region. For example when prompted, enter `5` for `us-east`:

```
6. $ ibmcloud cs region-set
7. Choose a region:
8. 1. ap-north
9. 2. ap-south
10.     3. eu-central
11.     4. uk-south
12.     5. us-east
13.     6. us-south
   Enter a number> 5
```

Access your cluster

You need to set the context to work with your cluster by using the kubectl CLI, access the Kubernetes dashboard, and gather basic information about your cluster.

1. Set the context for your cluster in your CLI.
   Every time you log in to the IBM Cloud Kubernetes Service CLI to work with the cluster, you must run these commands to set the path to the cluster's configuration file as a session variable. The Kubernetes CLI uses this variable to find a local configuration file and certificates that are necessary to connect with the cluster in IBM Cloud.
   a. List the available clusters:
   `ibmcloud cs clusters`
   b. Download the configuration file and certificates for your cluster by using the `cluster-config` command:
   `ibmcloud cs cluster-config <your_cluster_name>`
   c. Copy and paste the output command from the previous step to set the `KUBECONFIG` environment variable and configure your CLI to run `kubectl` commands against your cluster. Here's an example:
   `export KUBECONFIG=/Users/user-name/.bluemix/plugins/container-service/clusters/mycluster/kube-config-hou02-mycluster.yml`
2. Get basic information about your cluster and its worker nodes. This information can help you manage your cluster and troubleshoot issues.
   a. View details of your cluster:
   `ibmcloud cs cluster-get <your_cluster_name>`
   b. Verify the worker nodes in the cluster:
   `ibmcloud cs workers <your_cluster_name>`
   `ibmcloud cs worker-get <worker_ID>`
3. Validate access to your cluster:
   a. View nodes in the cluster:
   `kubectl get node`
   b. View services, deployments, and pods:
   `kubectl get svc,deploy,po --all-namespaces`

Clone the lab repo
From the command line, run:
```
git clone https://github.com/IBM/istio101

cd istio101/workshop
```
This is the working directory for the course labs. Use the sample .yaml files that are located in the workshop/plans directory for the labs.

Install Istio on IBM Cloud Kubernetes Service
You must download and install Istio.

1. Either download Istio from https://github.com/istio/istio/releases or get the latest version by using curl:
   `curl -L https://git.io/getLatestIstio | sh -`
2. Extract the installation files:
3. `tar -xvzf istio-<istio-version>-linux.tar.gz`

4. Add the `istioctl` client to your PATH variable. The `<version-number>` is in the directory name. For example, run the following command on a Mac OS or Linux system:
5. `export PATH=$PWD/istio-<version-number>/bin:$PATH`
6. Change the directory to the Istio file location.
7. `cd istio-<version-number>`

8. Install Istio into the `istio-system` namespace in your Kubernetes cluster:
9. `kubectl apply -f $PWD/install/kubernetes/istio-demo.yaml`
10. Ensure that the `istio-*` Kubernetes services are deployed before you continue:
11. `kubectl get svc -n istio-system`

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
grafana	ClusterIP	172.21.44.128	<none>	3000/TCP
istio-citadel	ClusterIP	172.21.62.12	<none>	8060/TCP,9093/TCP
istio-egressgateway	ClusterIP	172.21.115.236	<none>	80/TCP,443/TCP
istio-galley	ClusterIP	172.21.7.201	<none>	443/TCP,9093/TCP
istio-ingressgateway	LoadBalancer	172.21.19.202	169.61.151.162	80:31380/TCP,443:31390/TCP,31400:314
istio-pilot	ClusterIP	172.21.115.9	<none>	15010/TCP,15011/TCP,8080/TCP,9093/TC
istio-policy	ClusterIP	172.21.165.123	<none>	9091/TCP,15004/TCP,9093/TCP
istio-sidecar-injector	ClusterIP	172.21.164.224	<none>	443/TCP
istio-statsd-prom-bridge	ClusterIP	172.21.57.144	<none>	9102/TCP,9125/UDP
istio-telemetry	ClusterIP	172.21.165.71	<none>	9091/TCP,15004/TCP,9093/TCP,42422/TC
jaeger-agent	ClusterIP	None	<none>	5775/UDP,6831/UDP,6832/UDP
jaeger-collector	ClusterIP	172.21.154.138	<none>	14267/TCP,14268/TCP
jaeger-query	ClusterIP	172.21.224.97	<none>	16686/TCP
prometheus	ClusterIP	172.21.173.167	<none>	9090/TCP
servicegraph	ClusterIP	172.21.190.31	<none>	8088/TCP
tracing	ClusterIP	172.21.2.208	<none>	80/TCP
zipkin	ClusterIP	172.21.76.162	<none>	9411/TCP

Note: For Lite clusters, the istio-ingressgateway service will be in pending state with no external ip. This is normal.

12. Ensure the corresponding pods `istio-citadel-*`, `istio-ingressgateway-*`, `istio-pilot-*`, and `istio-policy-*` are all in the Running state before you continue.
13. `kubectl get pods -n istio-system`

```
grafana-85dbf49c94-gccvp                  1/1       Running     0          5d
istio-citadel-545f49c58b-j8tm5            1/1       Running     0          5d
istio-cleanup-secrets-smtxn               0/1       Completed   0          5d
istio-egressgateway-79f4b99d6f-t2lvk      1/1       Running     0          5d
istio-galley-5b6449c48f-sc92j             1/1       Running     0          5d
istio-grafana-post-install-djzm9          0/1       Completed   0          5d
istio-ingressgateway-6894bd895b-tvklg     1/1       Running     0          5d
istio-pilot-cb58b65c9-sj8zb               2/2       Running     0          5d
istio-policy-69cc5c74d5-gz8kt             2/2       Running     0          5d
istio-sidecar-injector-75b9866679-sldhs   1/1       Running     0          5d
istio-statsd-prom-bridge-549d687fd9-hrhfs 1/1       Running     0          5d
istio-telemetry-d8898f9bd-2gl49           2/2       Running     0          5d
istio-telemetry-d8898f9bd-9r9jz           2/2       Running     0          5d
istio-tracing-7596597bd7-tqwkr            1/1       Running     0          5d
prometheus-6ffc56584f-6jqhg               1/1       Running     0          5d
servicegraph-5d64b457b4-z2ctz             1/1       Running     0          5d
```

Before your continue, make sure all the pods are deployed and running. If they're in pending state, wait a few minutes to let the deployment finish.
Congratulations! You successfully installed Istio into your cluster.

Download the Guestbook app and create the Redis database

The Guestbook app is a sample app for users to leave comments. It consists of a web front end, Redis master for storage, and a replicated set of Redis slaves.
You will integrate the Guestbook app with the Watson Tone Analyzer service. This service detects the sentiment in user comments and responds with emoticons. You need to deploy this app to your Kubernetes cluster.

1. Open your preferred terminal and download the Guestbook app from GitHub into the `workshop` directory:
   `git clone https://github.com/IBM/guestbook.git ../guestbook`
2. Change to the app directory:
   `cd ../guestbook/v2`
   Next, you create a Redis database so that you can use to persist the data of your app. The Redis database comes with master and slave modules.
3. Create the Redis controllers and services for both the master and the slave:
4. `kubectl create -f redis-master-deployment.yaml`
5. `kubectl create -f redis-master-service.yaml`
6. `kubectl create -f redis-slave-deployment.yaml`
   `kubectl create -f redis-slave-service.yaml`
7. Verify that the Redis controllers for the master and the slave are created:
   `kubectl get deployment`

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
redis-master	1	1	1	1	5d
redis-slave	2	2	2	2	5d

8. Verify that the Redis services for the master and the slave are created:
9. `kubectl get svc`

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
redis-master	ClusterIP	172.21.85.39	<none>	6379/TCP	5d
redis-slave	ClusterIP	172.21.205.35	<none>	6379/TCP	5d

10. Verify that the Redis pods for the master and the slave are running:
    `kubectl get pods`

NAME	READY	STATUS	RESTARTS	AGE
redis-master-4sswq	1/1	Running	0	5d
redis-slave-kj8jp	1/1	Running	0	5d
redis-slave-nslps	1/1	Running	0	5d

Install the Guestbook app with manual sidecar injection

In Kubernetes, a sidecar is a utility container in the pod, and its purpose is to support the main container. For Istio to work, Envoy proxies must be deployed as sidecars to each pod of the deployment. You can inject the Istio sidecar into a pod in two ways:

- Manually by using the istioctl CLI tool
- Automatically by using the Istio Initializer

In this section, you will use the manual injection. Manual injection modifies the controller configuration, for example, deployment. It does this by modifying the pod template spec such that all pods for that deployment are created with the injected sidecar.

1. Inject the Istio Envoy sidecar into the guestbook pods and deploy the Guestbook app on to the Kubernetes cluster:
2. `kubectl apply -f <(istioctl kube-inject -f ../v1/guestbook-deployment.yaml)`
3. `kubectl apply -f <(istioctl kube-inject -f guestbook-deployment.yaml)`
   There are two versions of deployments: a new version (v2) in the current directory and a previous version (v1) in a sibling directory. They will be used in future labs to showcase the Istio traffic routing capabilities.
4. Create the guestbook service:
   `kubectl create -f guestbook-service.yaml`
5. Verify that the service was created:
6. `kubectl get svc`

```
NAME           TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)
guestbook      LoadBalancer   172.21.36.181   169.61.37.140   80:32149/TCP
```

   For Lite clusters, the external IP will not be available. This is expected.

7. Verify that the pods are running:
8. `kubectl get pods`

```
NAME                         READY     STATUS    RESTARTS   AGE
guestbook-v1-89cd4b7c7-frscs   2/2       Running   0          5d
guestbook-v1-89cd4b7c7-jn224   2/2       Running   0          5d
guestbook-v1-89cd4b7c7-m7hmd   2/2       Running   0          5d
guestbook-v2-56d98b558c-7fvd5  2/2       Running   0          5d
guestbook-v2-56d98b558c-dshkh  2/2       Running   0          5d
guestbook-v2-56d98b558c-mzbxk  2/2       Running   0          5d
```

Each guestbook pod has two containers in it:

- The guestbook container
- The Envoy proxy sidecar

Add the Watson Tone Analyzer

Watson Tone Analyzer detects the tone from the words that users enter in the Guestbook application. A recognized tone is converted to a corresponding emoticon.

1. Create the Watson Tone Analyzer in your account:
   `ibmcloud resource service-instance-create my-tone-analyzer-service tone-analyzer lite us-south`
2. Create the service key for the Tone Analyzer service. This command should output the credentials that you just created. You will need the value for the API key and URL later.
   `ibmcloud resource service-key-create tone-analyzer-key Manager --instance-name my-tone-analyzer-service`
   Tip: If you need to get the service keys later, run this command:
   `ibmcloud resource service-key tone-analyzer-key`
3. Open the `analyzer-deployment.yaml` and find the env section near the end of the file. Replace YOUR_API_KEY with your own API key and replace YOUR_URL with the URL value that you saved before. YOUR_URL should look something like this: `gateway.watsonplatform.net/tone-analyzer/api`. Save the file.
   The Tone Analyzer service uses IBM Cloud Identity and Access Management (IAM) tokens to make authenticated requests to the Tone Analyzer service. IAM authentication uses access

tokens for authentication, which are acquired by sending a request to a URL with an API key. As a result, you need to set up egress rules to allow the Tone Analyzer service access to those external URLs.

4. Apply the egress rules found in the `istio101/workshop/plans` directory:
5. `cd ../../plans`
6. `kubectl apply -f analyzer-egress.yaml`

7. Deploy the analyzer pods and service by using the `analyzer-deployment.yaml` and `analyzer-service.yaml` files found in the `guestbook/v2` directory. The analyzer service talks to Watson Tone Analyzer to help analyze the tone of a message.
8. `cd ../guestbook/v2/`
9. `kubectl apply -f <(istioctl kube-inject -f analyzer-deployment.yaml)`
   `kubectl apply -f analyzer-service.yaml`

Great! Your guestbook app is now running. In the next lab, you'll be able to see the application in action by accessing the service endpoint. You'll also be able to view telemetry data for the application.

Challenges with microservices

Microservice architecture is the perfect fit for cloud native applications, and it increases the delivery velocities greatly. If you have many microservices that are delivered by multiple teams, how do you observe the the overall platform and keep track of each service to find out exactly what is going on? When something goes wrong, how do you know which service or which communication problem among the your services is causing the problem?

Istio telemetry

Istio's tracing and metrics features provide broad and granular insight into the health of all services. Istio's role as a service mesh makes it the ideal data source for observability information, particularly in a microservices environment.

As requests pass through multiple services, identifying performance bottlenecks becomes increasingly difficult using traditional debugging techniques. Distributed tracing provides a holistic view of requests transiting through multiple services, allowing for immediate identification of latency issues.

With Istio, distributed tracing comes by default. This will expose latency, retry, and failure information for each hop in a request.

Read more about how Istio mixer enables telemetry reporting.

Configure Istio to receive telemetry data

1. Verify that the Grafana, Prometheus, ServiceGraph, and Jaeger add-ons were installed successfully. All add-ons are installed into the istio-system namespace:
2. `kubectl get pods -n istio-system`
3. `kubectl get services -n istio-system`
   Next, you'll configure Istio to automatically gather telemetry data for services that run in the service mesh.
4. Go back to the plans directory at `istio101/workshop/plans`:
5. `cd ../../plans`
6. Create a rule to collect telemetry data:
7. `kubectl create -f guestbook-telemetry.yaml`
8. Obtain the guestbook endpoint to access the guestbook from either a paid cluster or a lite cluster.

   Paid cluster

   - Access the guestbook through the external IP for your service as guestbook is deployed as a load balancer service. Find the EXTERNAL-IP of the guestbook service by running this command:
     `kubectl get service guestbook -n default`
   - Go to this external IP address in the browser to try out your guestbook.

   Lite cluster

   - Get the worker's public IP:
     `ibmcloud cs workers <cluster_name>`

ID	Public IP	Private IP	Machine Type	State	Status	Zone	Version
kube-xxx	169.60.87.20	10.188.80.69	u2c.2x4.encrypted	normal	Ready	wdc06	1.9.7_1510*

   - Get the node port:
   - `kubectl get svc guestbook -n default`

```
NAME        TYPE          CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
guestbook   LoadBalancer  172.21.134.6   pending       80:31702/TCP     4d
```

In this example, the node port is 169.60.87.20:31702.

- Go to this address in the browser to try out your guestbook.

9. Generate a small load to the app.
   `while sleep 0.5; do curl http://<guestbook_endpoint/; done`

View guestbook telemetry data

Jaeger

1. Establish port forwarding from local port 16686 to the Tracing instance:
2. `kubectl port-forward -n istio-system \`
3. `  $(kubectl get pod -n istio-system -l app=jaeger -o jsonpath='{.items[0].metadata.name}') \`
   `  16686:16686 &`
4. In your browser, go to http://127.0.0.1:16686.
5. From the **Services** menu, select either the **guestbook** or **analyzer** service.
6. Scroll to the bottom and click **Find Traces** to see traces.

Grafana

1. Establish port forwarding from local port 3000 to the Grafana instance:
2. `kubectl -n istio-system port-forward \`
3. `  $(kubectl -n istio-system get pod -l app=grafana -o jsonpath='{.items[0].metadata.name}') \`
4. `  3000:3000 &`
5. Browse to http://localhost:3000 and navigate to the Istio Mesh Dashboard by clicking the **Home** menu on the top left.

Prometheus

1. Establish port forwarding from local port 9090 to the Prometheus instance:
2. `kubectl -n istio-system port-forward \`
3. `  $(kubectl -n istio-system get pod -l app=prometheus -o jsonpath='{.items[0].metadata.name}') \`
   `  9090:9090`
4. Browse to http://localhost:9090/graph and in the **Expression** input box, enter `istio_request_byte_count`. Then, click **Execute**.

Service Graph

1. Establish port forwarding from local port 8088 to the Service Graph instance:
2. `kubectl -n istio-system port-forward \`
3. `  $(kubectl -n istio-system get pod -l app=servicegraph -o jsonpath='{.items[0].metadata.name}') \`
   `  8088:8088 &`
4. Browse to http://localhost:8088/dotviz.

Connecting the entire trace

Although Istio proxies are able to automatically send spans, they need some hints to tie together the entire trace. Applications need to propagate the appropriate HTTP headers so that when the proxies send span information to Zipkin or Jaeger, the spans can be correlated correctly into a single trace.

In this example, when a user visits the Guestbook app, the HTTP request is sent from the guestbook service to Watson Tone Analyzer. For the individual spans of guestbook service and Watson Tone Analyzer to be tied together, the guestbook service has been modified to extract the required headers (x-request-id, x-b3-traceid, x-b3-spanid, x-b3-parentspanid, x-b3-sampled, x-b3-flags, x-ot-span-context) and forward them onto the Tone Analyzer service when calling the analyzer service from the guestbook service.

The change is in the `v2/guestbook/main.go`. By using the `getForwardHeaders()` method, the required headers can be extracted, and then the required headers are used again when calling the Tone Analyzer service by using the `getPrimaryTone()` method.

Istio Ingress controller

The components deployed on the service mesh by default are not exposed outside the cluster. External access to individual services so far has been provided by creating an external load balancer or node port on each service.

An ingress is a collection of rules that allow inbound connections to reach the cluster services. You can create an Ingress Gateway resource to allow external requests through the Istio Ingress Gateway to the back-end services.

Follow the instructions depending on whether you have a paid cluster or a lite cluster.

Lastly, if you have a paid cluster, you can optionally set up the Istio ingres controller to work with the IBM Cloud Kubernetes Service.

Lite cluster users

Expose the Guestbook app with Ingress if you have lite cluster

1. Configure the guestbook default route with the Istio Ingress Gateway:

   ```
   kubectl create -f guestbook-gateway.yaml
   ```

2. Check the node port of the ingress:

   ```
   kubectl get svc istio-ingressgateway -n istio-system
   ```

   ```
   NAME            TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)                      AGE
   istio-ingress   LoadBalancer                    *              80:31702/TCP,443:32290/TCP   10d
   ```

3. Get the Public IP of your cluster:

   ```
   ibmcloud cs workers <cluster_name>
   ```

   ```
   ID            Public IP      Private IP      Machine Type         State    Status   Zone    Version
   kube-xxx      169.60.87.20   10.188.80.69    u2c.2x4.encrypted    normal   Ready    wdc06   1.9.7_1510*
   ```

   The node port in above sample output is 169.60.87.20:31702.

4. Make note of the IP and node port that you retrieved in the previous step because it will be used to access the Guestbook app in later parts of the course. You can create an environment variable called $INGRESS_IP with your IP address, for example:

   ```
   export INGRESS_IP=169.60.72.58:31702
   ```

Optional: Set up the Istio Ingress controller to work with the IBM Cloud Kubernetes Service

Optional: Set up the Istio Ingress controller to work with the IBM Cloud Kubernetes Service

You must have a standard, or paid, cluster if you want to set up the Istio Ingress controller to work with IBM Cloud Kubernetes Service.

Standard IBM Cloud Kubernetes clusters can expose applications deployed within your cluster by using a Kubernetes Ingress application load balancer (ALB). IBM Cloud Kubernetes Service automatically creates a highly available ALB for your cluster and assigns a unique public route to it in the format `<cluster_name>.<region_or_zone>.containers.appdomain.cloud`.

The Ingress resource provides IBM Cloud users with a secure, reliable, and scalable network stack to distribute incoming network traffic to apps in IBM Cloud. You can enhance the IBM provided Ingress application load balancer by adding annotations. Learn more about Ingress for IBM Cloud Kubernetes Service.

To use this IBM provided DNS for the Guestbook app, you must set the Kubernetes Ingress application load balancer (ALB) to route traffic to the Istio Ingress Gateway.

1. Check the IBM Ingress subdomain information:
2. `ibmcloud cs cluster-get <cluster_name>`

```
Ingress subdomain:      mycluster.us-east.containers.mybluemix.net
```

3. Prepend `guestbook.` to the subdomain that you retrieved in the previous step.

   This new URL will serve as host in the guestbook-frontdoor.yaml file, which you can find and edit in the `istio101/workshop/plans` directory.

   The file should now look something like this:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: guestbook-ingress
  namespace: istio-system
spec:
  rules:
    - host: guestbook.mycluster.us-
east.containers.appdomain.cloud
      http:
        paths:
          - path: /
            backend:
              serviceName: istio-ingressgateway
              servicePort: 80
```

4. Create the Ingress with the IBM provided subdomain:

```
kubectl apply -f guestbook-frontdoor.yaml
```

5. List the details for your Ingress:

```
kubectl get ingress guestbook-ingress -n istio-system -o yaml
```

Example output:

```
  apiVersion: extensions/v1beta1
  kind: Ingress
  metadata:
    annotations:
      kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"extensions/v1beta1","kind":"Ingress","metadata":{
"annotations":{},"name":"guestbook-ingress","namespace":"istio-
system"},"spec":{"rules":[{"host":"guestbook.mycluster.us-
east.containers.appdomain.cloud","http":{"paths":[{"backend":{"s
erviceName":"istio-
ingressgateway","servicePort":80},"path":"/"}]}}]}}
    creationTimestamp: 2018-11-28T19:02:59Z
    generation: 6
    name: guestbook-ingress
    namespace: istio-system
    resourceVersion: "1438905"
    selfLink: /apis/extensions/v1beta1/namespaces/istio-
system/ingresses/guestbook-ingress
    uid: 38f0c0bd-f340-11e8-b97a-4ef94ed74105
  spec:
    rules:
    - host: guestbook.mycluster.us-
east.containers.appdomain.cloud
      http:
        paths:
        - backend:
            serviceName: istio-ingressgateway
            servicePort: 80
          path: /
```

6. Make note of the IBM provided subdomain because it will be used to access your Guestbook app in later parts of the course. Here's an example of the subdomain:
7. `http://guestbook.mycluster.us-east.containers.appdomain.cloud`

Congratulations! You extended the base Ingress features by providing a DNS entry to the Istio serv

Traffic management rules

The core component used for traffic management in Istio is Pilot, which manages and configures all the Envoy proxy instances deployed in a particular Istio service mesh.

Pilot lets you specify what rules you want to use to route traffic between Envoy proxies, which run as sidecars to each service in the mesh.

Each service consists of any number of instances running on pods, containers, VMs, and so on. Each service can have any number of versions, or subsets. There can be distinct subsets of service instances running different variants of the app binary. These variants are not necessarily different API versions. They could be iterative changes to the same service, deployed in different environments (production, staging, development, and so on).

Pilot translates high-level rules into low-level configurations and distributes this configuration to Envoy instances. Pilot uses three types of configuration resources to manage traffic in its service mesh:

- Virtual services
- Destination rules
- Service entries

Virtual services

A VirtualService defines a set of traffic routing rules to apply when a host is addressed. Each routing rule defines matching criteria for traffic of a specific protocol. If the traffic is matched, it is sent to a named destination service (or subset or version of it) defined in the service registry.

Destination rules

A DestinationRule defines policies that apply to traffic that is intended for a service after routing has occurred. These rules specify configuration for load balancing, connection pool size from the sidecar, and outlier detection settings to detect and evict unhealthy hosts from the load balancing pool. Any destination host and subset referenced in a VirtualService rule must be defined in a corresponding DestinationRule.

Service entries

A ServiceEntry configuration enables services in the mesh to access a service that is not necessarily managed by Istio. The rule describes the endpoints, ports, and protocols of a white-listed set of mesh-external domains and IP blocks that services in the mesh are allowed to access.

Guestbook app

The Guestbook app has one service: guestbook. The guestbook service has two distinct versions:

- The base version (version 1)
- The modernized version (version 2)

Each version of the service has three instances based on the number of replicas in the guestbook-deployment.yaml file and the guestbook-v2-deployment.yaml file.

By default, before you create any rules, Istio will route requests equally across version 1 and version 2 of the guestbook service and their respective instances in a round-robin manner. However, new versions of a service can easily introduce bugs to the service mesh, so doing A/B testing and incrementally rolling out changes with canary deployments are good practices.

Perform A/B testing with Istio

A/B testing is a method of performing identical tests against two separate service versions to determine which performs better.

To prevent Istio from performing the default routing behavior between the original and modernized guestbook service, define the following rules, which you can also find in GitHub:

kubectl replace -f virtualservice-all-v1.yaml

Let's examine the rule:

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: virtual-service-guestbook
spec:
  hosts:
    - '*'
  gateways:
    - guestbook-gateway
  http:
    - route:
        - destination:
            host: guestbook
            subset: v1
```

kubectl create -f guestbook-destination.yaml

Let's examine the rule:

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: destination-guestbook
spec:
  host: guestbook
  subsets:
    - name: v1
      labels:
        version: '1.0'
    - name: v2
      labels:
        version: '2.0'
```

The VirtualService defines a rule that captures all HTTP traffic going to host name guestbook and routes 100% of the traffic to pods of the guestbook service with label version: v1. A subset or version of a route destination is identified with a reference to a named service subset that must be declared in a corresponding DesinationRule.

Because three instances match the criteria of host name guestbook and subset version: v1, by default Envoy will send traffic to all three instances in a round-robin manner.

You can view the guestbook service UI by using the IP address and port obtained in "Lab 2: Expose the service mesh with the Istio Ingress Gateway" and enter it as a URL in Firefox or Chrome web browsers.

To enable the Istio service mesh for A/B testing against the new service version, modify the original VirtualService rule:

kubectl replace -f virtualservice-test.yaml

Let's examine the rule:

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: virtual-service-guestbook
spec:
  hosts:
    - '*'
  gateways:
    - guestbook-gateway
  http:
    - match:
        - headers:
            user-agent:
              regex: '.*Firefox.*'
      route:
        - destination:
            host: guestbook
            subset: v2
    - route:
        - destination:
            host: guestbook
            subset: v1
```

In Istio VirtualService rules, there can be only one rule for each service and therefore when defining multiple HTTPRoute blocks, the order in which they are defined in the YAML file matters. Therefore, you should modify the original VirtualService rule rather than creating a new rule. With the modified rule, incoming requests originating from Firefox browsers will go to the newer version of guestbook. All other requests fall through to the next block, which routes all traffic to the original version of guestbook.

In a previous section, you set up some egress rules to allow the guestbook service to call the Watson Tone Analyzer service. By default, Istio blocks calls to services outside the service mesh. For calls to reach the Watson service, you applied the VirtualService and ServiceEntry found in /istio101/workshop/plans/analyzer-egress.yaml file.

The ServiceEntry defines addresses and ports that services within the mesh are allowed to make requests to. If two browsers are available on your system, observe the modernized guestbook service in Firefox and the original guestbook service in any other browser.

Incrementally roll out changes with canary deployments

In canary deployments, newer versions of services are incrementally rolled out to users to minimize the risk and impact of any bugs introduced by the newer version.

To begin incrementally routing traffic to the newer version of the guestbook service, modify the original VirtualService rule:

kubectl replace -f virtualservice-80-20.yaml

Let's examine the rule:

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: virtual-service-guestbook
spec:
  hosts:
    - '*'
  gateways:
    - guestbook-gateway
  http:
    - route:
        - destination:
            host: guestbook
            subset: v1
          weight: 80
        - destination:
            host: guestbook
            subset: v2
          weight: 20
```

In the modified rule, the routed traffic is split between two different subsets of the guestbook service. In this way, traffic to the modernized version 2 of guestbook is controlled on a percentage basis to limit the impact of any unforeseen bugs. This rule can be modified over time until eventually all traffic is directed to the newer version of the service.

You can see this in action by going to the ingress IP address (that you saved in a previous lab) in your browser. Ensure that you are using a hard refresh (command + Shift + R on Mac or Ctrl + F5 on Windows) to remove any browser caching. You should notice that the guestbook should swap between V1 or V2 at about the weight you specified.

Circuit breakers and destination rules

By using Istio Destination rules, you can configure Envoy's implementation of circuit breakers. Circuit breakers are critical for defining the behavior for service-to-service communication in the service mesh. In the event of a failure for a particular service, circuit breakers allow you to set global defaults for failure recovery on a per-service or per-service-version basis or both. You can apply a traffic policy at the top level of the Destination rule to create circuit breaker settings for an entire service, or the rule can be defined at the subset level to create settings for a particular version of a service.

Depending on whether a service handles HTTP requests or TCP connections, Destination rules expose a number of ways for Envoy to limit traffic to a particular service and define failure recovery behavior for services initiating the connection to an unhealthy service.

Related links

Traffic Management

Traffic Management: Deeper Dive

Istio Rules API

Istio V1alpha1 to V1alpha3 Converter Tool

Istio Proxy Debug Tool

Circuit Breaking

Timeouts and Retries

Mutual authentication with Transport Layer Security (mTLS)

Istio can secure the communication between microservices without requiring app code changes. Security is provided by authenticating and encrypting communication paths within the cluster. This is becoming a common security and compliance requirement. Delegating communication security to Istio (as opposed to implementing TLS in each microservice), ensures that your app will be deployed with consistent and manageable security policies.

Citadel is an optional part of Istio's control plane components. When enabled, it provides each Envoy sidecar proxy with a strong (cryptographic) identity in the form of a certificate. Identity is based on the microservice's service account and is independent of its specific network location, such as cluster or current IP address. Envoys then use the certificates to identify each other and establish an authenticated and encrypted communication channel between them.

Citadel is responsible for these tasks:

- Providing each service with an identity representing its role
- Providing a common trust root to allow Envoys to validate and authenticate each other
- Providing a key management system, automating generation, distribution, and rotation of certificates and keys

When an application microservice connects to another microservice, the communication is redirected through the client-side and server-side Envoys. This is the end-to-end communication flow:

- Local TCP connection (that is, localhost, not reaching the "wire") between the app and Envoy (client- and server-side)
- Mutually authenticated and encrypted connection between Envoy proxies

When Envoy proxies establish a connection, they exchange and validate certificates to confirm that each is indeed connected to a valid and expected peer. The established identities can later be used as basis for policy checks, for example, access authorization.

Related links

Basic TLS/SSL Terminology

SSL or TLS Handshake

Testing Mutual TLS

Mutual TLS Authentication

Set up Istio Certificate Authority (CA)

Version 2 of the guestbook application uses an external service (tone analyzer) which is not Istio-enabled. Thus, you will disable mTLS globally and enable it only for communication between internal cluster services in this lab.

1. Ensure Citadel is running. Citadel is Istio's in-cluster Certificate Authority (CA) and is required for generating and managing cryptographic identities in the cluster.

   ```
   kubectl get deployment -l istio=citadel -n istio-system
   ```

   This is the expected output:

   ```
   NAME           DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
   istio-citadel   1         1         1            1           15h
   ```

2. Define the mTLS authentication policy for the Tone Analyzer service:

```
cat <<EOF | istioctl create -f -
apiVersion: authentication.istio.io/v1alpha1
kind: Policy
metadata:
  name: mtls-to-analyzer
  namespace: default
spec:
   targets:
   - name: analyzer
   peers:
   - mtls:
EOF

Created config policy/default/mtls-to-analyzer at revision
3934195
```

17. Confirm the policy was created:

```
kubectl get policies.authentication.istio.io
```

```
NAME              AGE
mtls-to-analyzer  1m
```

19. Enable mTLS from guestbook by using a Destination rule:

```
cat <<EOF | istioctl create -f -
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: route-with-mtls-for-analyzer
  namespace: default
spec:
  host: "analyzer.default.svc.cluster.local"
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
EOF

Created config destination-rule/default/route-with-mtls-for-analyzer at revision 3934279
```

Verify the authenticated connection

If mTLS is working correctly, the Guestbook app should continue to operate as expected without any visible impact. Istio will automatically add and manage the required certificates and private keys.

To confirm their presence in the Envoy containers, follow these steps:

1. Get the name of a guestbook pod. Make sure the pod is running.

   ```
   kubectl get pods -l app=guestbook
   ```

   ```
   NAME                            READY     STATUS    RESTARTS   AGE
   guestbook-v2-784546fbb9-299jz   2/2       Running   0          13h
   guestbook-v2-784546fbb9-hsbnq   2/2       Running   0          13h
   guestbook-v2-784546fbb9-lcxdz   2/2       Running   0          13h
   ```

2. SSH into the Envoy container. Make sure to change the pod name to the corresponding one on your system. This command will execute into istio-proxy container (sidecar) of the pod:

   ```
   kubectl exec -it guestbook-v2-xxxxxxxx -c istio-proxy /bin/bash
   ```

3. Verify that the certificate and keys are present:

   ```
   ls /etc/certs/
   ```

You should see the following (plus some others):

```
cert-chain.pem key.pem root-cert.pem
```

Note that `cert-chain.pem` is Envoy's public certificate (that is, presented to the peer), and `key.pem` is the corresponding private key. The root-cert.pem file is Citadel's root certificate, which is used to verify peer certificates.

4. Exit the container:

```
shell exit
```

Service isolation with the denier adapter

Back-end systems such as access control systems, telemetry capturing systems, quota enforcement systems, billing systems, and so on traditionally directly integrate with services, which creates a hard coupling, and it bakes in specific semantics and usage options.

Istio Mixer provides a generic intermediation layer between app code and infrastructure back ends. Its design moves policy decisions out of the app layer and into configuration instead under operator control. Instead of having app code integrate with specific back ends, the app code instead does a fairly simple integration with Mixer, and Mixer takes responsibility for interfacing with the back-end systems.

Because individual infrastructure back ends each have different interfaces and operational models, Mixer needs custom code to deal with each, and these custom bundles of code are called adapters.

Here are some built-in adapters:

- Denier
- Prometheus
- Memquota (memory quota)
- Stackdriver

In this lab, you'll use the denier adapter.

Related links

Policies, Telemetry, and Istio Mixer

How to write Istio mixer policies

Create a policy that denies access to services

1. Block access to the Guestbook service:

```
kubectl create -f mixer-rule-denial.yaml
```

2. Review the rule:

```
    apiVersion: "config.istio.io/v1alpha2"
    kind: denier
    metadata:
      name: denyall
      namespace: istio-system
    spec:
      status:
          code: 7
          message: Not allowed
      ---
      # The (empty) data handed to denyall at run time
      apiVersion: "config.istio.io/v1alpha2"
      kind: checknothing
      metadata:
        name: denyrequest
        namespace: istio-system
      spec:
      ---
      # The rule that uses denier to deny requests to the
guestbook service
      apiVersion: "config.istio.io/v1alpha2"
      kind: rule
      metadata:
        name: deny-hello-world
        namespace: istio-system
      spec:
        match:
destination.service=="guestbook.default.svc.cluster.local"
        actions:
        - handler: denyall.denier
          instances:
         - denyrequest.checknothing
```

32. Verify that the service is denied.

    In a previous lab, you created the Ingress resource. Ensure that the $INGRESS_IP environment variable is still present. Then, in a terminal, run this command:

    ```
    curl http://$INGRESS_IP/
    ```

    You should see the error message

    ```
    PERMISSION_DENIED:denyall.denier.istio-system:Not allowed.
    ```

33. Remove the rule:

    ```
    kubectl delete -f mixer-rule-denial.yaml
    ```

DOCUMENTS

Tasks

Install Tools

Install and Set Up kubectl

Install Minikube

Administer a Cluster
Administration with kubeadm

Certificate Management with kubeadm

Upgrading kubeadm clusters

Adding Windows nodes

Upgrading Windows nodes

Manage Memory, CPU, and API Resources

Configure Default Memory Requests and Limits for a Namespace

Configure Default CPU Requests and Limits for a Namespace

Configure Minimum and Maximum Memory Constraints for a Namespace

Configure Minimum and Maximum CPU Constraints for a Namespace

Configure Memory and CPU Quotas for a Namespace

Configure a Pod Quota for a Namespace

Install a Network Policy Provider

Use Calico for NetworkPolicy

Use Cilium for NetworkPolicy

Use Kube-router for NetworkPolicy

Romana for NetworkPolicy

Weave Net for NetworkPolicy

Access Clusters Using the Kubernetes API

Access Services Running on Clusters

Advertise Extended Resources for a Node

Autoscale the DNS Service in a Cluster

Change the default StorageClass

Change the Reclaim Policy of a PersistentVolume

Cloud Controller Manager Administration

Cluster Management

Configure Multiple Schedulers

Configure Out of Resource Handling

Configure Quotas for API Objects

Control CPU Management Policies on the Node

Control Topology Management Policies on a node

Customizing DNS Service

Debugging DNS Resolution

Declare Network Policy

Developing Cloud Controller Manager

Enabling EndpointSlices

Enabling Service Topology

Encrypting Secret Data at Rest

Guaranteed Scheduling For Critical Add-On Pods

IP Masquerade Agent User Guide

Limit Storage Consumption

Namespaces Walkthrough

Operating etcd clusters for Kubernetes

Reconfigure a Node's Kubelet in a Live Cluster

Reserve Compute Resources for System Daemons

Safely Drain a Node while Respecting the PodDisruptionBudget

Securing a Cluster

Set Kubelet parameters via a config file

Set up High-Availability Kubernetes Masters

Share a Cluster with Namespaces

Using a KMS provider for data encryption

Using CoreDNS for Service Discovery

Using NodeLocal DNSCache in Kubernetes clusters

Using sysctls in a Kubernetes Cluster

Configure Pods and Containers

Assign Memory Resources to Containers and Pods

Assign CPU Resources to Containers and Pods

Configure GMSA for Windows Pods and containers

Configure RunAsUserName for Windows pods and containers

Configure Quality of Service for Pods

Assign Extended Resources to a Container

Configure a Pod to Use a Volume for Storage

Configure a Pod to Use a PersistentVolume for Storage

Configure a Pod to Use a Projected Volume for Storage

Configure a Security Context for a Pod or Container

Configure Service Accounts for Pods

Pull an Image from a Private Registry

Configure Liveness, Readiness and Startup Probes

Assign Pods to Nodes

Assign Pods to Nodes using Node Affinity

Configure Pod Initialization

Attach Handlers to Container Lifecycle Events

Configure a Pod to Use a ConfigMap

Share Process Namespace between Containers in a Pod

Create static Pods

Translate a Docker Compose File to Kubernetes Resources

Setup Konnectivity Service

Set up Konnectivity service

Manage Kubernetes Objects

Declarative Management of Kubernetes Objects Using Configuration Files

Declarative Management of Kubernetes Objects Using Kustomize

Managing Kubernetes Objects Using Imperative Commands

Imperative Management of Kubernetes Objects Using Configuration Files

Inject Data Into Applications

Define a Command and Arguments for a Container

Define Environment Variables for a Container

Expose Pod Information to Containers Through Environment Variables

Expose Pod Information to Containers Through Files

Distribute Credentials Securely Using Secrets

Inject Information into Pods Using a PodPreset

Run Applications

Run a Stateless Application Using a Deployment

Run a Single-Instance Stateful Application

Run a Replicated Stateful Application

Update API Objects in Place Using kubectl patch

Scale a StatefulSet

Delete a StatefulSet

Force Delete StatefulSet Pods

Horizontal Pod Autoscaler

Horizontal Pod Autoscaler Walkthrough

Specifying a Disruption Budget for your Application

Run Jobs

Running Automated Tasks with a CronJob

Parallel Processing using Expansions

Coarse Parallel Processing Using a Work Queue

Fine Parallel Processing Using a Work Queue

Access Applications in a Cluster

Web UI (Dashboard)

Accessing Clusters

Configure Access to Multiple Clusters

Use Port Forwarding to Access Applications in a Cluster

Use a Service to Access an Application in a Cluster

Connect a Front End to a Back End Using a Service

Create an External Load Balancer

Configure Your Cloud Provider's Firewalls

List All Container Images Running in a Cluster

Set up Ingress on Minikube with the NGINX Ingress Controller

Communicate Between Containers in the Same Pod Using a Shared Volume

Configure DNS for a Cluster

Monitoring, Logging, and Debugging

Application Introspection and Debugging

Auditing

Auditing with Falco

Debug a StatefulSet

Debug Init Containers

Debug Pods and ReplicationControllers

Debug Running Pods

Debug Services

Debugging Kubernetes nodes with crictl

Determine the Reason for Pod Failure

Developing and debugging services locally

Events in Stackdriver

Get a Shell to a Running Container

Logging Using Elasticsearch and Kibana

Logging Using Stackdriver

Monitor Node Health

Resource metrics pipeline

Tools for Monitoring Resources

Troubleshoot Applications

Troubleshoot Clusters

Troubleshooting

Extend Kubernetes

Configure the Aggregation Layer

Use Custom Resources

Extend the Kubernetes API with CustomResourceDefinitions

Versions in CustomResourceDefinitions

Setup an Extension API Server

Use an HTTP Proxy to Access the Kubernetes API

TLS

Certificate Rotation

Manage TLS Certificates in a Cluster

Manage Cluster Daemons

Perform a Rolling Update on a DaemonSet

Perform a Rollback on a DaemonSet

Install Service Catalog

Install Service Catalog using Helm

Install Service Catalog using SC

Network

Validate IPv4/IPv6 dual-stack

Extend kubectl with plugins

Manage HugePages

Schedule GPUs

Edit This Page

Install and Set Up kubectl

The Kubernetes command-line tool, kubectl, allows you to run commands against Kubernetes clusters. You can use kubectl to deploy applications, inspect and manage cluster resources, and view logs. For a complete list of kubectl operations, see Overview of kubectl.

- Before you begin
- Install kubectl on Linux
- Install kubectl on macOS
- Install kubectl on Windows
- Download as part of the Google Cloud SDK

- Verifying kubectl configuration
- Optional kubectl configurations
- What's next

Before you begin

You must use a kubectl version that is within one minor version difference of your cluster. For example, a v1.2 client should work with v1.1, v1.2, and v1.3 master. Using the latest version of kubectl helps avoid unforeseen issues.

Install kubectl on Linux

Install kubectl binary with curl on Linux

1. Download the latest release with the command:
2. `curl -LO https://storage.googleapis.com/kubernetes-release/release/`curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt`/bin/linux/amd64/kubectl`
   To download a specific version, replace the `$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)` portion of the command with the specific version.
   For example, to download version v1.18.0 on Linux, type:
   `curl -LO https://storage.googleapis.com/kubernetes-release/release/v1.18.0/bin/linux/amd64/kubectl`
3. Make the kubectl binary executable.
4. `chmod +x ./kubectl`
5. Move the binary in to your PATH.
6. `sudo mv ./kubectl /usr/local/bin/kubectl`
7. Test to ensure the version you installed is up-to-date:
8. `kubectl version --client`

Install using native package management

- Ubuntu, Debian or HypriotOS
- CentOS, RHEL or Fedora

```
sudo apt-get update && sudo apt-get install -y apt-transport-https gnupg2
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/kubernetes.list
sudo apt-get update
sudo apt-get install -y kubectl
```

Install using other package management

- Snap
- Homebrew

If you are on Ubuntu or another Linux distribution that support snap package manager, kubectl is available as a snap application.

```
snap install kubectl --classic
kubectl version -client
```

If you are on Linux and using Homebrew package manager, kubectl is available for installation.

```
brew install kubectl
kubectl version -client
```

Install kubectl on macOS

Install kubectl binary with curl on macOS

1. Download the latest release:
2. `curl -LO "https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/darwin/amd64/kubectl"`
   To download a specific version, replace the `$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)` portion of the command with the specific version.
   For example, to download version v1.18.0 on macOS, type:

```
curl -LO https://storage.googleapis.com/kubernetes-
release/release/v1.18.0/bin/darwin/amd64/kubectl
```

3. Make the kubectl binary executable.
4. `chmod +x ./kubectl`
5. Move the binary in to your PATH.
6. `sudo mv ./kubectl /usr/local/bin/kubectl`
7. Test to ensure the version you installed is up-to-date:
8. `kubectl version --client`

Install with Homebrew on macOS

If you are on macOS and using Homebrew package manager, you can install kubectl with Homebrew.

1. Run the installation command:
2. `brew install kubectl`
   or
   `brew install kubernetes-cli`
3. Test to ensure the version you installed is up-to-date:
4. `kubectl version --client`

Install with Macports on macOS

If you are on macOS and using Macports package manager, you can install kubectl with Macports.

1. Run the installation command:
2. `sudo port selfupdate`
3. `sudo port install kubectl`
4. Test to ensure the version you installed is up-to-date:
5. `kubectl version --client`

Install kubectl on Windows

Install kubectl binary with curl on Windows

1. Download the latest release v1.18.0 from this link.
   Or if you have `curl` installed, use this command:
   ```
   curl -LO https://storage.googleapis.com/kubernetes-
   release/release/v1.18.0/bin/windows/amd64/kubectl.exe
   ```
   To find out the latest stable version (for example, for scripting), take a look at https://storage.googleapis.com/kubernetes-release/release/stable.txt.
2. Add the binary in to your PATH.
3. Test to ensure the version of `kubectl` is the same as downloaded:
4. `kubectl version --client`

Note: Docker Desktop for Windows adds its own version of `kubectl` to PATH. If you have installed Docker Desktop before, you may need to place your PATH entry before the one added by the Docker Desktop installer or remove the Docker Desktop's `kubectl`.

Install with Powershell from PSGallery

If you are on Windows and using Powershell Gallery package manager, you can install and update kubectl with Powershell.

1. Run the installation commands (making sure to specify a `DownloadLocation`):
2. `Install-Script -Name install-kubectl -Scope CurrentUser -Force`
3. `install-kubectl.ps1 [-DownloadLocation <path>]`

Note: If you do not specify a `DownloadLocation`, `kubectl` will be installed in the user's temp Directory.

The installer creates `$HOME/.kube` and instructs it to create a config file.

2. Test to ensure the version you installed is up-to-date:
3. `kubectl version --client`

Note: Updating the installation is performed by rerunning the two commands listed in step 1.

Install on Windows using Chocolatey or Scoop

1. To install kubectl on Windows you can use either Chocolatey package manager or Scoop command-line installer.

- choco
- scoop

```
choco install kubernetes-cli
```

2. Test to ensure the version you installed is up-to-date:
3. `kubectl version --client`
4. Navigate to your home directory:
5. `cd %USERPROFILE%`
6. Create the `.kube` directory:

7. `mkdir .kube`
8. Change to the `.kube` directory you just created:
9. `cd .kube`
10. Configure kubectl to use a remote Kubernetes cluster:
11. `New-Item config -type file`

Note: Edit the config file with a text editor of your choice, such as Notepad.

Download as part of the Google Cloud SDK

You can install kubectl as part of the Google Cloud SDK.

1. Install the Google Cloud SDK.
2. Run the `kubectl` installation command:
3. `gcloud components install kubectl`
4. Test to ensure the version you installed is up-to-date:
5. `kubectl version --client`

Verifying kubectl configuration

In order for kubectl to find and access a Kubernetes cluster, it needs a kubeconfig file, which is created automatically when you create a cluster using kube-up.sh or successfully deploy a Minikube cluster. By default, kubectl configuration is located at `~/.kube/config`.

Check that kubectl is properly configured by getting the cluster state:

```
kubectl cluster-info
```

If you see a URL response, kubectl is correctly configured to access your cluster.

If you see a message similar to the following, kubectl is not configured correctly or is not able to connect to a Kubernetes cluster.

```
The connection to the server <server-name:port> was refused - did you specify the right host or port?
```

For example, if you are intending to run a Kubernetes cluster on your laptop (locally), you will need a tool like Minikube to be installed first and then re-run the commands stated above.

If kubectl cluster-info returns the url response but you can't access your cluster, to check whether it is configured properly, use:

```
kubectl cluster-info dump
```

Optional kubectl configurations

Enabling shell autocompletion

kubectl provides autocompletion support for Bash and Zsh, which can save you a lot of typing.

Below are the procedures to set up autocompletion for Bash (including the difference between Linux and macOS) and Zsh.

- Bash on Linux
- Bash on macOS
- Zsh

Introduction

The kubectl completion script for Bash can be generated with the command `kubectl completion bash`. Sourcing the completion script in your shell enables kubectl autocompletion.

However, the completion script depends on **bash-completion**, which means that you have to install this software first (you can test if you have bash-completion already installed by running `type _init_completion`).

Install bash-completion

bash-completion is provided by many package managers (see here). You can install it with `apt-get install bash-completion` or `yum install bash-completion`, etc.

The above commands create `/usr/share/bash-completion/bash_completion`, which is the main script of bash-completion. Depending on your package manager, you have to manually source this file in your `~/.bashrc` file.

To find out, reload your shell and run `type _init_completion`. If the command succeeds, you're already set, otherwise add the following to your `~/.bashrc` file:

```
source /usr/share/bash-completion/bash_completion
```

Reload your shell and verify that bash-completion is correctly installed by typing `type _init_completion`.

Enable kubectl autocompletion

You now need to ensure that the kubectl completion script gets sourced in all your shell sessions. There are two ways in which you can do this:

- Source the completion script in your `~/.bashrc` file:
- `echo 'source <(kubectl completion bash)' >>~/.bashrc`

- Add the completion script to the `/etc/bash_completion.d` directory:
- `kubectl completion bash >/etc/bash_completion.d/kubectl`

If you have an alias for kubectl, you can extend shell completion to work with that alias:

```
echo 'alias k=kubectl' >>~/.bashrc
echo 'complete -F __start_kubectl k' >>~/.bashrc
```

Note: bash-completion sources all completion scripts in `/etc/bash_completion.d`.
Both approaches are equivalent. After reloading your shell, kubectl autocompletion should be working.