


# **Journey to Kubernetes Security**

# Introduction

Kubernetes is an open source tool that can be extremely useful when orchestrating your containerized application running across a cluster of machines. But while kubernetes automates many of the tedious tasks required to deploy containerized apps, one critical thing that it doesn't manage in most respects is security. Kubernetes is not a container security tool, and it would be a big mistake to assume that Kubernetes can defend your containerized app from cyber attacks.

During this lab, we will learn how to secure you containerized application using different tools and techniques.

 While some of the tools you will be seeing below are specific to IBM Cloud, similar tools are also available on other cloud providers. The links to these and different open source tools will be provided. The purpose of this lab is to learn the concepts while showing a demo on IBM Cloud.

## Prerequisites

- Sign up for IBM Cloud Account and create a cluster
- Install CLI tools and get access to the cluster from command line
- Clone github repository

Once you have the prerequisites set up, you can follow along with any of the labs below.



**Secure your container image using  
Vulnerability Advisor**

[/step-2-secure-your-container-image-  
using-vulnerability-advisor](#)



**Protect sensitive information using KMS**

[/protect-sensitive-information-using-kms](#)

→ **Secure service to service communication using Istio**

/secure-service-to-service-communication-using-istio

→ **Setup runtime security for the cluster using Sysdig Falco**


/setup-runtime-security-for-the-cluster-using-sysdig-falco

→ **Conclusion**


/conclusion

# Prerequisites

## Sign up for IBM Cloud Account

-  For creating a cluster and other services like Key Protect on IBM Cloud, you will either need to create a Standard IBM Cloud Account or a promo code to upgrade an existing lite account.

You can create your account here: [http://ibm.biz/kubernetes\\_security\\_webinar](http://ibm.biz/kubernetes_security_webinar)

-  If you already have an IBM Cloud account that was created recently, you can log into that account.

Once you've filled out the form, you should see the following confirmation screen:



## Thanks!

To complete your registration, check your email.

Can't find the email? [Resend](#)

Cookie Preferences

Next, **click on the confirmation link** in the email you received from IBM to activate your cloud account.

Once you have activated your account, login to the account and you will be ready to proceed with the lab.

---

## Create Kubernetes cluster on IBM Cloud

You can create the cluster using only a Standard IBM Cloud Account. However for the purpose of this webinar, we have created few clusters that you can get access to using the following steps.

Navigate [here](#) and enter the following :

**Lab Key:** Provided during the webinar

**Your IBMid:** The email address you used while creating the IBM Cloud Account.

# Welcome to an IBM Cloud lab

Use this form to get access to a lab environment

**Lab Key (provided by the host)**



ask your host for the key

**Your IBMid**



john@acme.com

☐

I agree to the [terms and conditions](#)

Submit

Click on submit and login to your IBM Cloud Account [here](#).

Make sure to select IBM Account .

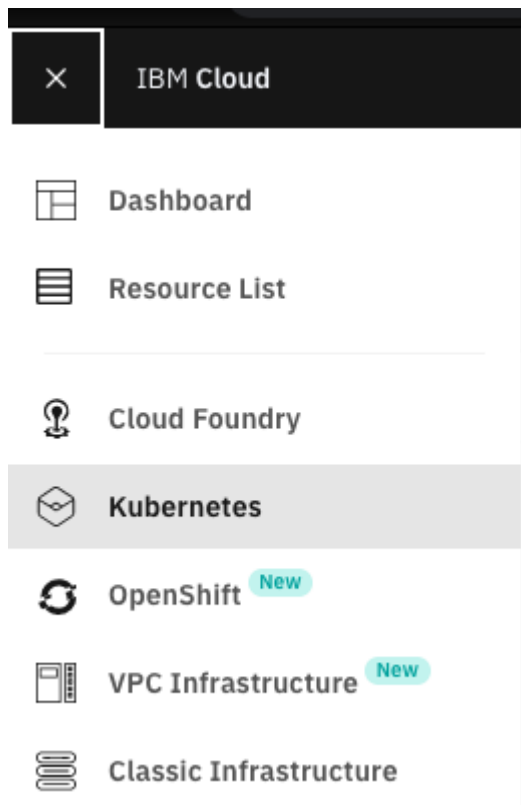
Manage



1840867 - ...




Click on the hamburger menu and navigate to Kubernetes to see your cluster.



## Install CLI Tools

- Lets install CLI tools for IBM cloud with Kubernetes plugin.

 On Windows OS, run the commands in Powershell.

Linux/MacOS

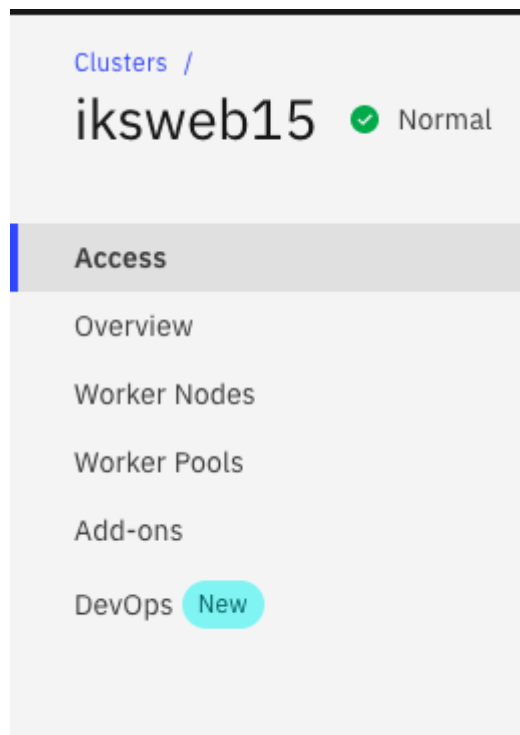
```
curl -sL https://ibm.biz/ibt-installer | bash
```

```
Set-ExecutionPolicy Unrestricted; iex(New-Object Net.WebClient).Download
```

- To work on the lab for Vulnerability Advisor, you will also need [Docker CLI](#).
- To work on the lab for Secure communication using Istio, you will also need [Istio CLI](#)

## Access the cluster from your terminal

Once the cluster has been created, you will need access to it from your local machine to create different services and policies. You can get also see the steps to access you cluster in the Access Tab



- Log in to your IBM Cloud account. Include the `--sso` option if using a federated ID. Select the IBM Cloud Account `1840867` when prompted.



```
ibmcloud login -a cloud.ibm.com -r <your-region> -g Default
```

- Download the kubeconfig files for your cluster. Replace cluster-ID with your own Cluster ID that you can get from the Overview tab in the cluster.

```
ibmcloud ks cluster config --cluster <cluster-ID>
```

- Set the KUBECONFIG environment variable. Copy the output from the previous command and paste it in your terminal. The command output looks similar to the following example:

```
export KUBECONFIG=/Users/$USER/.bluemix/plugins/container-service/clusters/b
```

- Verify that you can connect to your cluster.

```
kubectl version --short
```

---

## Clone the Github Repository

Clone the following repository to your local machine:



Chandni97/SecureCube

<https://github.com/Chandni97/SecureCube>

Once you have the prerequisites set up, you can follow along with any of the labs ahead.

# Secure your container image using Vulnerability Advisor



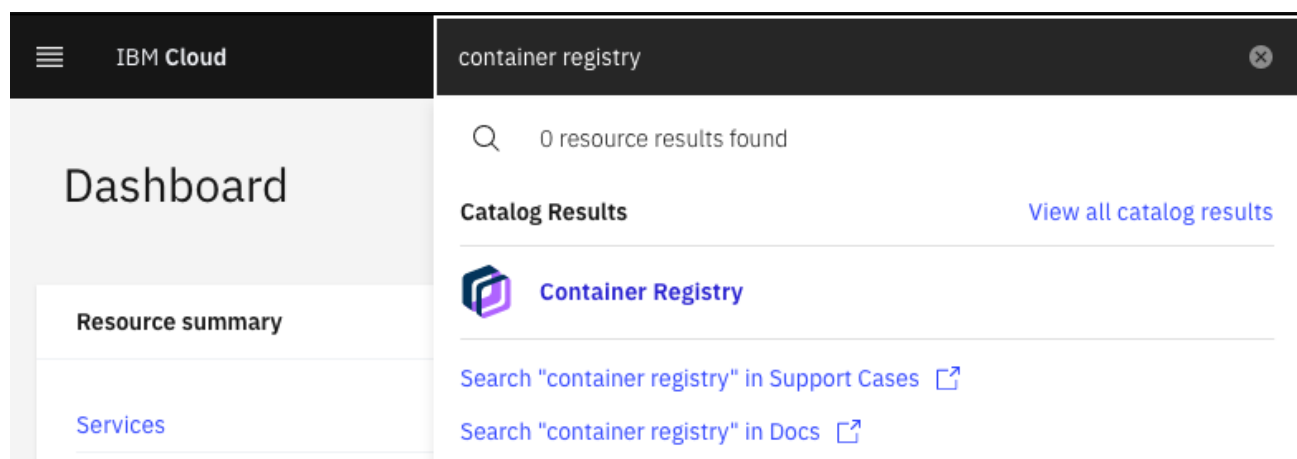
Prerequisite:

- [IBM Cloud Account](#)
- [IBM Cloud CLI](#)
- [Docker CLI](#)
- [Github Repository cloned](#)

Vulnerability Advisor scans your container images for vulnerabilities and misconfiguration. It is integrated with [IBM Container Registry](#) and hence it automatically scans the images as soon as you push them to the registry.

## Step 1: Create Container Registry

First let's search for container registry in the dashboard.



Click on **Container Registry**

On the service creation page click the blue **Create** button:

# Create

## Step 2: Create a namespace

As mentioned in the Quick Start tab, you will need to install Container Registry Plug-in. You can do this easily from your terminal.

```
ibmcloud plugin install container-registry -r 'IBM Cloud'
```

Next, ensure that you're targeting the correct IBM Cloud Container Registry region. Replace the <region> to your region which you can find in step 5 of the Quick Start tab.

```
ibmcloud cr region-set <region>
```

We will now create a namespace where will later push our image.

```
ibmcloud cr namespace-add <my_namespace>
```

## Step 3: Build and push a vulnerable image

Before running the following commands, make sure to change directory to **registry-vulnerable-workflow** inside of the cloned github repository.

Log your local Docker daemon into IBM Cloud Container Registry by running the following command:

```
ibmcloud cr login
```

We will now build and push a vulnerable image to the registry. It is a Debian base image is used, and the `apt` package is rolled back to a version that is vulnerable to remote code.

```
docker build -t us.icr.io/<my_namespace>/hello-world -f Dockerfile-vulnerabl
```

```
docker push us.icr.io/<my_namespace>/hello-world
```

### Step 4: View the vulnerability report for your image

Once pushed, go to the Images tab in your container registry service and click on the image hello-world.

Navigate to Issues by Type to look at the different issues detected by Vulnerability Advisor in your image. It will give information about the vulnerability as will also show how to resolve it.

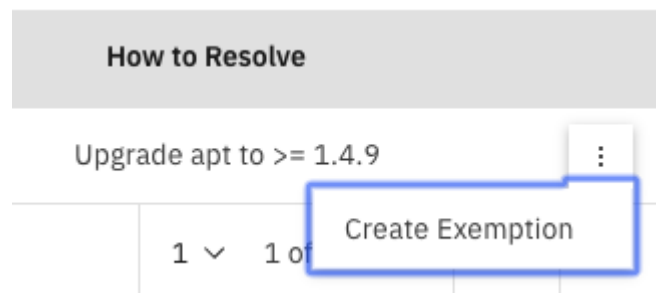
Vulnerabilities

Vulnerability Advisor checks your images for known vulnerabilities based on official community maintained lists. [Learn more](#)

Filter table


Vulnerability ID	Policy Status	Affected Packages	How to Resolve
▼ CVE-2019-3462	🔴 Active	apt	Upgrade apt to >= 1.4.9

You can also create an exemption if for some reason your organization needs to proceed with the image without resolving the issue.



Vulnerability is just one tool that can be used for container image security. There are other tools such as [Twistlock](#), [CoreOS Clair](#), [Dagda](#), [Anchore](#), etc that can continuously scan your image for possible vulnerabilities and alert you when developers are not using secure images

# Protect sensitive information using KMS

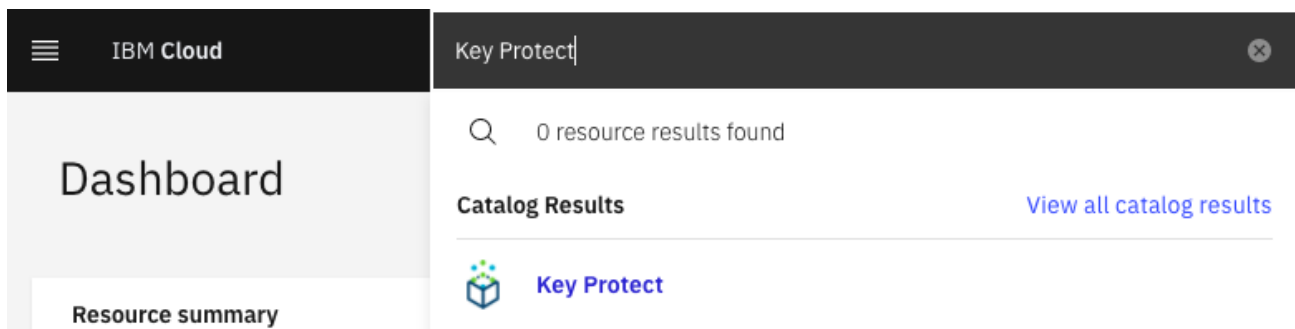
 Prerequisites : [IBM Cloud Account](#).

[IBM Key Protect](#) which is a [Key Management Service](#) helps secure your sensitive data from unauthorized access. It provides mandatory control of user access requests to encryption keys and manages the entire lifecycle of keys. Key Protect provisions and stores cryptographic keys using FIPS 140-2 Level 3 certified (Federal Information Processing Standard) hardware security module (HSM) devices located within secure IBM data centers.

We will use IBM Key protect to automatically encrypt data in etcd, existing cluster secrets, and new secrets that are created in the cluster.

## Step 1: Create Key Protect Service

First let's search for Key Protect in the dashboard.



Click on **Key Protect**

On the service creation page fill up the required details and click the blue **Create** button:

# Create

## Step 2: Create a root key

Once the service is created, on the Manage tab, click on the Add Key button to create the root key.

Add a new key

Create encryption keys that are protected by cloud-based hardware security modules, or import your own keys.

☒ Create a key ☐ Import your own key

Key type  
New to Key Protect? [Learn more](#)

Root key

Name  
test-key1

CancelCreate key

You can also [import your own root key](#) to be managed by Key protect.

## Step 3: Enable KMS for your cluster





Ensure that you have the [Administrator IBM Cloud IAM platform role](#) for the cluster.

We will now go ahead and enable KMS for our cluster which will use the created Key Protect service to encrypt and protect sensitive information in your cluster.

Navigate to your cluster and click on the overview tab. Click on Enable for Key Management Service

Key management service

Enable

Select the root key previously created and enable the service.

## Enable secret encryption



Encrypt your cluster's secrets with a [key management service \(KMS\)](#) root key.



### Important:

Do not delete your KMS instance or root key. If you delete a root key that a cluster uses, the cluster becomes unusable, loses all its data, and cannot be recovered.

Key management service instance

Key Protect-c4



Root key

test-key



Cancel

Enable

Once enabled, your information will be encrypted and you can also [verify that your secrets are encrypted](#).

Most of the major cloud providers such as [GCP](#), [AWS](#) and [Azure](#) offer Key Management Service that can be used to secure information in your cloud architecture.

# Secure service to service communication using Istio



## Prerequisites :

- [IBM Cloud Account](#)
- [Kubernetes Cluster](#)
- [Istio CTL](#)
- [Github Repo Cloned](#)
- [Access to the cluster](#)

One of the main challenges of managing microservices based solutions is how to properly secure not just the microservices themselves but also the communication between them. We will use [Istio](#) that can help us secure this communication without making changes to the application code itself.

We will first need to enable Istio for our cluster running on IBM Cloud. IBM provides managed Istio which allows seamless installation of Istio with other benefits such as automatic updates and lifecycle management of different components.

To enable Istio, navigate to your cluster and click on the Add-ons tab.

You will then see a tile for Managed Istio. Click on install to install istio in your cluster.



### Managed Istio

Install

Istio is a service mesh for providing a uniform way to integrate microservices, manage traffic flow across microservices, enforce policies and aggregate telemetry data. The Istio add-on simplifies the installation and maintenance of your istio control plane so you can focus on managing your microservices. [Learn More.](#)

Wait for the installation to be ready. You will see Add on ready status as shown below



Managed Istio v1.4 ✔ Normal - Addon Ready

:

Istio is a service mesh for providing a uniform way to integrate microservices, manage traffic flow across microservices, enforce policies and aggregate telemetry data. The Istio add-on simplifies the installation and maintenance of your istio control plane so you can focus on managing your microservices. [Learn More.](#)

Once the add on is ready, Setup default configuration profile for Istio by running the following command in your terminal

```
istioctl manifest apply
```

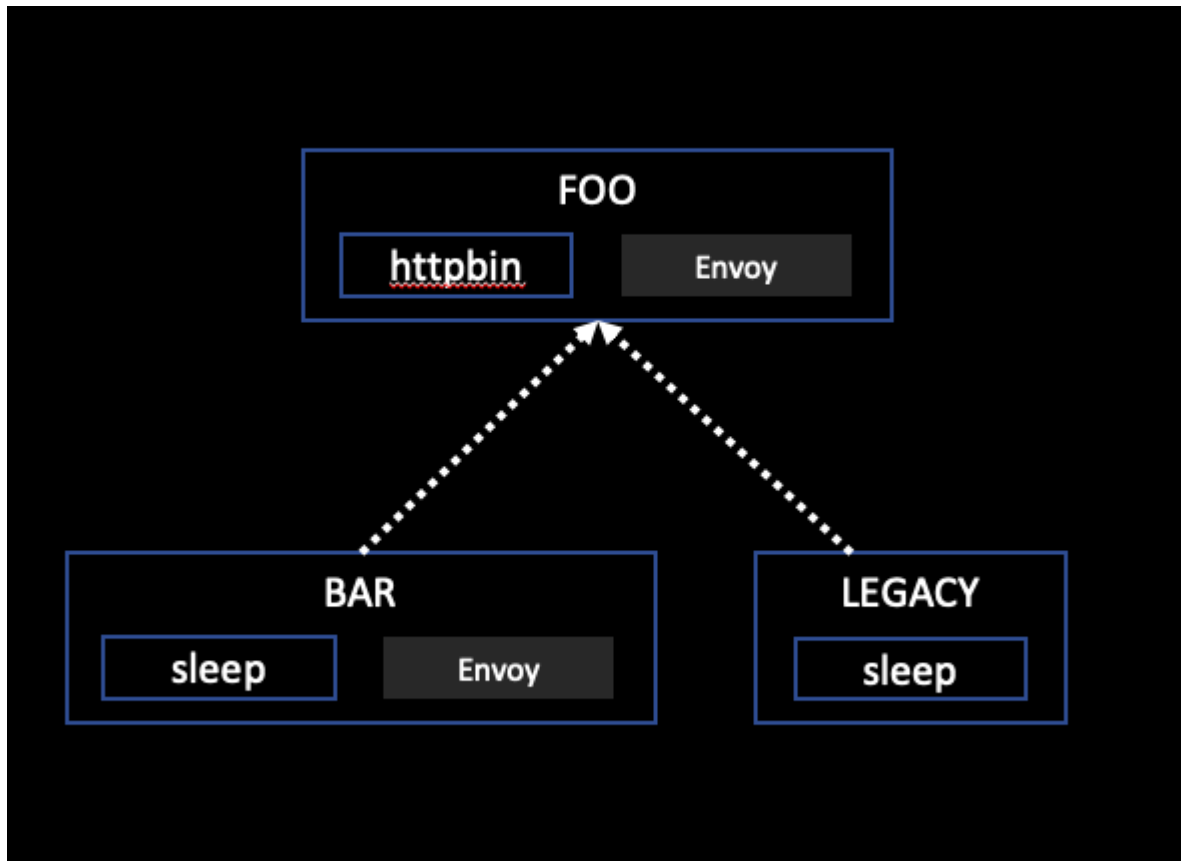
## Authentication

Istio offers [mutual TLS](#) as a full stack solution for transport authentication, which can be enabled without requiring service code changes. This solution:

- Provides each service with a strong identity representing its role to enable interoperability across clusters and clouds.
- Secures service-to-service communication.
- Provides a key management system to automate key and certificate generation, distribution, and rotation.

We will now create few services in our cluster and see how authentication works when we enable Istio.

To see how authentication works we will create 3 namespaces (foo, bar and legacy) with different services (sleep and httpbin) running as shown below. Foo and Bar namespaces are running services with an [Envoy](#) proxy to add istio capabilities while legacy namespace is running a service without an istio capabilities.



## Step 1: Create namespaces

Create namespace foo, bar and legacy

```
kubectl create ns foo
```

```
kubectl create ns bar
```

```
kubectl create ns legacy
```

## Step 2: Create services

Before running the following commands, make sure to change directory to istio-workflow in the cloned github repository

We will create the `httpbin` service in the foo

```
kubectl apply -f <(istioctl kube-inject -f httpbin.yaml) -n foo
```

Next let's create `sleep` service in the `bar` (with istio) and `legacy` (without istio) namespace.

```
kubectl apply -f <(istioctl kube-inject -f sleep.yaml) -n bar
```

```
kubectl apply -f sleep.yaml -n legacy
```

Make sure that the services are running in each namespace as shown below.

```
kubectl get pods -n foo
```

```
kubectl get pods -n bar
```

```
kubectl get pods -n legacy
```

The output should be as follows

```
1 $ kubectl get pods -n foo
```

2 NAME	READY	STATUS	RESTARTS	AGE
--------	-------	--------	----------	-----

```

3 httpbin-55dcc67d67-dppwc 2/2 Running 0 6d7h
4
5 $ kubectl get pods -n bar
6 NAME READY STATUS RESTARTS AGE
7 sleep-565f8679c-jzgxl 2/2 Running 0 6d7h
8
9 $ kubectl get pods -n legacy
10 NAME READY STATUS RESTARTS AGE
11 sleep-f8cbf5b76-pxmg2 1/1 Running 0 6d7h

```

### Step 3: Send request from bar to foo

By default (from version 1.4 onwards), Istio tracks the server workloads migrated to Istio proxies, and configures client proxies to send mutual TLS traffic to those workloads automatically, and to send plain text traffic to workloads without sidecars.

Thus, all traffic between workloads with proxies uses mutual TLS, without you doing anything. For example, take the response from a request to `httpbin/header`. When using mutual TLS, the proxy injects the `X-Forwarded-Client-Cert` header to the upstream request to the backend. That header's presence is evidence that mutual TLS is used.

Replace `<pod-name-in-bar>` with the name of the pod running in bar namespace.

```
kubectl exec <pod-name-in-bar> -n bar -- curl http://httpbin.foo:8000/header
```

The output will be as follows and you can see the `X-Forwarded-Client-Cert` towards the end.

```

1 "headers": {
2   "Accept": "*/*",
3   "Content-Length": "0",
4   "Host": "httpbin.foo:8000",
5   "User-Agent": "curl/7.64.0",
6   "X-B3-Parentspanid": "833746fa90d48a4d",
7   "X-B3-Sampled": "0",
8   "X-B3-Spanid": "26765f211c8f2fb5",
9   "X-B3-Traceid": "2dbf96936064d5ca833746fa90d48a4d",
10  "X-Forwarded-Client-Cert": "By=spiffe://cluster.local/ns/foo/sa/httpbi

```

```
11 }
```

## Step 4: Send request from legacy to foo

Let's now send a request from legacy to foo and we will see how the

`X-Forwarded-Client-Cert` header will be excluded.

Replace `<pod-name-in-bar>` with the name of the pod running in legacy namespace.

```
kubectl exec <pod-name-in-bar> -n legacy -- curl http://httpbin.foo:8000/headers
```

The output will be as follows

```
1  "headers": {
2    "Accept": "*/*",
3    "Content-Length": "0",
4    "Host": "httpbin.foo:8000",
5    "User-Agent": "curl/7.64.0",
6    "X-B3-Sampled": "0",
7    "X-B3-Spanid": "df4454d90cecf080",
8    "X-B3-Traceid": "a0f060dbb32ae043df4454d90cecf080"
9  }
```

## Step 5: Enabling Istio mutual TLS in STRICT mode

While Istio automatically upgrades all traffic between the proxies and the workloads to mutual TLS between, workloads can still receive plain text traffic. To prevent non-mutual TLS for the whole mesh, set a mesh-wide peer authentication policy to set mutual TLS mode to `STRICT` .

```
kubectl apply -f policies-istio/authentication-strict.yaml
```



Once the authentication policy is enabled, request from legacy to foo should not be allowed.

```
kubectl exec <pod-name-in-bar> -n legacy -- curl http://httpbin.foo:8000/hea
```

Output will be as follows

```
1 curl: (56) Recv failure: Connection reset by peer
2 command terminated with exit code 56
```

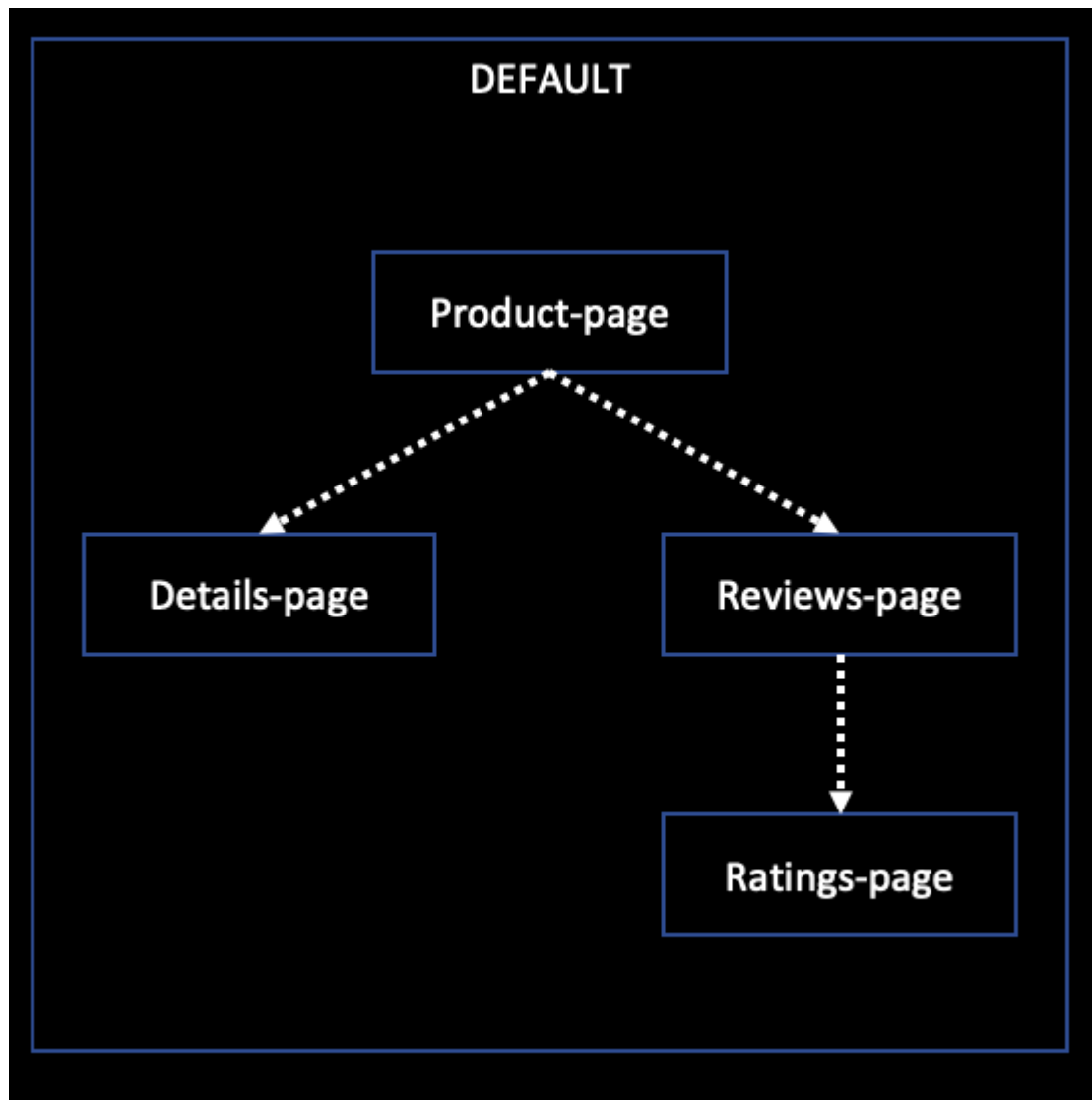
---

## Authorization

Each Envoy proxy runs an authorization engine that authorizes requests at runtime. When a request comes to the proxy, the authorization engine evaluates the request context against the current authorization policies, and returns the authorization result, either `ALLOW` or `DENY`. Operators specify Istio authorization policies using `.yaml` files.

### Step 1: Setup sample application

The sample application shows information about books. The services communicate as shown below:



We will first set up a sample application to understand how authorization works

```
kubectl apply -f <(istioctl kube-inject -f bookinfo.yaml)
```

Confirm all services and pods are correctly defined and running:

```
kubectl get pods
```

```
kubectl get services
```

Now that the Bookinfo services are up and running, you need to make the application accessible from outside of your Kubernetes cluster, e.g., from a browser. An [Istio Gateway](#) is used for this purpose.

```
kubectl apply -f bookinfo-gateway.yaml
```

Confirm the gateway has been created:

```
kubectl get gateway
```

Follow the following to set the `INGRESS_HOST` and `INGRESS_PORT` variables for accessing the gateway.

```
export INGRESS_HOST=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.status.loadBalancer.ingress[0].ip}')
```

```
export INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.status.loadBalancer.ingress[0].port}')
```

Set `GATEWAY_URL` :

```
export GATEWAY_URL=$INGRESS_HOST:$INGRESS_PORT
```

You can now point your browser to `http://$GATEWAY_URL/productpage` to view the Bookinfo web page. You will see the page properly loaded with details and reviews of the book.

## Step 2: Deny all traffic between services.

We will now apply a policy to deny all traffic.

```
kubectl apply -f policies-istio/authorization-deny-all.yaml
```

Reload `http://$GATEWAY_URL/productpage` and you not be able to access the page anymore (it might take few seconds for the policy to take effect).

### Step 3: Allow access to product service.

Run the following command to create a `productpage-viewer` policy to allow access with `GET` method to the `productpage` workload.

```
kubectl apply -f policies-istio/authorization-product.yaml
```

Reload `http://$GATEWAY_URL/productpage` and you will be able to access the product service without access to details and reviews (it might take few seconds for the policy to take effect).

### Step 4: Allow product service to access details service

Run the following command to create the `details-viewer` policy to allow the `productpage` workload, which issues requests using the `cluster.local/ns/default/sa/bookinfo-productpage` service account, to access the `details` workload through `GET` methods:

```
kubectl apply -f policies-istio/authorization-details.yaml
```

Reload `http://$GATEWAY_URL/productpage` and you will be able to access the product service with access to details (it might take few seconds for the policy to take effect).

You can do the same for reviews to be accessed by product service.

# Setup runtime security for the cluster using Sysdig Falco



Prerequisites :

- [IBM Cloud Account](#)
- [Kubernetes Cluster](#)
- [Access to the cluster.](#)
- [Cloned Github Repo](#)

Even when processes are in place for vulnerability scanning and implementing pod security and network policies, not every risk will be addressed. You still need mechanisms to confirm these security barriers are effective, help configure them, and provide with a last line of defense when they fail.

[Falco](#) lets you continuously monitor and detect container, application, host, and network activity, all in one place, from one source of data, with one set of [rules](#).

Make sure you have changed directory to falco-workflow in the cloned github repository. Let's go ahead and set up falco for our cluster.

---

## Setup Falco

### Step 1: Configure role-based access control

Since v1.8 RBAC has been available in Kubernetes, and running with RBAC enabled is considered the best practice. The `falco-account.yaml` is used to create a [Service Account](#)

for Falco, as well as the [ClusterRoles](#) and bindings to grant the appropriate permissions to the Service Account.

Let's create the service account by running the following command

```
kubectl apply -f falco-account.yaml
```

## Step 2: Create a service

We also create a service that allows other services to reach the embedded web-server in falco, which listens on https port 8765

```
kubectl apply -f falco-service.yaml
```

## Step 3: Create the falco-config ConfigMap

The Daemon Set also relies on a Kubernetes ConfigMap to store the Falco configuration and make the configuration available to the Falco Pods. This allows you to manage custom configuration without rebuilding and redeploying the underlying Pods. Any modification of the configuration should be performed on these copies rather than the original files.

The configuration files are explained below:

- `falco-config/falco.yaml` - It contains key-value pairs for falco's configuration for example location of the rules files, output format etc. Any configuration option can be overridden on the command line via the `-o/--option key=value` flag.
- `falco-config/falco_rules.yaml` - Describes rules that falco uses to log events. The [rules](#) are created using [classes](#), [macros](#) and [lists](#).
- `falco-config/falco_rules.local.yaml` - you can add your custom rules here
- `falco-config/k8s_audit_rules.yaml` - rules related to k8s architecture specifically

Combine all of the above files into a single ConfigMap by using the `--from-file` argument with a directory

```
kubectl create configmap falco-config --from-file=falco-config
```

Later, in the deployment of the daemonset, this `configmap` is mounted under `/etc/falco`.

## Step 4: Start the Falco DaemonSet

Finally, run the Falco application. It is run as a [DaemonSet](#), which enables you to run one per node

```
kubectl apply -f falco-daemonset-configmap.yaml
```

During its first-run installation, it uses Dynamic Kernel Module Support (DKMS) to compile and install a kernel module, which is how Falco picks up the system calls.

---

## Test Falco

Now you can see Falco in action. You will tail the logs in one terminal, and then synthetically create some events in the other terminal and watch the events come through the logs.

### Step 1: Get the pod name

Run the following command to get the pod name that is running falco.

```
kubectl get pods
```



---

## Step 2: Tail the logs in one terminal

We will now tail the logs by running the following command. Replace the pod-name with the output of the previous command

```
kubectl logs -f <pod-name>
```

## Step 3: Create security events in second terminal

Remember to re-export `KUBECONFIG` :

```
kubectl exec -it <pod-name> /bin/bash
```

In the first terminal you can see the event:

```
{"output":"07:25:42.605410076: Notice A shell was spawned in a container with
```

Now lets simulate another event by running the following command in the pod

```
nc -l 4444
```

The output will be as followed

```
{"output":"07:27:01.526414223: Notice Network tool launched in container (us
```

You can see in the rule file that we have a rule to generate the event when the netcat (nc) command is executed in the pod.

```
cat falco-config/falco_rules.yaml | grep -A 12 'Netcat Remote'
```

The output will show the condition and the output format of the event

```
1 - rule: Netcat Remote Code Execution in Container
2   desc: Netcat Program runs inside container that allows remote code execu
3   condition: >
4     spawned_process and container and
5     ((proc.name = "nc" and (proc.args contains "-e" or proc.args contains
6       (proc.name = "ncat" and (proc.args contains "--sh-exec" or proc.args
7         or proc.args contains "-c " or proc.args con
8     )
9   output: >
10    Netcat runs inside container that allows remote code execution (user=%
11    command=%proc.cmdline container_id=%container.id container_name=%conta
12   priority: WARNING
13   tags: [network, process, mitre_execution]
```

---

## Send alerts to slack channel

### Step 1: Create a Slack app

Click [here](#) to create the app.

### Step 2: Enable Incoming Webhooks

After creating, you'll be redirected to the settings page for your new app. From here select the *Incoming Webhooks* feature, and click the *Activate Incoming Webhooks* toggle to switch it on.

### Step 3: Create an Incoming Webhook

Now that Incoming Webhooks are enabled, click on *Add New Webhook to Workspace*. Select a channel and click Allow. Once done, copy the URL given under Webhook URL.

### Step 4: Integrate the app with Falco.

In `falco-config/falco.yaml` search for keyword slack to look for slack configuration. To include your slack webhook as shown below.

```
1 program_output:
2   enabled: true
3   keep_alive: false
4   program: "jq '{text: .output}' | curl -d @- -X POST https://hooks.slack.
```

### Step 5: Redo both the `ConfigMap` and the `DaemonSet`

```
1 kubectl delete configmap falco-config
2 kubectl create configmap falco-config --from-file=falco-config
3 kubectl delete -f falco-daemonset-configmap.yaml
4 kubectl apply -f falco-daemonset-configmap.yaml
```

Trigger the alerts again to test the slack integration.



**SecureCube** APP 11:25 AM

07:25:42.605410076: Notice A shell was spawned in a container with an attached terminal (user=root k8s.ns=default k8s.pod=falco-daemonset-x7bgs container=b1531b83f7d4 shell=bash parent=runc cmdline=bash terminal=34819 container\_id=b1531b83f7d4 image=[docker.io/falcosecurity/falco](https://hub.docker.io/falcosecurity/falco)) k8s.ns=default k8s.pod=falco-daemonset-x7bgs container=b1531b83f7d4 k8s.ns=default k8s.pod=falco-daemonset-x7bgs container=b1531b83f7d4

---

## Conclusion

You can do a lot more with Falco, including hooking up Falco events to [serverless](#) computing platforms such as OpenWhisk and [Knative](#). Hopefully, this introduction gave you some basic information that helps you get started with your next project.

This lab was completed with the help of a [code content](#) developed by a [developer advocate](#) at IBM.

# Conclusion

## Resources

### Learning

- Journey to Kubernetes Security - <https://developer.ibm.com/technologies/containers/articles/journey-to-kubernetes-security/>
- Learn more about Security capabilities for Istio - <https://istio.io/docs/tasks/security/>
- How to add authentication for end users for cloud applications - <https://suedbroecker.net/2019/12/15/no-code-changes-needed-to-secure-your-application-on-kubernetes/>
- Calico for fine-grained network access policies - <https://docs.projectcalico.org/v2.0/getting-started/kubernetes/>

### Developer Code Content

- <https://developer.ibm.com/>
- <https://developer.ibm.com/technologies/security/>

### IBM Developer Events

- San Francisco - <https://www.meetup.com/IBM-Developer-SF-Bay-Area-Meetup/>
- Dubai - <https://www.meetup.com/ibmdeveloper-dubai/>

