Component-based Software Development

Container Orchestration
Intro to Kubernetes and Rancher

Dr. Vinod Dubey
SWE 645
George Mason University

ACKNOWLEDGEMENT

Information in this lecture is adapted from materials on Kubernetes and Rancher official sites.

Issue - Recap

- **Starting and stopping one container on your laptop or in a development environment is easy!**
- **Managing a cluster of containerized applications in production is a hard problem**
  - How to make containers resilient?
  - How to achieve horizontal scalability across multiple servers?
  - How to roll out new version of my software without any service interruption?
  - How do you implement network connectivity between worker nodes and amongst containers/pods?
  - How to group related containers so that they run on the same host in order to work?

Issue - Recap

- **Starting and stopping one container on your laptop or in a development environment is easy!**
- **Managing a cluster of containerized applications in production is a hard problem**
  - How to make containers resilient?
  - How to achieve horizontal scalability across multiple servers?
  - How to roll out new version of my software without any service interruption?
  - How do you implement network connectivity between worker nodes and amongst containers/pods?
  - How to group related containers so that they run on the same host in order to work?
- **Answer: Container Orchestration**

What is Container Orchestration?

- **A general term for technologies that enable managing large collection of containers easily**
  - Allows deploying docker containers on a cluster and scale
  - Automates container lifecycle
- **Key Responsibilities**
  - Provisioning and deployment of containers
  - Health and monitoring of containers and hosts
  - Resiliency and fault tolerance
  - Redundancy and availability of containers
  - Scalability, Elasticity and load balancing
  - Movement of containers between hosts
  - Container access control (ingress/egress)
  - Allocation of container resources

Container Orchestrators

- **Popular container orchestrators**
  - Kubernetes (**K8s**) by Google
  - Docker Swarm
    - The official orchestration platform by Docker
  - Google Kubernetes Engine (GKE)
    - runs Kubernetes under the hood
  - EC2 Container Service (ECS)/EKS by AWS
  - Azure Kubernetes Service (AKS)
  - Mesosphere DC/OS
  - Marathon

What is Kubernetes?

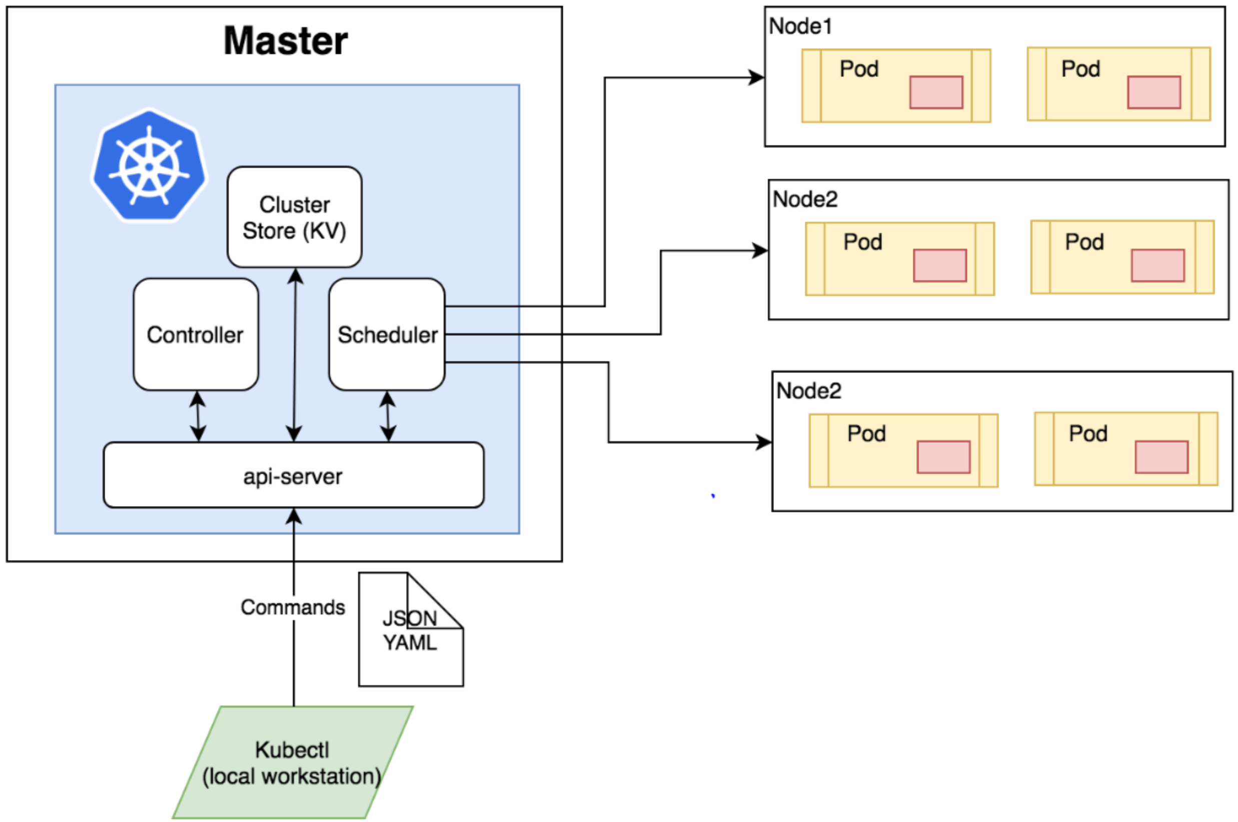
- **Kubernetes is an open-source container-orchestration system**
  - for automating deployment, scaling, and management of containerized applications
  - It was originally designed by Google
  - Kubernetes can run anywhere – on premises or cloud

What is Kubernetes?

- **Kubernetes takes declared state of how you would like to configure your containers in a form of YAML file and it makes that happen, even across computers.**
  - It will deploy your containers and make them publicly available, among other things.

Kubernetes Architecture

- This is a visualization of what a Kubernetes cluster would look like



Source: Google

Key Components that Kubernetes uses to deploy an application

- Kubernetes objects to deploy and manage powerful distributed applications
  - Pod
  - ReplicaSet
  - Deployment
  - Service
  - Ingress
  - Volume
  - StatefulSet
  - ConfigMap
  - Secret
- Kubernetes uses **YAML to specify the configuration** of Kubernetes objects

YAML

- YAML stands for
  - Yet another Markup Language OR
  - YAML Ain't Markup Language (depending on who you are talking to!)
- A human-readable text-based format for specifying configuration information for Kubernetess components to be deployed on the Kubernetes cluster

YAML

- **Four top level keys in any YAML file to specify configuration of K8s objects:**
  - apiVersion,
  - kind,
  - metadata, and
  - spec

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
```

- **Metadata generally defines:**
  - name,
  - labels, which are arbitrary key value pairs that uniquely identifies the group of pods together, Can be used for selectors
  - annotations

Key Components: Pod

- **The smallest unit of deployment on K8s cluster**
  - In K8s, you work with pods (not with containers directly).
- **Provides a runtime environment for your containerized application**
  - An abstraction over a container
- **Usually a pod runs one container inside of it**
  - But may have more than one container that may need to be scheduled together
- **Each Pod gets its own IP address**
- **Pods are ephemeral**
  - Pods can crash for any reason
  - When a pod crashes, K8s (kubelet) starts a new pod and the new pod gets new IP address

Key Components: Pod

- **Pod Manifest**
  – Everything in K8s happens through yaml manifest
  – Spec section provides the technical details to define your application

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

Key Components: ReplicaSet

- **ReplicaSet manages pods**
  - it's an abstraction over Pod
- **It ensures how many replicas of a pod should be running.**
  - Guarantees the availability of a specified number of identical Pods

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # modify replicas according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
```

Key Components: Deployment

- *In reality, you do not work with ReplicaSet or Pod objects directly!*
- **Use a Deployment object instead, and define your application in the spec section**
- **Deployment manages ReplicaSets and provides a level of abstraction above ReplicaSet, which manages pods**
  - Deployment serves as a controller.

Key Components: Deployment

- **Deployment defines a blueprint** for *your pod* and **specifies how many replicas** of that Pod you would like to run

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Key Components: Deployment

- **In this example**
  - A Deployment named nginx-deployment is created, indicated by the .metadata.name field
  - The .spec.selector field defines how the Deployment finds which Pods to manage.
    - In this case, you select a label that is defined in the Pod template (app: nginx).

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Key Components: Deployment

- **In this example**
  - The template field contains the following sub-fields:
    - The Pods are labeled app: nginx using the .metadata.labels field.
    - The Pod template's specification, or .template.spec field, indicates that the Pods run one container, nginx, which runs the nginx image version 1.14.2 (pulling from Docker Hub by default).
    - Create one container and name it nginx using the .spec.template.spec.containers[0].name field.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Deployment – Another example

- **In this example, Deployment defines a** **blueprint** for *your pod* and **specifies** **<u>one replica</u>** of that Pod you would like to run

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.16-alpine
        ports:
        - containerPort: 80
        volumeMounts:
        - name: index
```

Deployment – Another example

- **Deployment creates ReplicaSet and ReplicaSet creates bunch of pods**
  - Under the spec section, there is a template section that contains definition for the Pod
  - <u>As such, Deployment launches pods</u>

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.16-alpine
        ports:
        - containerPort: 80
        volumeMounts:
        - name: index
```

Deployment – Another example

- **ReplicaSet uses values defined in the matchLables under selector to identify pods it's going to manage**
  - In the example, the ReplicaSet manages everything that has label app: nginx, which is the pod

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.16-alpine
        ports:
        - containerPort: 80
        volumeMounts:
        - name: index
```

Key Components: Service

- **Service defines a stable DNS name and a stable IP address to communicate with a group of pod instances**
  - Provides a consistent endpoint for a group of pods
  - The life cycles of Service and the Pod are not connected to each other.
    - If the Pod dies, the Service and its IP address will stay.
    - Apps using service ip address will not be impacted even when a pod crashes and replaced by a new one
  - Exposes workload inside or outside the cluster

Key Components: Service

- A Service also serves as a virtual load balancer for pods – it receives the traffic and routes them to the destination
- 3 Types: ClusterIP, NodePort, LoadBalancer
  - When you create NodePort, it also creates ClusterIP
  - When you create LoadBalancer, it also creates NodePort and ClusterIP
- **Visual example**
  - https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0

Key Components: Service

- **ClusterIP Service**
  - ClusterIP creates service (with an address) that’s accessible only inside the cluster
  - Useful for pods communicating with another pod inside the cluster

Key Components: Service

- **NodePort Service**
  - NodePort service first creates ClusterIP service and then opens a high port on every node in the cluster
    - randomly chosen in range 30000 to 32767
  - The traffic lands on the port, then gets routed to clusterIP service, and then gets routed to the pod
  - NodePort services generally used for external load balancers,
    - Either the load balancer you configure yourself or the cloud load balancer that Kubernetes can configure

Key Components: Service

- **LoadBalancer Service**
  - If your cluster is in a cloud, then Kubernetes first creates ClusterIP service, then opens high port on each node, and then reaches out to cloud provider's API and configures a cloud load balancer with the host and port
  - If you add node or remove node, it automatically updates load balancer config
  - Traffic lands on load balancer, gets routed to the NodePort, gets routed to the clusterIP service, and then finally gets routed to the Pod

Key Components: Service

- **Services use selector to determine which pod group to forward the requests to**
  – In the example, it is going to route traffic to any pod that has label app: nginx

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: nginx
  name: nginx
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  sessionAffinity: None
  type: NodePort
```

Key Components: Service

- **Another example with Deployment and Service (NodePort) side-by-side –** Services use selector to determine which pod group to forward the requests to
  - If K8s cluster hosted on EC2 in AWS, this nginx page can be accessed using public-DNS-of-EC2:xxxxx, where xxxxx is the 5-digit high port on the worker node
  - https://kubernetes.io/docs/concepts/services-networking/connect-applications-service/

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  selector:
    matchLabels:
      run: my-nginx
  replicas: 2
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
      - name: my-nginx
        image: nginx
        ports:
        - containerPort: 80
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-nginx
  labels:
    run: my-nginx
spec:
  type: NodePort
  ports:
  - port: 8080
    targetPort: 80
    protocol: TCP
    name: http
  - port: 443
    protocol: TCP
    name: https
  selector:
    run: my-nginx
```

Issues with NodePort and LoadBalancer services

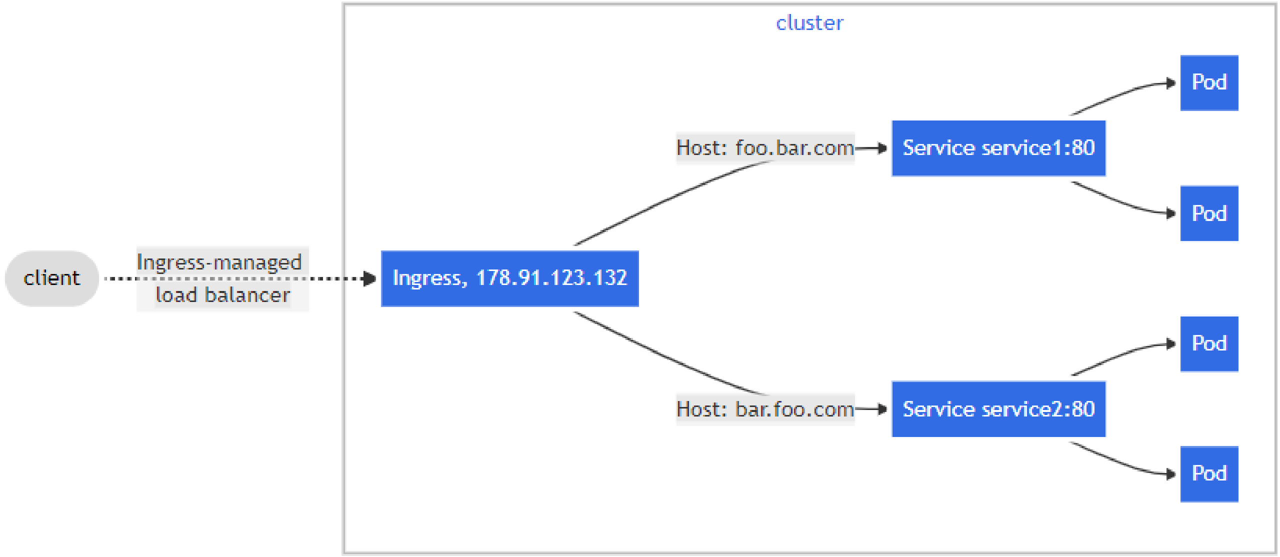
- **Although NodePort service can facilitate external traffic to get to the relevant pods, using NodePort service with 5-digit high port number (e.g., mydomain.com:#####) is not the convention and not practical**
  - You would rather use mydomain.com/app1 etc.
- **Regarding load balancer, there is 1-1 relationship between service on the cluster and cloud load balancer**
  - This can be expensive due to the cost of the cloud load balancer if your Kubernetes has hundreds of services serving hundreds of websites/applications
  - You require same number of cloud load balancers as number of load balancer services
- *Kubernetes solves these two problems with Ingress, discussed next*

Key Components: Ingress

- **Ingress defines <u>how the traffic outside the cluster</u> is <u>routed to pods inside the cluster</u>**
  - Used to expose Kubernetes services to the world
  - Routes traffic to internal services based on host and path
- **Ingress is usually implemented by an ingress controller (a layer 7 load balancer), such as Nginx, HAProxy, etc.**
  - Kubernetes cluster on Rancher already has Nginx Ingress Controller running

Key Components: Ingress

- **You create a Kubernetes Ingress object and configure it to route traffic to a service object e.g., ClusterIP or NodePort.**
  - The service object (e.g., of type ClusterIP) should have been created before creating/configuring Ingress object.
  - The service object, in turn, forwards the requests to appropriate application pod(s) based on path.
- **So, the request goes first to Ingress, which does the forwarding to the Service, such as ClusterIP which forwards traffic to a pod encapsulating an application.**



Key Components: Ingress

- **Ingress manifest**
  - It says traffic coming in for the specified host and specified path (/ in the example) be sent to service named nginx at port 80.
  - Here service nginx could be of type ClusterIP

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: demo-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: training-a.cl.monach.us
    http:
      paths:
      - path: /
        backend:
          serviceName: nginx
          servicePort: 80
```

Key Components: Ingress

- **host name is valid DNS name (CNAME record) that points to a physical load balancer (e.g., in AWS)**
  - A CNAME record can be created using Route53 DNS service in AWS or in an Active Directory

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: demo-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: training-a.cl.monach.us
    http:
      paths:
      - path: /
        backend:
          serviceName: nginx
          servicePort: 80
```

Key Components: Ingress

- **The physical load balancer (in aws) should listen (using listeners configuration, which is part of LB) at port 80 and port 443.**
- **The LB forward traffic to targets (*created using target groups that point to worker nodes of the Kubernetes cluster*) on port 80 and 443.**
  - Worker nodes are where Ingress Controller is deployed, listening on port 80 and 443.
  - Ingress controller listens only on port 80 and 443

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: demo-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: training-a.cl.monach.us
    http:
      paths:
      - path: /
        backend:
          serviceName: nginx
          servicePort: 80
```

Key Components: Ingress

- **To configure Ingress object with SSL termination requires TLS-Secret**
  - Kubernetes provides a built-in Secret type kubernetes.io/tls for storing a certificate and its associated key that are typically used for TLS .
  - When using this type of Secret, the **tls.key** and the **tls.crt key** must be provided in the **data field** of the Secret configuration.

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: secret-tls
type: kubernetes.io/tls
data:
  # the data is abbreviated in this example
  tls.crt: |
        MIIC2DCCAcCgAwIBAgIBATANBgkqh ...
  tls.key: |
        MIIEpgIBAAKCAQEA7yn3bRHQ5FHMQ ...
```

***An example YAML with TLS Secret configuration*
***The data for tls.crt and tls.key should be in one continuous line without any line break.*

Key Components: Ingress with SSL termination

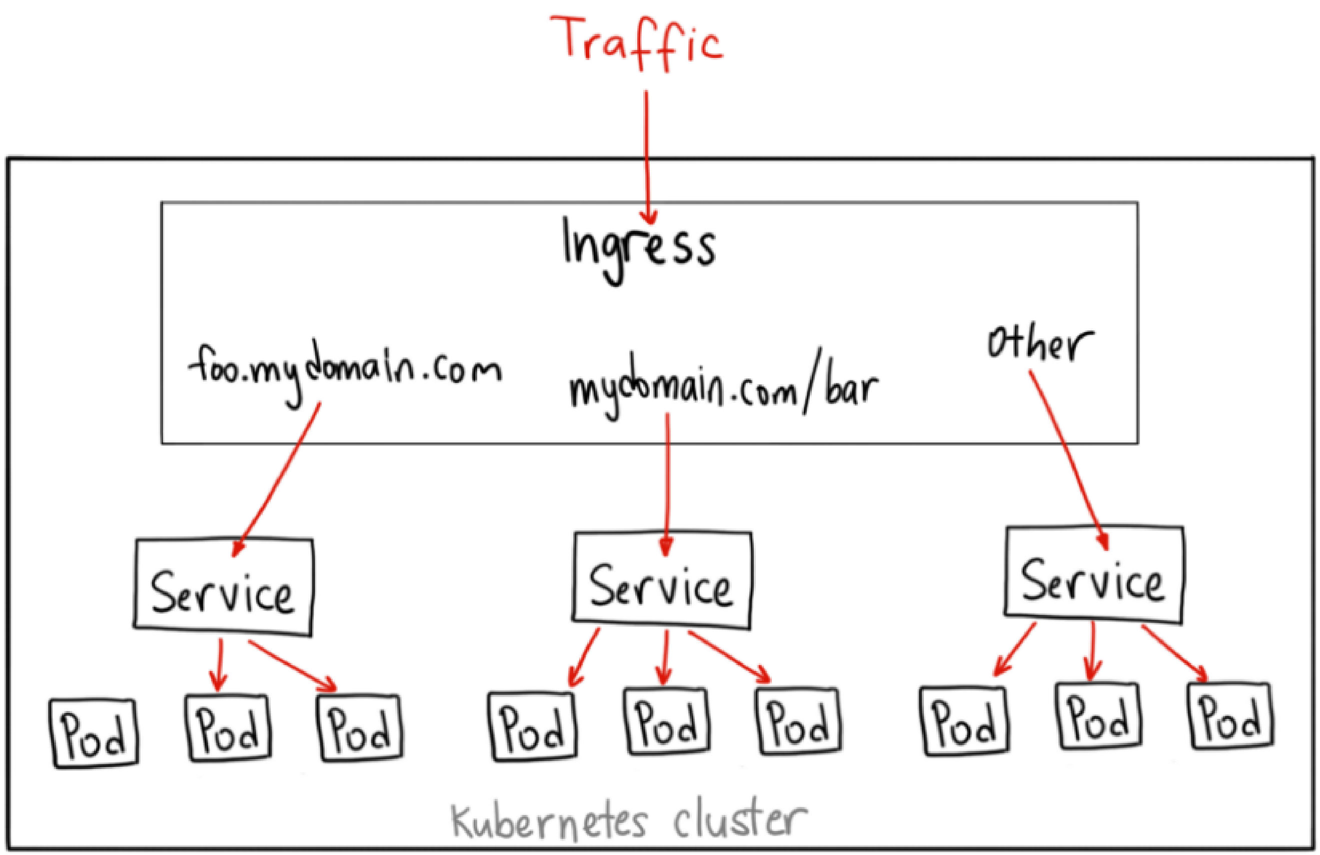
- **You can secure an Ingress by specifying a Secret**
  - that contains a TLS private key and certificate.
- **The Ingress resource only supports a single TLS port 443, and assumes TLS termination at the ingress point**
  - traffic to the Service and its Pods is in plaintext.

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: testsecret-tls
  namespace: default
data:
  tls.crt: base64 encoded cert
  tls.key: base64 encoded key
type: kubernetes.io/tls
```

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: tls-example-ingress
spec:
  tls:
  - hosts:
      - https-example.foo.com
    secretName: testsecret-tls
  rules:
  - host: https-example.foo.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: service1
            port:
              number: 80
```

Key Components: Ingress

- **Ingress depiction**
  – It says traffic coming in for the specified host and specified path be sent to service named such-and-such at port such and such



Key Components: ConfigMap

- **ConfigMap contains an external configuration for your application.**
  - For example, URLs of databases
  - ConfigMap is created independently of the pod and then passed to the pod on startup.
  - Allows to override data inside container at runtime
- **key-value pairs that appear inside containers as environment variables**
- **If you change the name of the service endpoint, just update the ConfigMap , restart the pod.**
  - You don't have to rebuild an image and go through this whole cycle.

Key Components: Secret

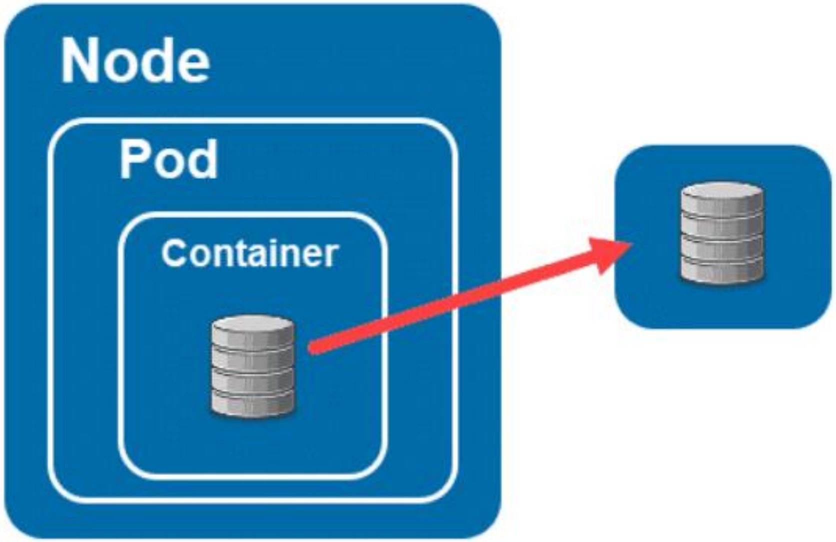
- **Secret is like ConfigMap, but is used to store secret data, like credentials or certificates**
  - It's stored in base64 encoded format.
- **Like ConfigMap, you connect Secret to your Pod**
- **You can use the data from ConfigMap or Secret inside your application Pod either as environmental variables or as a properties file.**

Data Persistence in Kubernetes : Volume

- **The container's local filesystem is ephemeral which means when a Pod dies, the data is lost.**
- **Volumes are ways to provide long-term storage (**e.g., a directory or ebs volume) accessible to all **containers in Pods in the K8s cluster.**
  - A K8s object that persists, even between the pod restarts.
- **It attaches a physical storage on a hard drive to your Pod.**
  - Persistence storage is like an external hard drive plugged in to K8s cluster, and accessible to the containers in pods.

Key Components: Volume

- You can use a **mount point in the container** so that containers can persist and share (read/write) data




```yaml
apiVersion: v1
kind: Pod
metadata:
    name: busybox
    namespace: default
spec:
    containers:
    - image: busybox
      name: busy
      command:
        - sleep
        - "3600"
      volumeMounts:
      - mountPath: /scratch
        name: scratch-volume
    volumes:
    - name: scratch-volume
      emptyDir: {}
```

Key Components: Volume

- There are **different types of volumes** in Kubernetes and the **type** defines how the volume is created

  - awsElasticBlockStore
  - azureDisk
  - azureFile
  - cephfs
  - csi
  - downwardAPI
  - emptyDir
  - fc (fibre channel)
  - flocker
  - gcePersistentDisk
  - gitRepo
  - glusterfs
  - hostPath
  - iscsi
  - local
  - nfs
  - projected
  - portworxVolume
  - quobyte
  - rbd
  - scaleIO
  - secret
  - storageos
  - vsphereVolume
  - persistentVolumeClaim

Key Components: Persistent Volume

- **Persistent Volume (PV)** – A networked storage provisioned by the Kubernetes administrator.
  - It's a resource in the cluster which is independent of any individual pod that uses the PV.
  - Persistent Volumes are not namespaced, rather cluster wide objects accessible from any namespace
  - <u>PV has the actual storage backend</u>

Key Components: Persistent Volume Claim

- **Persistent Volume Claim (PVC) –** The storage requested by Kubernetes for its pods
  - A handle to persisted volume – a claim binds to a volume
    - Through handle you get access to persistent volumes
  - The claims must be created in the same namespace where the pod is created.
  - Binding between PV and PVC is one-to-one
  - We use PVC in the Pod's definition

YAML examples for PV and PVC

- **A PVC <u>matches</u> a possible PV in the cluster**

```yaml
kind: PersistentVolume --------
apiVersion: v1
metadata:
   name: pv0001 ---------------
   labels:
      type: local
spec:
   capacity: -----------------
      storage: 10Gi -----------
   accessModes:
      - ReadWriteOnce ---------
      hostPath:
         path: "/tmp/data01" --
```

```yaml
kind: PersistentVolumeClaim ---
apiVersion: v1
metadata:
   name: myclaim-1 ------------
spec:
   accessModes:
      - ReadWriteOnce ---------
   resources:
      requests:
         storage: 3Gi ---------
```

Using PV/PVC with Pod

- **Pods request the volume (PV) through PVC**
- **PVC tries to find a matching PV in the cluster**
- **In this Example:**
  - **volumeMounts:** specifies the path in the container to mount the volume
  - **Volumes**: provides the volume definition to be claimed
  - **persistentVolumeClaim**: defines the PVC name to be used in the pod
- **Apps can access the mounted data in /usr/share/tomcat/html**

```
apiVersion: v1
metadata:
   name: mypod
   labels:
      name: frontendhttp
spec:
   containers:
   - name: myfrontend
      image: nginx
      ports:
      - containerPort: 80
         name: "http-server"
      volumeMounts:
      - mountPath: "/usr/share/tomcat/html"
         name: mypd
   volumes:
      - name: mypd
         persistentVolumeClaim:
         claimName: myclaim-1
```

Volume is mounted into Container

Volume is mounted into the Pod

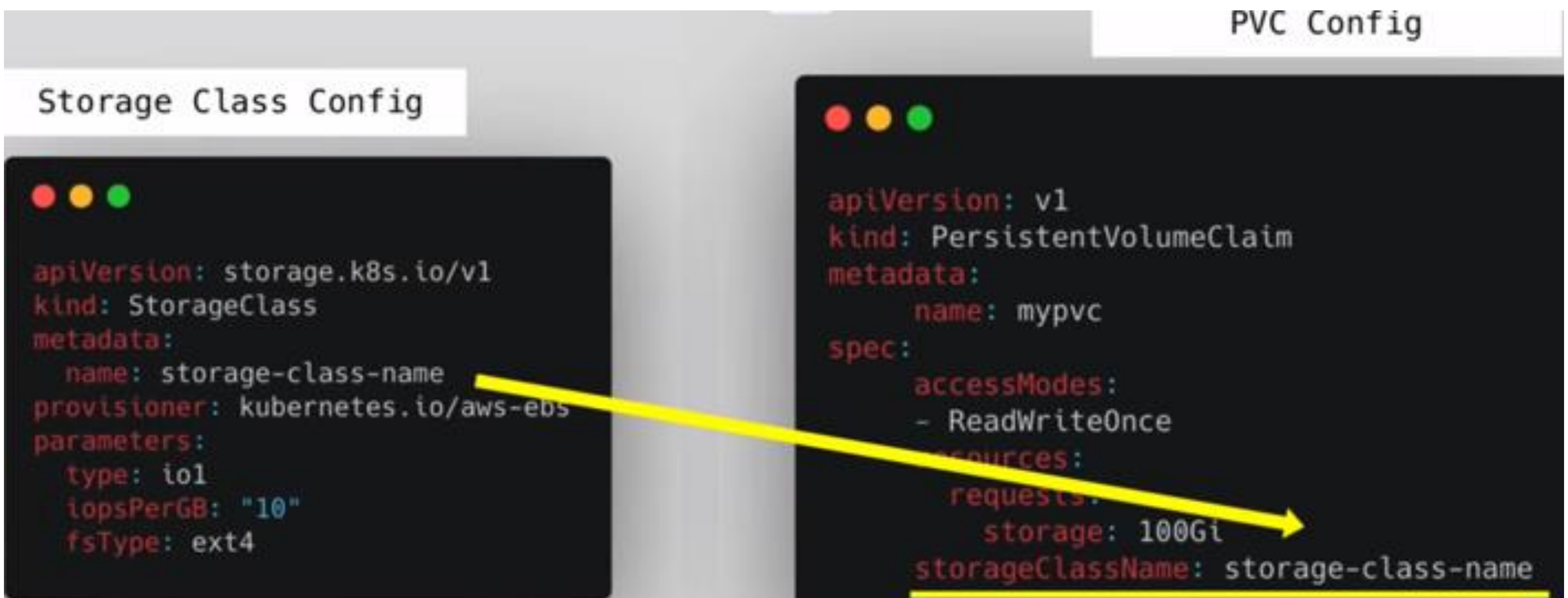
Storage Class (SC)

- **SC provides dynamic provisioning of PVs, …when PVC claims it**
  - Dynamic provisioning avoids wasting storage
  - Storage backend is defined in the SC component via provisioner attribute
  - Parameters: specifies the parameters to be configured for storage we want to request for PV

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: storage-class-name
provisioner: kubernetes.io/aws-ebs
parameters:
  type: io1
  iopsPerGB: "10"
  fsType: ext4
```

Storage Class

- **Storage Class is requested by PersistentVolumeClaim**
  - Pod claims storage via PVC
  - PVC requests storages from StorageClass
  - StorageClass creates PV that meets the needs of the claim



An example with Storage Class, PV, PVC, Pod

1

```yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: ebs-sc
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
```

2

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ebs-claim
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: ebs-sc
  resources:
    requests:
      storage: 4Gi
```

3

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
  - name: app
    image: centos
    command: ["/bin/sh"]
    args: ["-c", "while true; do echo $(date -u) >> /data/out.txt; sleep 5; done"]
    volumeMounts:
    - name: persistent-storage
      mountPath: /data
  volumes:
  - name: persistent-storage
    persistentVolumeClaim:
      claimName: ebs-claim
```

Example: PV using Storage Class

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: test-pv
spec:
  capacity:
    storage: 50Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  storageClassName: ebs-sc
  csi:
    driver: ebs.csi.aws.com
    volumeHandle: vol-05786ec9ec9526b67
    fsType: xfs
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: topology.ebs.csi.aws.com/zone
          operator: In
          values:
          - us-east-1c
```

Amazon EBS CSI Driver

- **The Amazon Elastic Block Store Container Storage Interface (CSI) Driver provides a CSI interface used by Container Orchestrators, such as Kubernetes, to manage the lifecycle of Amazon EBS volumes.**
  - **Controller Service**:
    - CreateVolume, DeleteVolume, ControllerPublishVolume, ControllerUnpublishVolume, ControllerGetCapabilities, ValidateVolumeCapabilities, CreateSnapshot, DeleteSnapshot, ListSnapshots
  - **Node Service**:
    - NodeStageVolume, NodeUnstageVolume, NodePublishVolume, NodeUnpublishVolume, NodeGetCapabilities, NodeGetInfo
  - **Identity Service**:
    - GetPluginInfo, GetPluginCapabilities, Probe

Source: https://github.com/kubernetes-sigs/aws-ebs-csi-driver

Amazon EBS CSI Driver

- **CreateVolume parameters that can be passed into CreateVolumeRequest.parameters map**

Parameters	Values	Default	Description
"csi.storage.k8s.io/fsType"	xfs, ext2, ext3, ext4	ext4	File system type that will be formatted during volume creation
"type"	io1, io2, gp2, sc1, st1,standard	gp2	EBS volume type
"iopsPerGB"			I/O operations per second per GiB. Required when io1 or io2 volume type is specified. If this value multiplied by the size of a requested volume produces a value below the minimum or above the maximum IOPs allowed for the volume type, as documented here, AWS will return an error and volume creation will fail
"encrypted"			Whether the volume should be encrypted or not. Valid values are "true" or "false"
"kmsKeyId"			The full ARN of the key to use when encrypting the volume. When not specified, the default KMS key is used

Amazon EBS CSI Driver

- **Features**
  - **Static Provisioning -** create a new or migrating existing EBS volumes, then create PV from the EBS volume and consume the PV from container using PVC
  - **Dynamic Provisioning -** uses PVC to request the Kuberenetes to create the EBS volume on behalf of user and consumes the volume from inside container.
    - Storage class's allowed Topologies could be used to restrict which AZ the volume should be provisioned in.
    - The topology key should be topology.ebs.csi.aws.com/zone.
  - **Mount Option -** mount options could be specified in PV to define how the volume should be mounted.
  - **Volume Resizing -** expand the volume size.

Amazon EBS CSI Driver

- **Prerequisite**
  - A working Kubernetes cluster on AWS
  - Can access Kubernetes cluster using kubectl, helm

Amazon EBS CSI Driver: Installation

- **Set up driver permission**
  - The ebs csi driver requires IAM permission to talk to Amazon EBS to manage the volume on user's behalf.
- **Two prominent methods to grant driver IAM permission**
  - **Using secret object OR**
  - **Grant proper permissions to all worker nodes**

Amazon EBS CSI Driver: Installation

- **Using secret object -** create an IAM user with proper permission, put that user's credentials in secret manifest then deploy the secret.
- curl https://raw.githubusercontent.com/kubernetes-sigs/aws-ebs-csi-driver/master/deploy/kubernetes/secret.yaml > secret.yaml
- # Edit the secret with user credentials
- kubectl apply -f secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: aws-secret
  namespace: kube-system
stringData:
  key_id: ""
  access_key: ""
```

Amazon EBS CSI Driver: Installation

– **Grant proper permissions to all worker nodes -** Pass an IAM role to an EC2 instance to grant all the worker nodes with proper permission by attaching policy to the instance profile of the workers.

https://github.com/kubernetes-sigs/aws-ebs-csi-driver/blob/master/docs/example-iam-policy.json


```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:AttachVolume",
        "ec2:CreateSnapshot",
        "ec2:CreateTags",
        "ec2:CreateVolume",
        "ec2:DeleteSnapshot",
        "ec2:DeleteTags",
        "ec2:DeleteVolume",
        "ec2:DescribeAvailabilityZones",
        "ec2:DescribeInstances",
        "ec2:DescribeSnapshots",
        "ec2:DescribeTags",
        "ec2:DescribeVolumes",
        "ec2:DescribeVolumesModifications",
        "ec2:DetachVolume",
        "ec2:ModifyVolume"
      ],
      "Resource": "*"
    }
  ]
}
```

Amazon EBS CSI Driver: Installation

- **Deploy Driver–**
  - kubectl apply -k "github.com/kubernetes-sigs/aws-ebs-csi-driver/deploy/kubernetes/overlays/stable/?ref=master“
  - OR

```
helm upgrade --install aws-ebs-csi-driver \
    --namespace kube-system \
    --set enableVolumeScheduling=true \
    --set enableVolumeResizing=true \
    --set enableVolumeSnapshot=true \
    https://github.com/kubernetes-sigs/aws-ebs-csi-driver/releases/download/v0.7.0/helm-chart.tgz
```

- **Verify driver is running:**
  - kubectl get pods -n kube-system

Amazon EBS CSI Driver: Configuring Storage Class

- **An <u>example</u> to configure Kubernetes storage class to provision EBS volumes with various configuration parameters.**
  - You may need to edit the StorageClass spec in example manifest and update storageclass parameters to desired value.
  - In this example, a io1 EBS volume is created and formatted to xfs filesystem with encryption enabled using the default KMS key.

Source: https://github.com/kubernetes-sigs/aws-ebs-csi-driver/tree/master/examples/kubernetes/storageclass

Amazon EBS CSI Driver: Configuring Storage Class

```yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: ebs-sc
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  csi.storage.k8s.io/fstype: xfs
  type: io1
  iopsPerGB: "50"
  encrypted: "true"
allowedTopologies:
- matchLabelExpressions:
  - key: topology.ebs.csi.aws.com/zone
    values:
    - us-east-1a
---
```

1

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ebs-claim
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: ebs-sc
  resources:
    requests:
      storage: 4Gi
---
```

2

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
  - name: app
    image: centos
    command: ["/bin/sh"]
    args: ["-c", "while true; do echo $(date -u) >> /data/out.txt; sleep 5; done"]
    volumeMounts:
    - name: persistent-storage
      mountPath: /data
  volumes:
  - name: persistent-storage
    persistentVolumeClaim:
      claimName: ebs-claim
```

3

Amazon EBS CSI Driver: Dynamic Provisioning of Persistent Volume

1

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: ebs-sc
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
```

2

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ebs-claim
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: ebs-sc
  resources:
    requests:
      storage: 4Gi
```

3

```
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
  - name: app
    image: centos
    command: ["/bin/sh"]
    args: ["-c", "while true; do echo $(date -u) >> /data/out.txt; sleep 5; done"]
    volumeMounts:
    - name: persistent-storage
      mountPath: /data
  volumes:
  - name: persistent-storage
    persistentVolumeClaim:
      claimName: ebs-claim
```

Source: https://github.com/kubernetes-sigs/aws-ebs-csi-driver/tree/master/examples/kubernetes/dynamic-provisioning/specs

Rancher

- **Rancher is software to create and manage Kubernetes cluster(s),**
  - including UI, central authentication, logging, monitoring, service mesh and much more
- **Rancher supports any Kubernetes cluster which can be added to Rancher in different ways:**
  - Import existing cluster by launching needed resources to manage the cluster (Rancher agents)
  - Create a hosted Kubernetes cluster (for example, GKE (Google), AKS (Azure), EKS (AWS))
  - Create a custom cluster (uses parts of RKE)

How to Install Rancher

- **Using Docker installation**
  - Appropriate for dev/test or homework needs
- **Using RKE and helm chart**
  - Allows to create Single node or high availability Kubernetes Installation
  - And then install rancher using Rancher Helm chart
  - Appropriate for enterprise applications in production
  - https://rancher.com/docs/rancher/v2.6/en/installation/

Installing Docker Runtime

- Docker is required to be installed on all nodes that runs the Rancher server.
- Refer to the official Docker documentation about how to install Docker on Linux.
  - The steps will vary based on the Linux distribution.
- Rancher's Docker installation scripts,
  - For example, this command could be used to install Docker 19.03 **on Ubuntu**:
  - curl https://releases.rancher.com/install-docker/19.03.sh | sh
- For example, on Redhat AMI based EC2 instances, follow steps the below AWS link
  - https://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html

Installing Docker on Redhat AMI-based EC2 instance

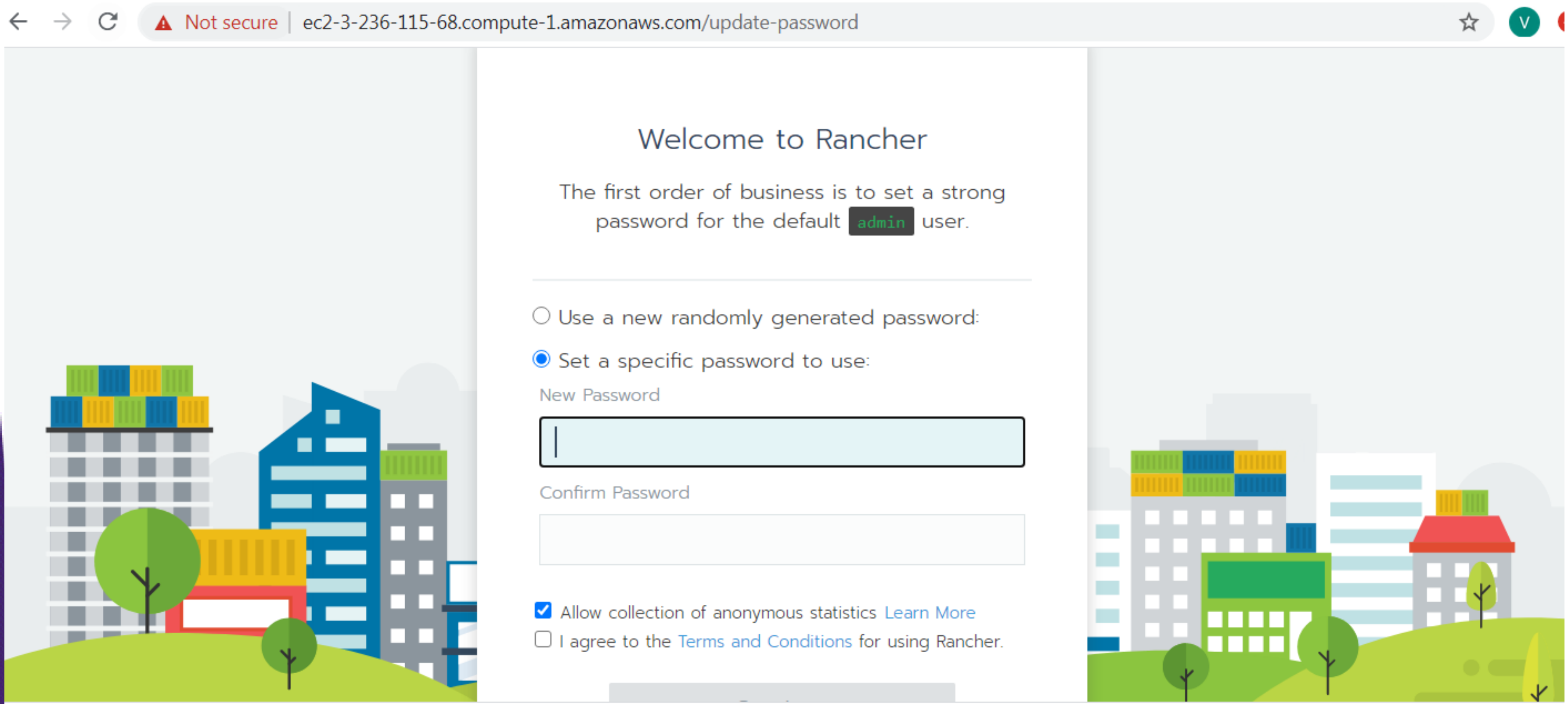
- Launch and connect to an instance with Redhat AMI.
- Update the installed packages and package cache on your instance
  - sudo yum update –y
- Install the most recent Docker Community Edition package.
  - sudo yum install docker
- Start the Docker service.
  - sudo service docker start
- Add the ec2-user to the docker group so you can execute Docker commands without using sudo.
  - sudo usermod -a -G docker ec2-user
- Log out and log back in again to pick up the new docker group permissions.
- Verify that the ec2-user can run Docker commands without sudo.
  - docker info
- Source: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/docker-basics.html

Installing Rancher

- On the EC2 instance where you have the docker runtime running, execute the following docker run command:
  - docker run -d --restart=unless-stopped -p 80:80 -p 443:443 rancher/rancher:latest   OR
  - docker run -d --restart=unless-stopped -p 80:80 -p 443:443 rancher/rancher:stable
  - You can check the status with 'docker ps'
  - Once the rancher docker instance is running, go to the public DNS of your EC2 to see Rancher console
  - Source: https://rancher.com/docs/rancher/v2.x/en/installation/other-installation-methods/single-node-docker/
  - https://rancher.com/docs/rancher/v2.6/en/installation/other-installation-methods/single-node-docker/

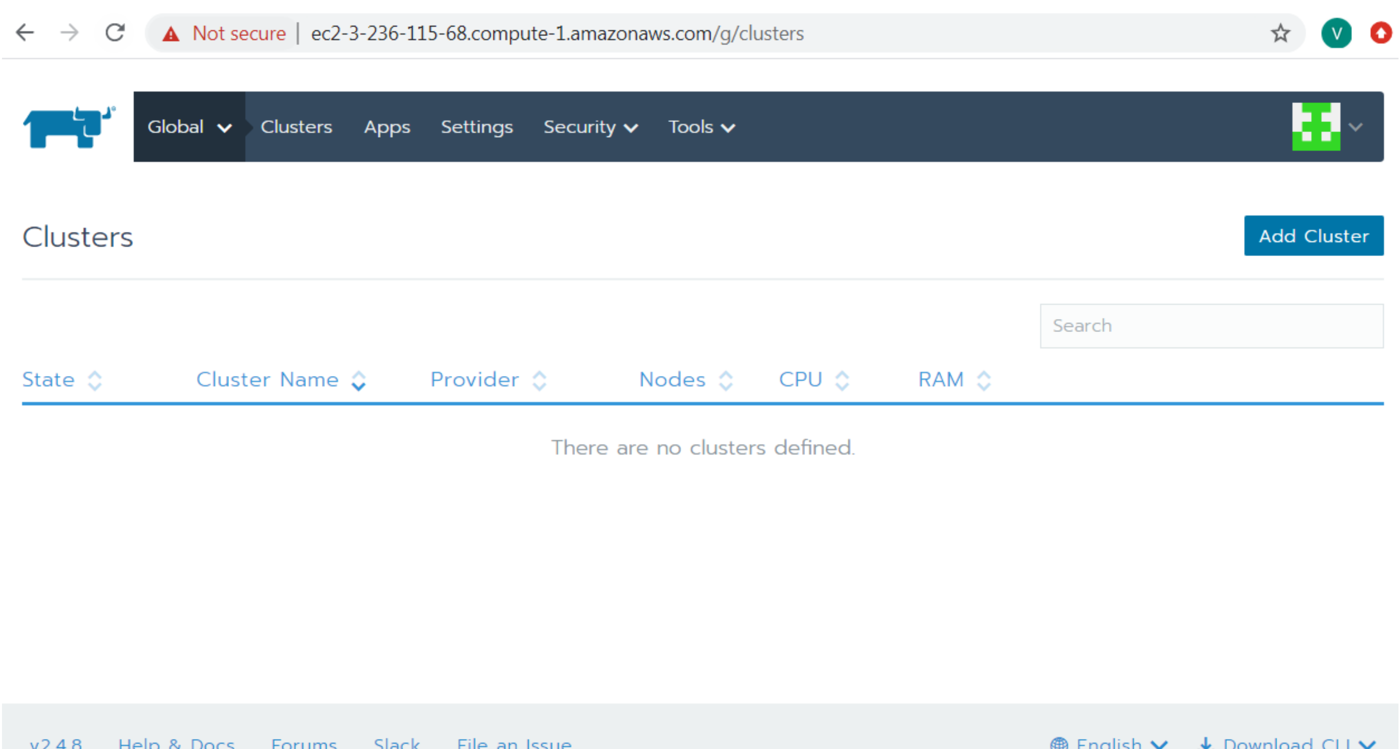
Accessing Rancher

- Once the rancher docker instance is running, go to the public DNS of your EC2 to see Rancher console
- It uses 'admin' as username and prompts you to create password



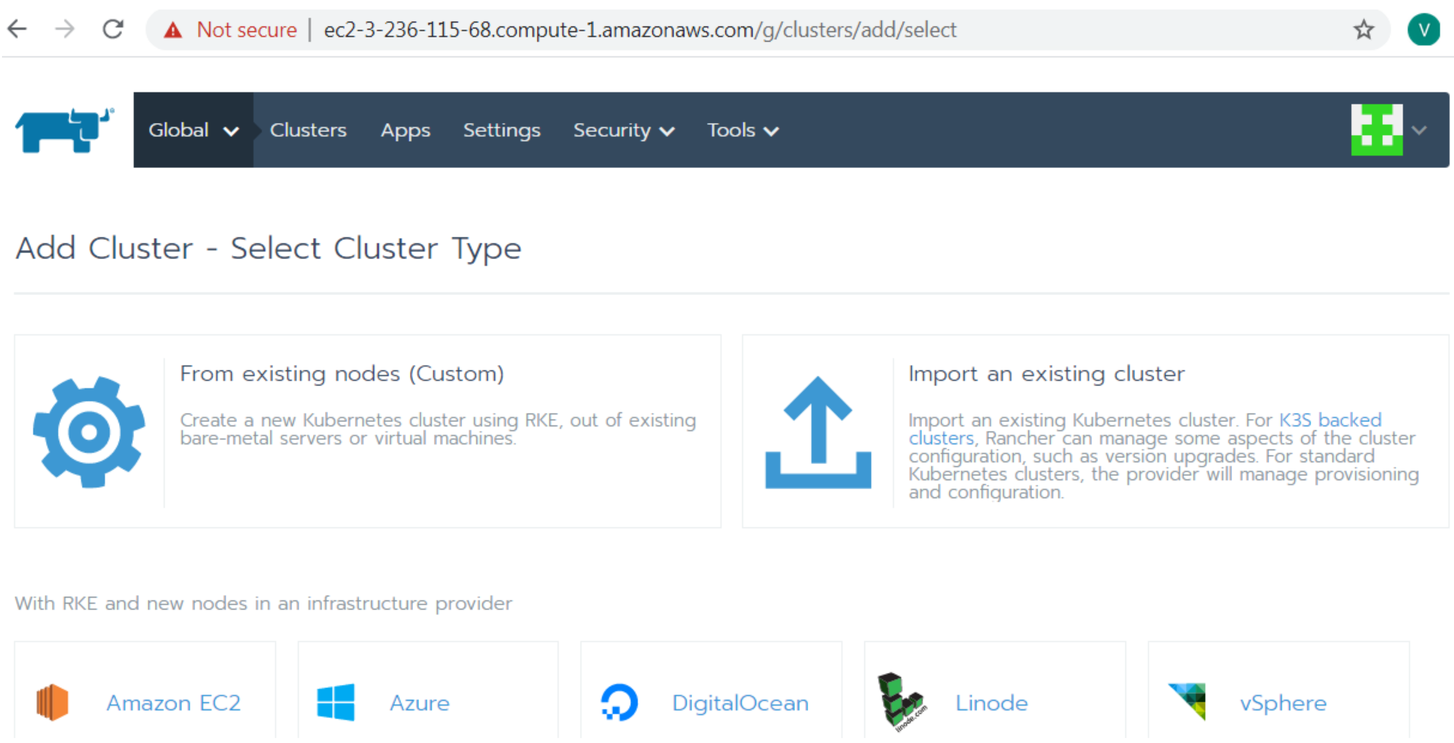
Installing Rancher

- After providing a valid password, you will be directed to Rancher console
- Make sure you save the URL, username (admin) and password for future reference
- On the Rancher console, press “Add Cluster” in the upper right corner



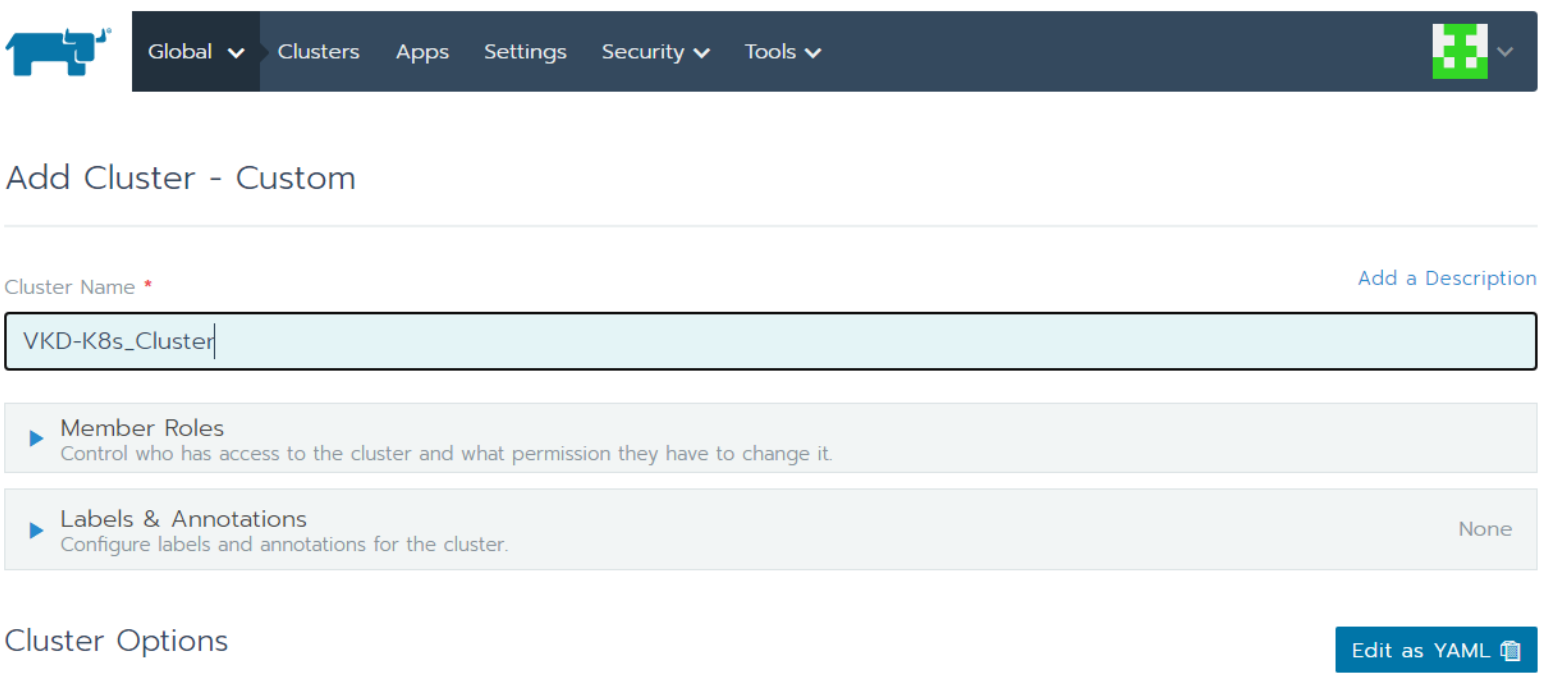
Creating K8s Cluster using Rancher

- After pressing “Add Cluster” in the upper right corner, you will be presented different options to create cluster
- Select “Custom”



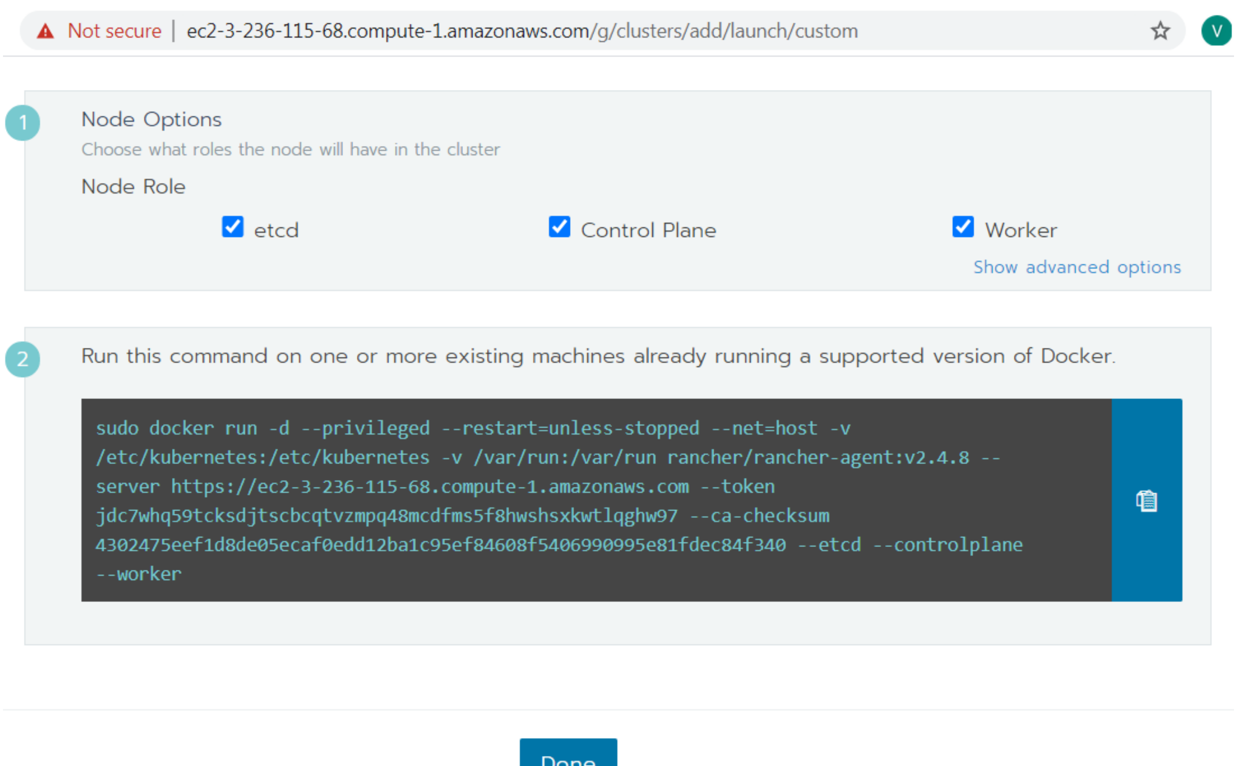
Creating K8s Cluster using Rancher

- After selecting “Custom”, provide the name of your cluster, you can leave the rest as default, and then press “Next” in the bottom



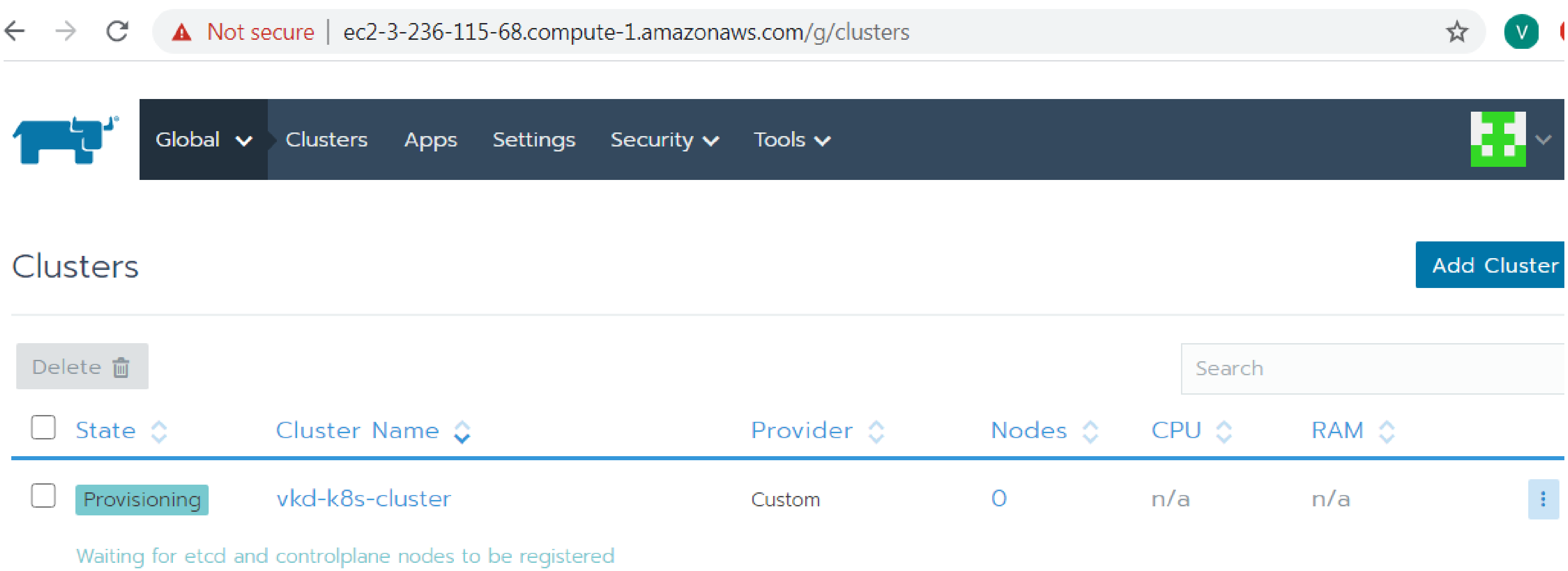
Creating K8s Cluster using Rancher

- Next, you will be presented with options to choose what roles the node will have in the cluster
- Make sure you check all three checkboxes: etcd, control plane, and worker
- You also have option to create master (with etcd, control plane) and worker nodes separate



Creating K8s Cluster using Rancher

- Copy and run the “sudo docker run …….” on another EC2 instance to start the cluster



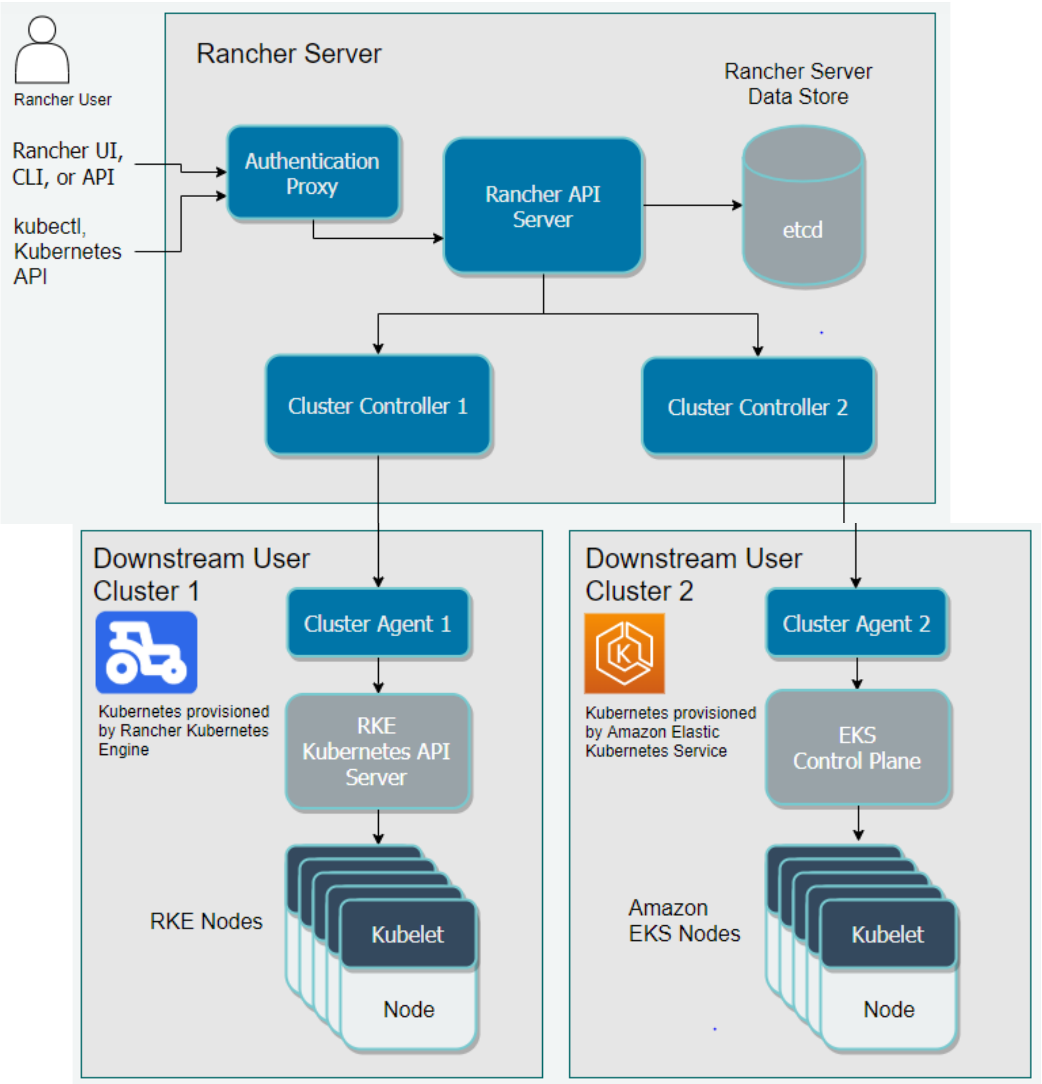
kubectl

- **Command line utility to interact with Kubernetes cluster**
  - Apply manifest to K8s using kubectl
  - %kubectl apply –f mypod.yaml
- **Introduction to YAML – creating Kubernetes deployment**
  - https://www.mirantis.com/blog/introduction-to-yaml-creating-a-kubernetes-deployment/

Rancher Kubernetes Engine (RKE)

- **Kubernetes is a rich and full featured and this makes it difficult to install and configure**
  - There are installers that simplify deploying raw K8s that provide their own API to abstract access to K8s
- **RKE, pronounced as “Rake”, is a lightweight Kubernetes installer for bare-metal and virtualized servers**
- **Also, RKE is a CNCF-certified Kubernetes distribution that runs entirely within Docker containers.**
- **Free and Open Source (FOSS) product by Rancher Labs**
- **Easy configuration and startup**
  - Specify node Ips and roles
  - Running cluster in minutes

Rancher Kubernetes Engine (RKE)



Helm

- **Package Manager for Kubernetes is called helm**
  - Helm helps you manage Kubernetes application
  - Using **Helm Charts,** you can define, install, and upgrade even the most complex Kubernetes application.
  - https://hub.helm.sh/
  - https://github.com/helm/charts

Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- **Log into the AWS Learner Lab**
- **Traverse to the EC2 Instance page.**
- **Launch two EC2 instances using the Ubuntu Server 20.24 AMI**
  - Ubuntu Server 20.04 LTS (HVM), SSD Volume Type
- **Select create a new security group and allow for inbound traffic from anywhere on ports 8080, 80, 443, and 22.**
- **Also, configure the outbound rule to allow all traffic.**

Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- **Connect to running EC2 instances using ssh**

  $ ssh -i <pem file name> ubuntu@<ec2 instance address>

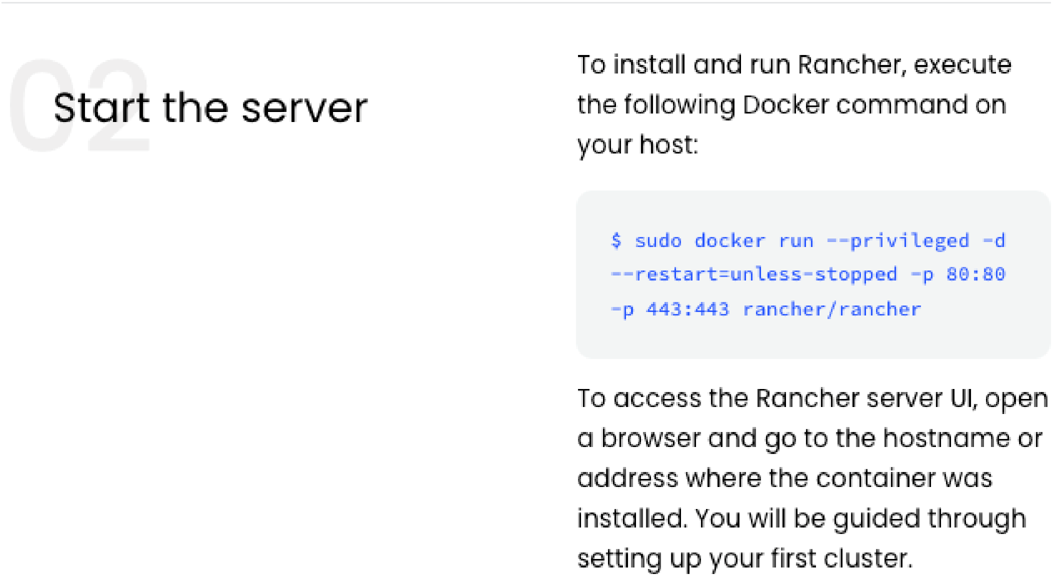
- **Install docker on both instances:**

  $ sudo apt-get update

  $ sudo apt install docker.io

Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- **Go to rancher.io; Click "Get Started"**
- **Scroll down a little and copy the command that is in the "Start Server" box.**
- **Then paste that command in your instance1 you ssh'd to.**
  - It will install the RancherUI so that you are able to access it.



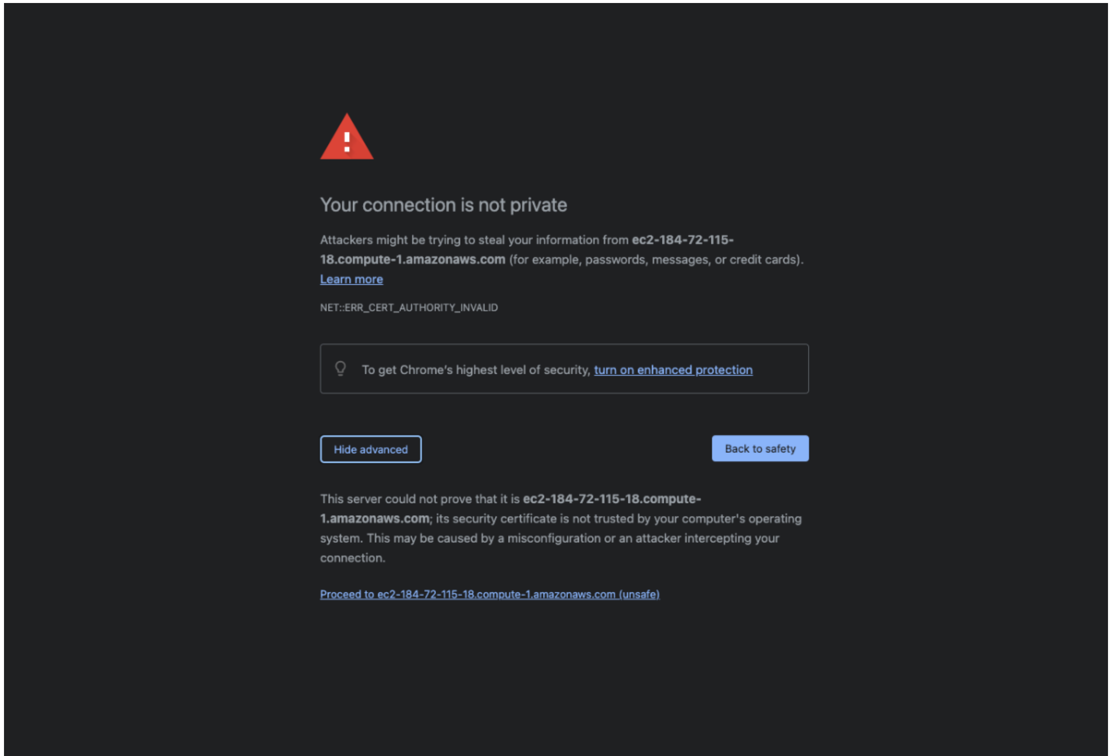
Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- **Once that install is complete, run “sudo docker ps” to view the current docker instance.**
  - Notice the container-ID as this will be important for an upcoming step.

```
ubuntu@ip-172-31-90-59:~$ sudo docker ps
CONTAINER ID   IMAGE             COMMAND             CREATED              STATUS
         PORTS
   NAMES
ee83a2ecd8bf   rancher/rancher   "entrypoint.sh"   About a minute ago   Up 57 se
conds   0.0.0.0:80->80/tcp, :::80->80/tcp, 0.0.0.0:443->443/tcp, :::443->443/tcp
   boring_rhodes
```

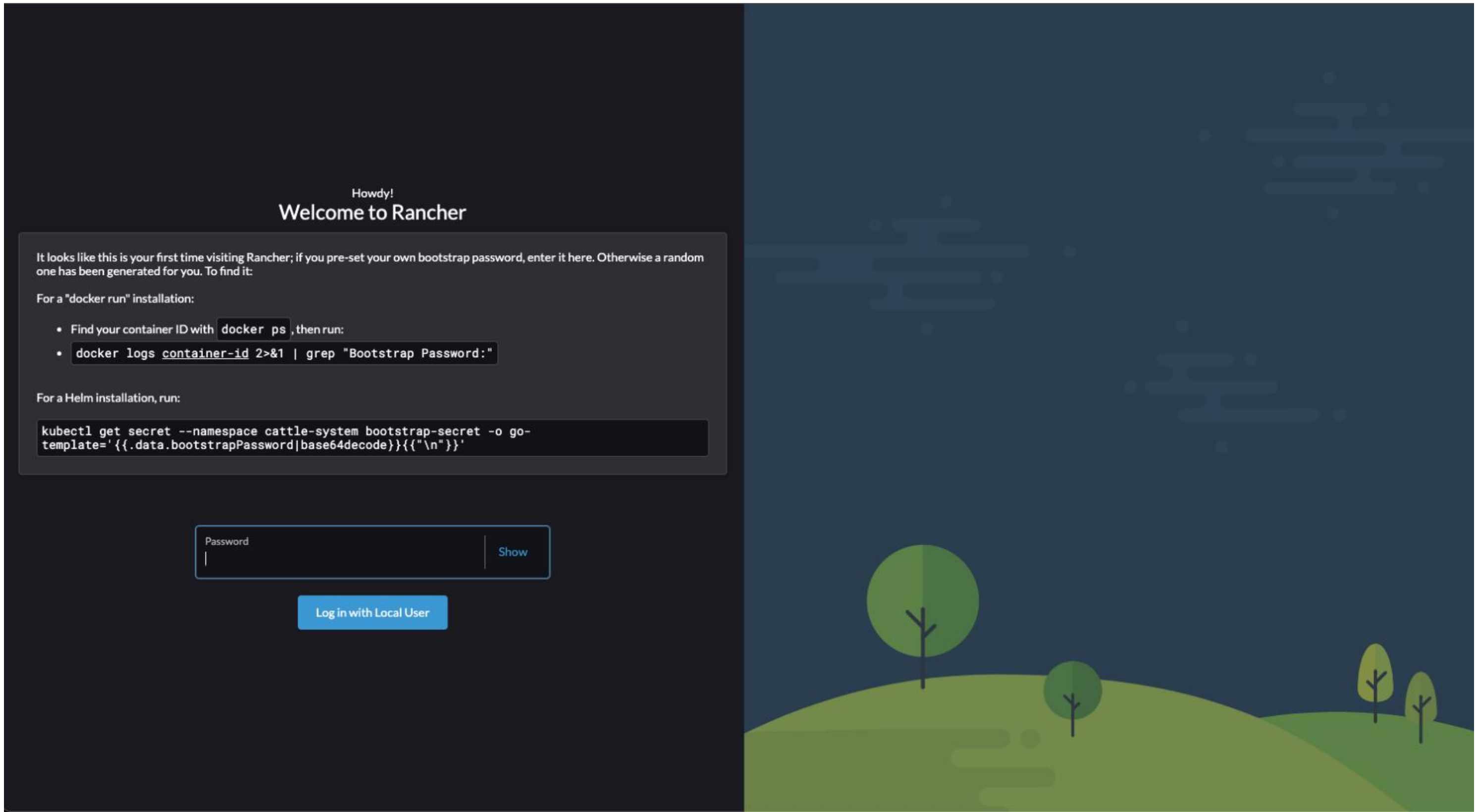
Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- **Now go to your instance1 that you connected to and click the public IPv4 DNS address.**
  - Once you click that, it will open a new tab and direct you here:



Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- **Click proceed, and now you are at the RancherUI**
- **Copy the password command on the UI page.**



Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- **Copy the password command on the UI page.**

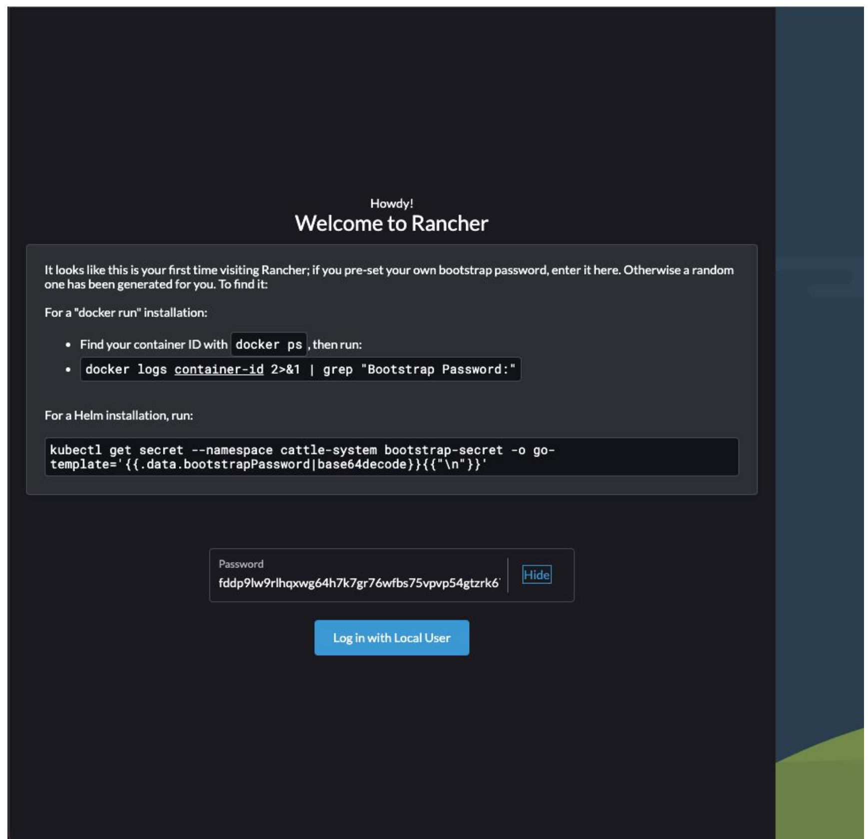
```
• docker logs container-id 2>&1 | grep "Bootstrap Password:"
```

- **Go back to your instance1 terminal and paste the command.**
  - Make sure to have sudo on the front of it.
  - Also, you need to replace the container-ID, and remember we got that from running sudo docker ps. This will give us the password to log into the RancherUI with.
  - Now take that password and paste it into the RancherUI webpage. Then log in with local user.

```
ubuntu@ip-172-31-90-59:~$ sudo docker logs ee83a2ecd8bf  2>&1 | grep "Bootstrap Password:"
2022/04/02 23:37:11 [INFO] Bootstrap Password: fddp9lw9rlhqxwg64h7k7gr76wfbs75vpvp54gtzrk6747xn687wg4
```

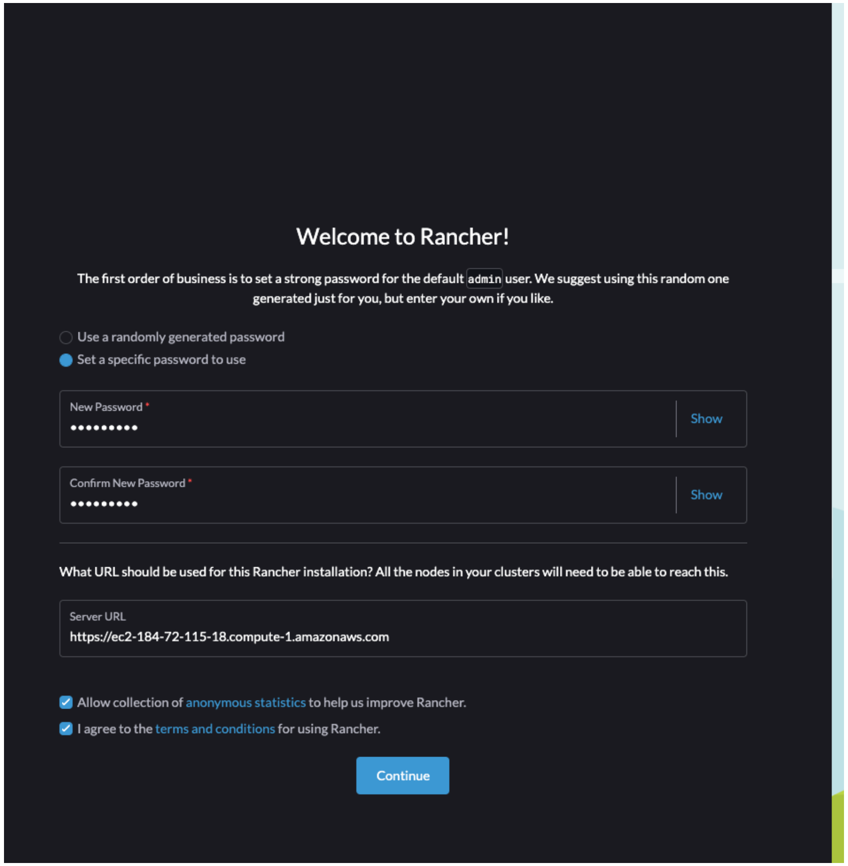
Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- **Now take that password and paste it into the RancherUI webpage.**
  - Then log in with local user.



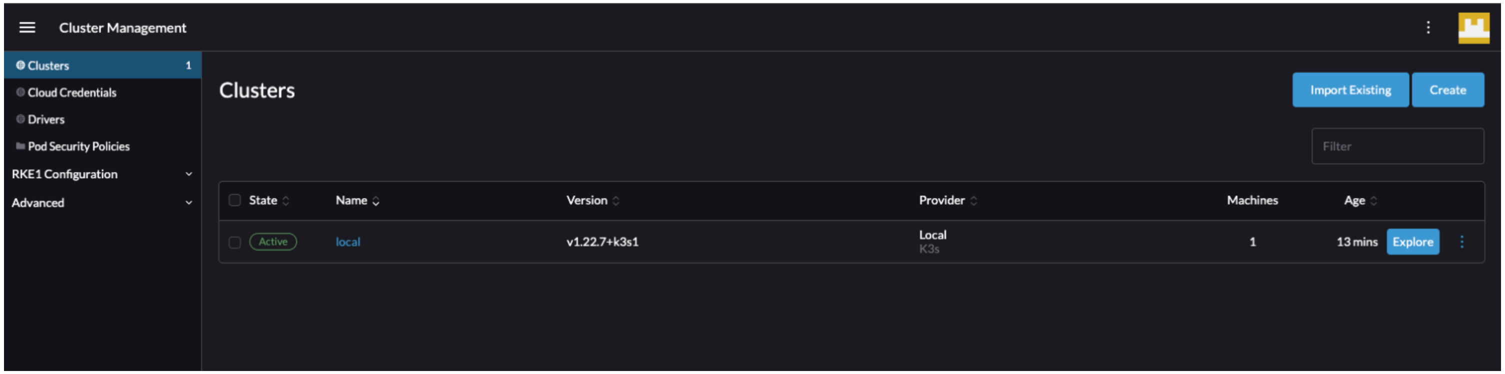
Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- **Once you log in, select the option "Set a specific password to use".**
  - Set it to something you will remember. Then hit continue.



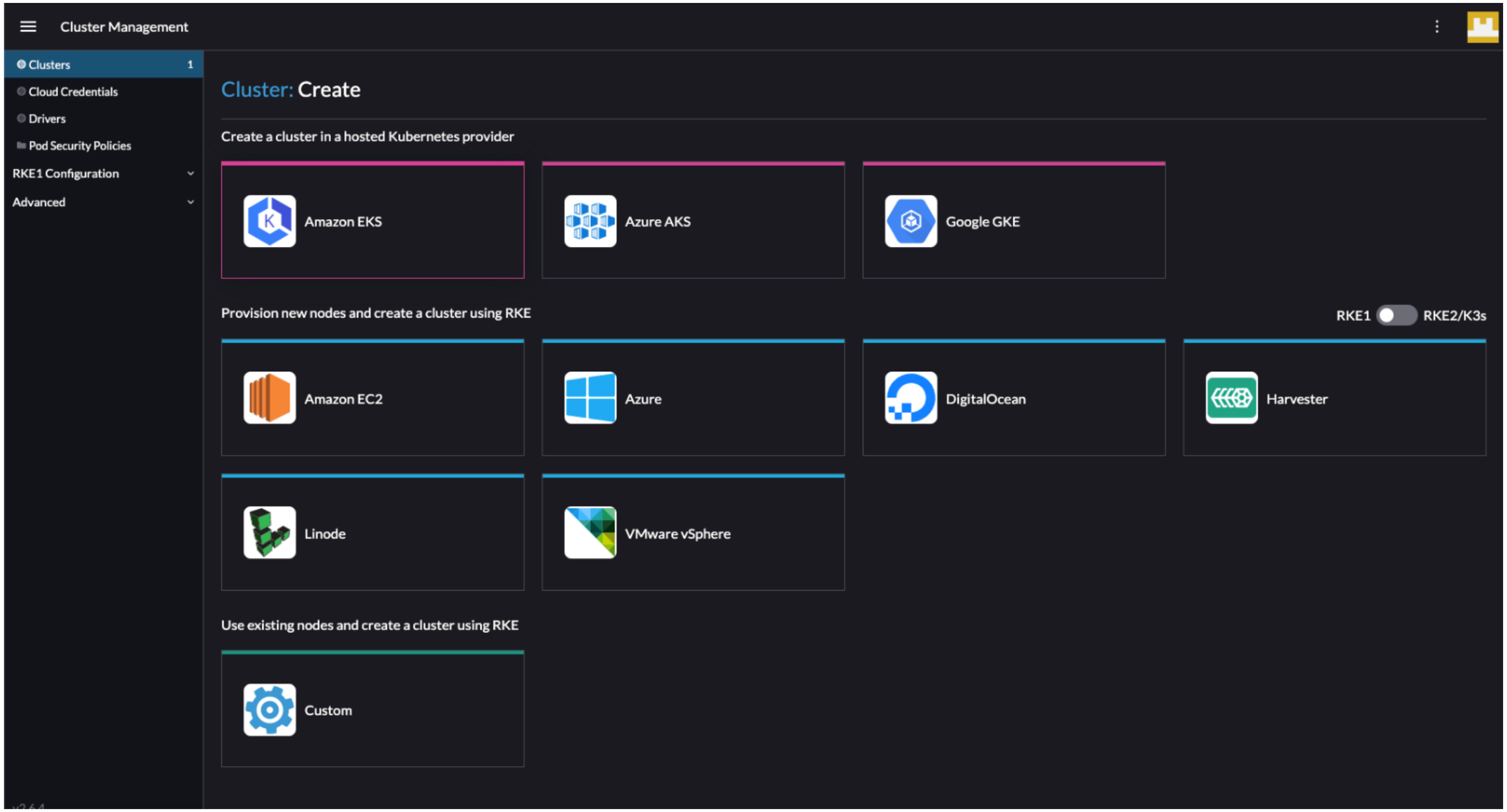
Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- **Once you are logged in, you'll see a cluster already created,**
  - however that won't support what you need and is there just to host this UI application.
- **We need to create a new cluster.**
  - Click the three lines up in the top left-hand corner.
  - Then click cluster management.
  - Then you should see something like this:
- **Click create**



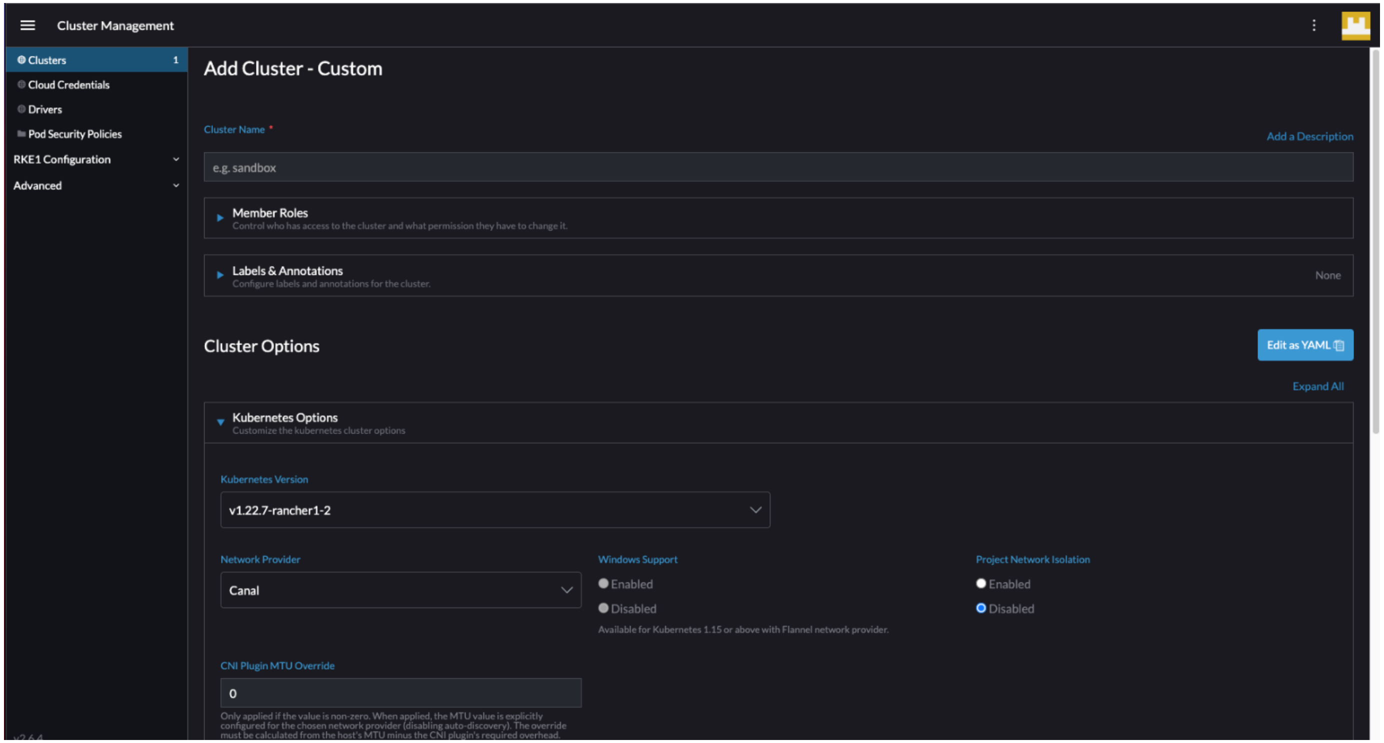
Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- Under "Use existing nodes and create a cluster using RKE", click custom.



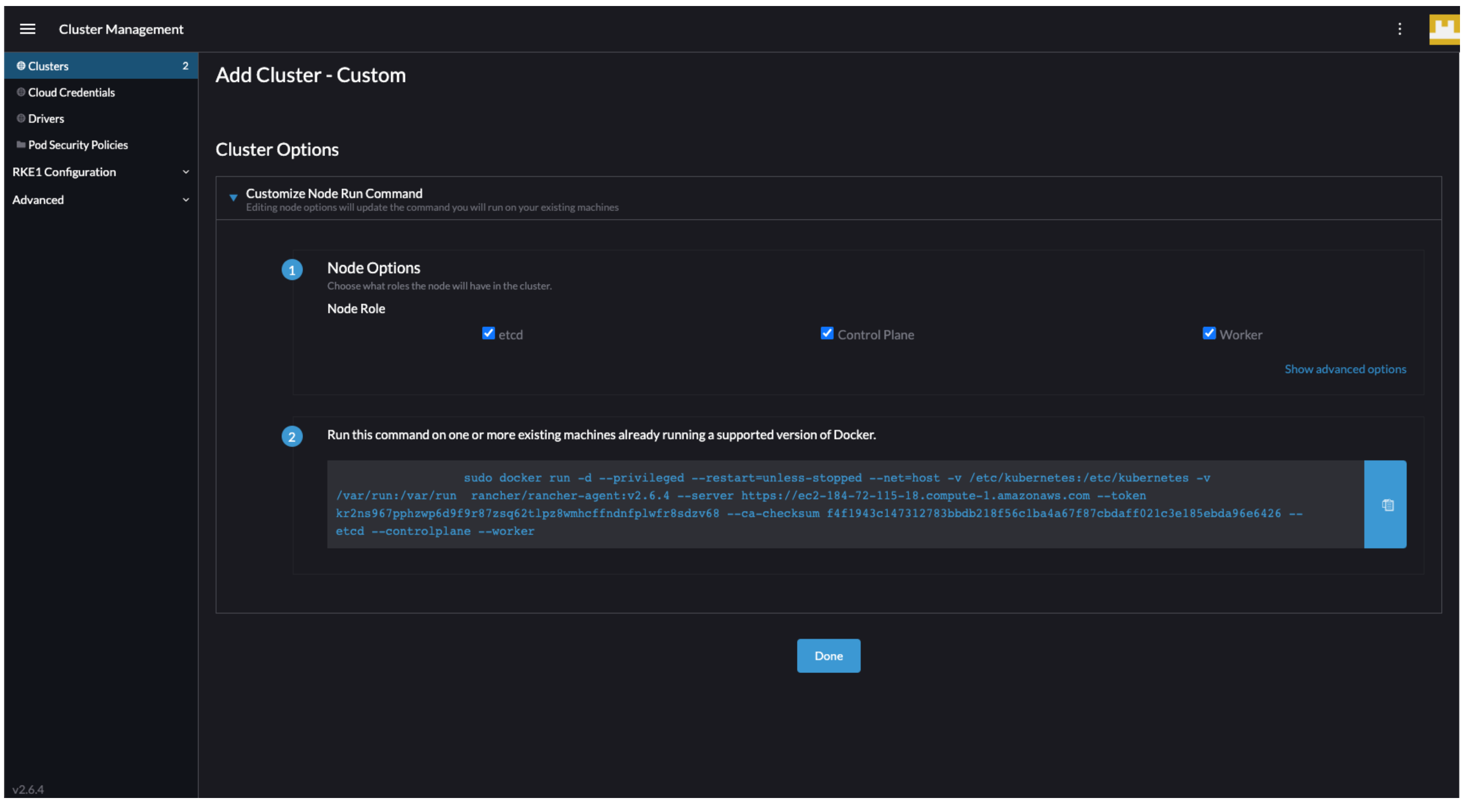
Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- **Once you click Custom, you'll see this page:**
  - Add a cluster name for the first page, and then hit next.



Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- Click the three checkboxes and copy that command it gives you



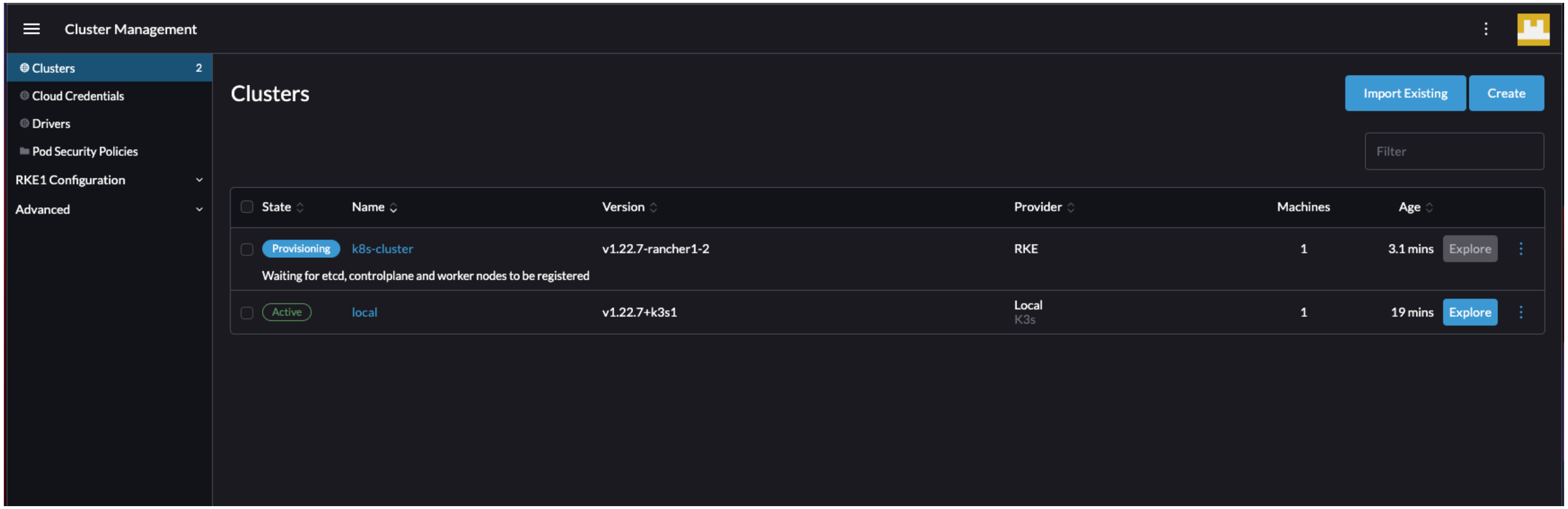
Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- **Copying that command, go to your terminal connected to instance2 and paste it within.**

```
ubuntu@ip-172-31-85-33:~$ sudo docker run -d --privileged --restart=unless-stopp
ed --net=host -v /etc/kubernetes:/etc/kubernetes -v /var/run:/var/run  rancher/r
ancher-agent:v2.6.4 --server https://ec2-184-72-115-18.compute-1.amazonaws.com -
-token kr2ns967pphzwp6d9f9r87zsq62tlpz8wmhcffndnfplwfr8sdzv68 --ca-checksum f4f1
943c147312783bbdb218f56c1ba4a67f87cbdaff021c3e185ebda96e6426 --etcd --controlpla
ne --worker
```

Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- **Once that is done running, click done in the RancherUI and we should see a cluster that is provisioning.**

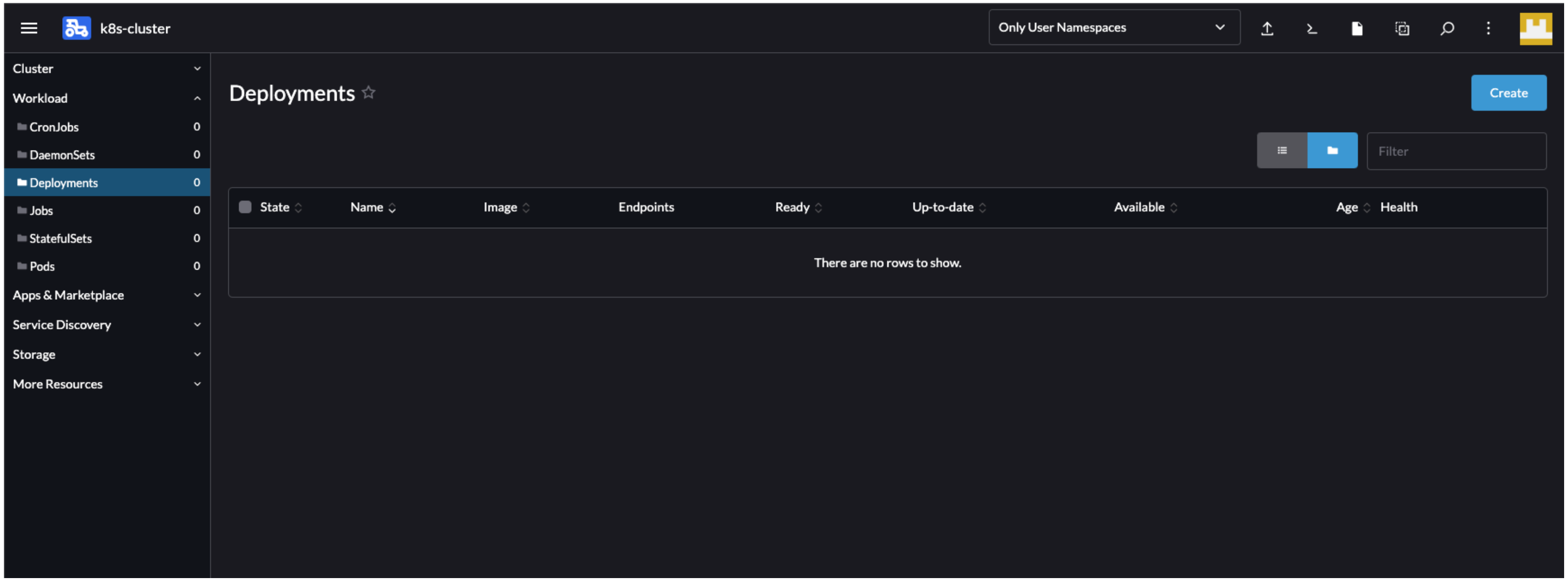


Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- Once a cluster is complete, we should see it as active

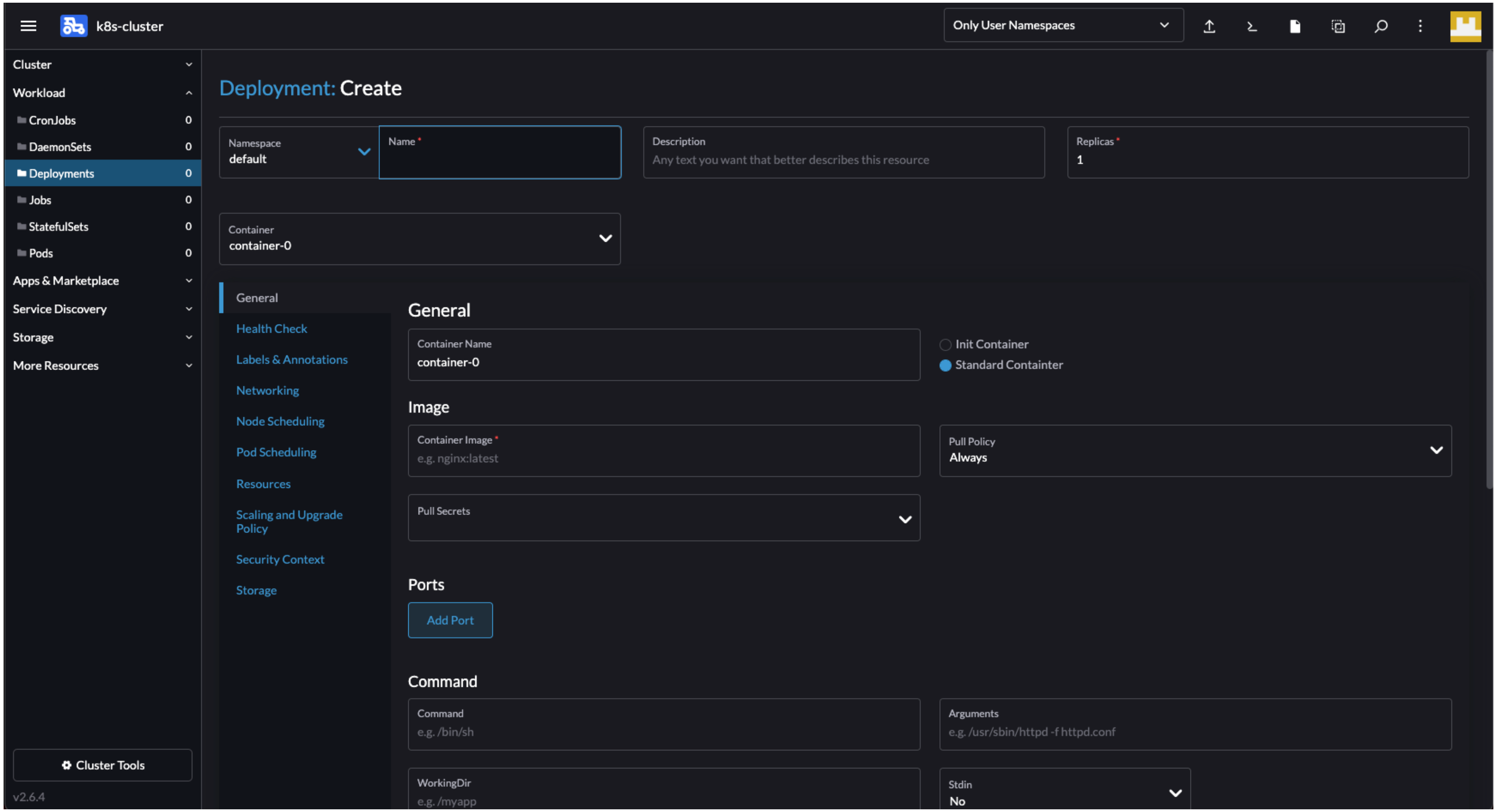
Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- Now we are ready to deploy!
- So, click the explore button on the cluster we created.
- Then click workload, then deployments. You should see this:



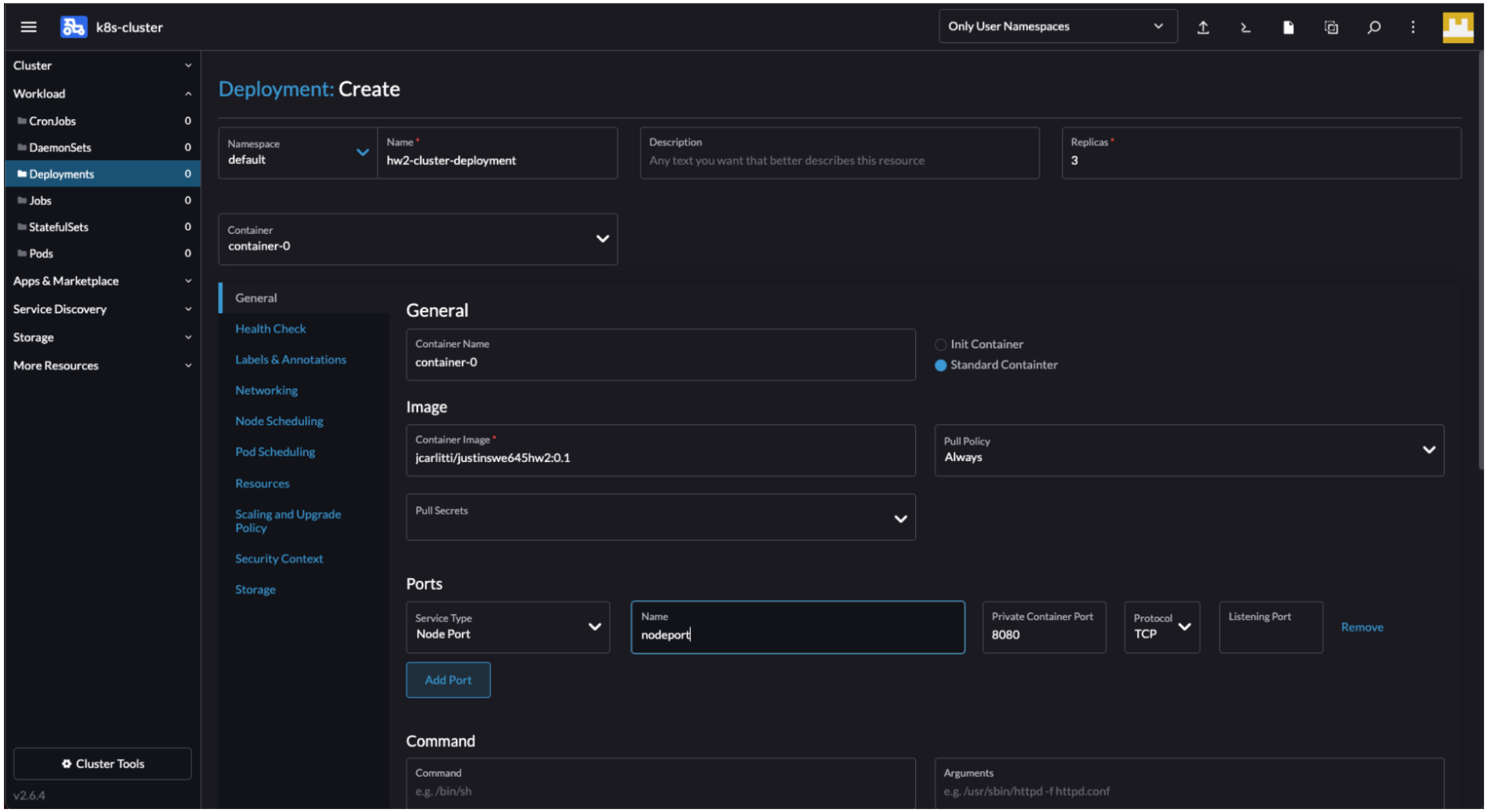
Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- Click create and you'll see this:



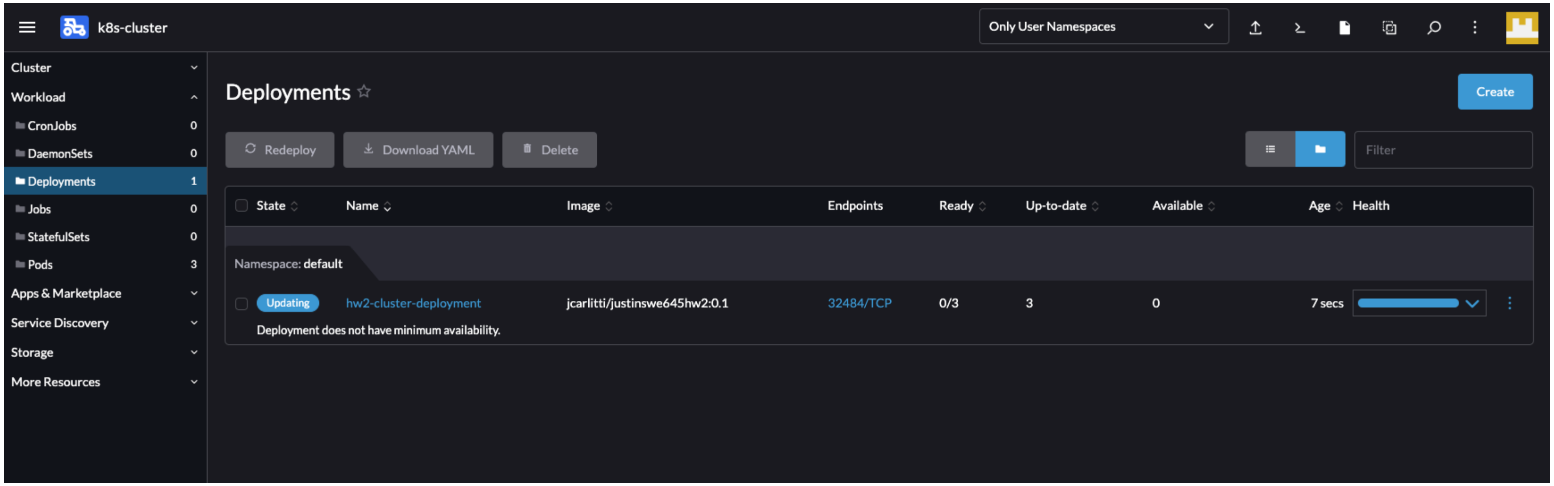
Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- **Filling in the name,**
  - Increase the replicas to 3,
  - Add the container Image (this is what you uploaded to docker hub), and add a port.
  - For the port service type, select NodePort.
  - Then give it a name and for the private container port, list 8080.
  - Leave the listening port blank as it will auto populate that with a port from the range 30000-32767.
  - Then hit create at the bottom.



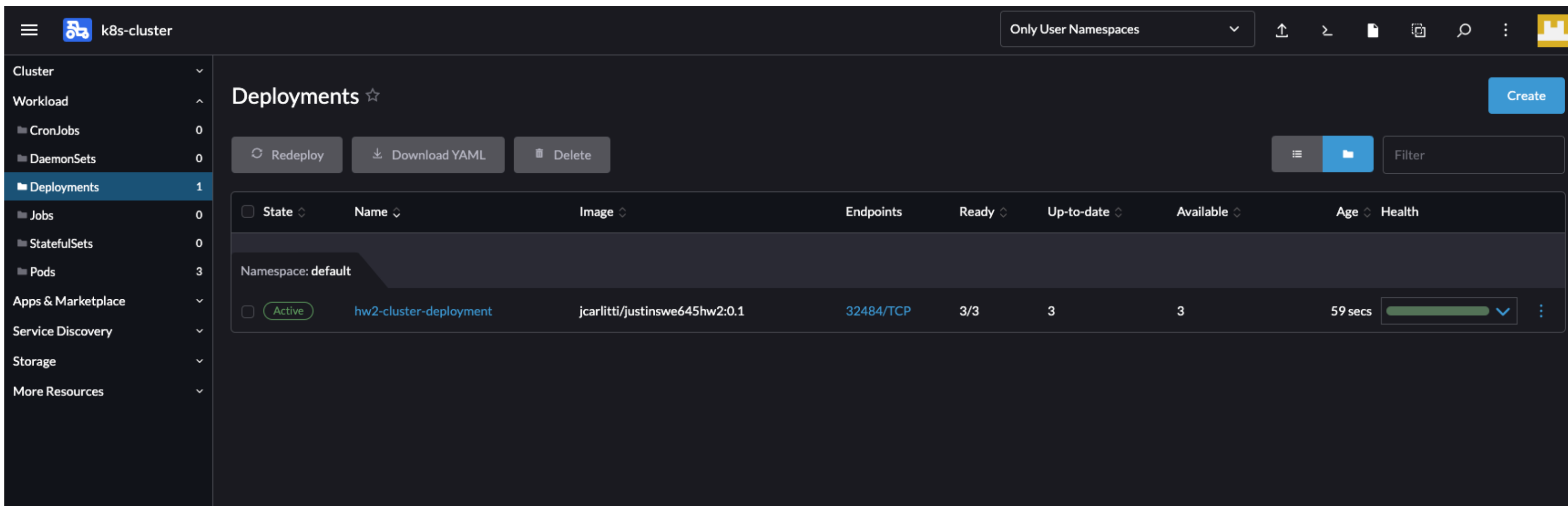
Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- **Then you'll see this.**
  – Wait for a little bit because it must configure everything.



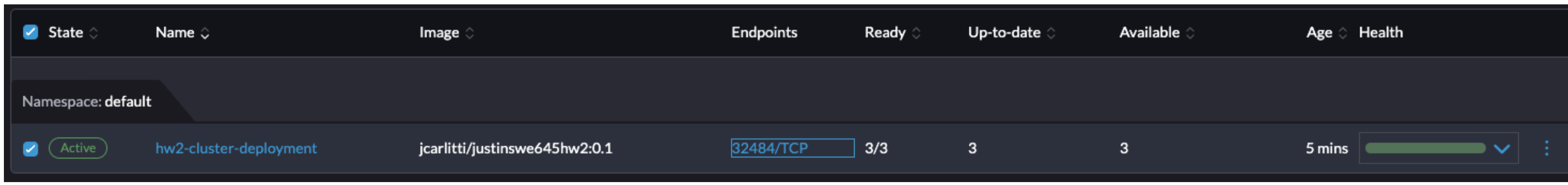
Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- Once it is complete, you'll see it as active.

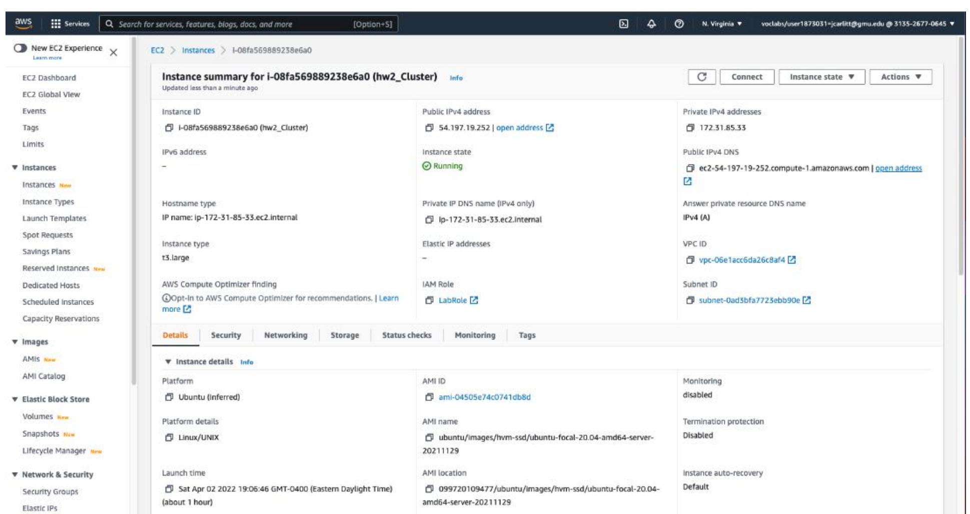


Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- Now to view your image, click the <port #>/tcp under the Endpoints column and it will take you to a new tab.



- Another way to view your newly deployed image is to click the Public IPv4 DNS address of the instance2 you created. (If you click the hyperlink, and it redirects you to a blank page, even if it is active, it is not ready yet. Give it some more time.).
- Make sure to add the highport number and the war file onto the URL path.
- Also make sure you are searching with HTTP, and not HTTPS.
  - AWS automatically goes to HTTPS, so change it to HTTP.



Setting up Rancher/Kubernetes cluster in AWS Learner Lab

- **When using nodeport the port must be specified in the URL as done above.**
- **For example**
  - http://ec2-3-92-55-50.compute-1.amazonaws.com:30752/SWE645-HW2/

Backup

Key Components: StatefulSet

- **Issues**
  - If we have replicas of the database Pod, they will all access and update the same data storage.
  - To avoid data inconsistencies, need a mechanism that manages which replicas are currently writing to that storage and which ones are reading from that storage
  - This can be addressed using StatefulSet objects in K8s
- **StatefulSets are valuable for applications that require one or more of the following.**
  - Stable, unique network identifiers.
  - Stable, persistent storage.
  - Ordered, graceful deployment and scaling.
  - Ordered, automated rolling updates.

Key Components: StatefulSet

- **StatefulSet objects are used for managing stateful applications**
- **StatefulSet considers each Pod instance as unique and ensures ordering of application pods in which they are created and deleted,**
  - although Pods deployed using a StatefulSet use the same Pod specification.
  - Default deployment scheme in StatefulSet is sequential starting with 0, such as app-0, app-1, app-2 etc.
    - The last one created is the first one to be deleted
  - To track each Pod as a unique object, the controller uses identity composed of stable storage, stable network identity, and an ordinal.
  - This identity remains with the node regardless to which node the Pod is running on at any one time.
- **The next Pod will not launch until the current Pod reaches a running and ready state.**
  - Pods in StatefulSet are not deployed in parallel.

StatefulSet vs. Deployment

StatefulSet	Deployment
Used for managing stateful applications	Used for managing stateless applications
StatefulSet considers each pod unique	Deployment considers each pod identical
StatefulSet ensures ordering of application pods in which they are created and deleted	Ordering does not matter since all pods are equivalent
Default deployment scheme in StatefulSet is sequential starting with 0, such as app-0, app-1, app-2 etc.	Deployment pods can be deployed simultaneously. The deployment scheme is sequential – pod names end with randomly generated content.

Key Components: StatefulSet

- A blueprint for managing stateful applications
- Kind of applications suitable for StatefulSet involve persistent storage, such as
  - MySQL,
  - MongoDB,
  - Elasticsearch,
  - Kafka
- Just like Deployment, StatefulSet would take care of replicating the Pods and scaling them up or down, but in addition it ensures database reads and writes are synchronized, so that no database inconsistencies happen.

Key Components: StatefulSet

- **Manages the deployment and scaling of a set of Pods, and provides guarantees about the ordering and uniqueness of these Pods.**
  - Like a Deployment, a StatefulSet manages Pods that are based on an identical container spec.
  - Unlike a Deployment, a StatefulSet maintains a sticky identity for each of their Pods.
- **These pods are created from the same spec, but are not interchangeable:**
  - each has a persistent identifier that it maintains across any rescheduling.

Key Components: StatefulSet

- If you want to use storage volumes to provide persistence for your workload, you can use a StatefulSet as part of the solution.
- Although individual Pods in a StatefulSet are susceptible to failure, the persistent Pod identifiers make it easier to match existing volumes to the new Pods that replace any that have failed.

Key Components: StatefulSet

- **Additional Characteristics**
  - The storage for a given Pod must either be provisioned by a PersistentVolume provisioner based on the requested storage class, or pre-provisioned by an admin.
  - Deleting and/or scaling down a StatefulSet will not delete the volumes associated with the StatefulSet.
    - This is done to ensure data safety.
  - StatefulSets currently require a Headless Service to be responsible for the network identity of the Pods.
    - You are responsible for creating this Service.
  - StatefulSets do not provide any guarantees on the termination of pods when a StatefulSet is deleted.

Key Components: StatefulSet

- **How to create StatefulSet**
  - Like everything else in Kubernetes, StatefulSets can be configured using an YAML file.
  - StatefulSets require a headless service, which can be created in the same manifest file.

Key Components: StatefulSet

- **StatefulSet YAML example**
  – including headless service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: nginx
---
```

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx # has to match .spec.template.metadata.lab
  serviceName: "nginx"
  replicas: 3 # by default is 1
  template:
    metadata:
      labels:
        app: nginx # has to match .spec.selector.matchLabel
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "my-storage-class"
      resources:
        requests:
          storage: 1Gi
```

Another example: Deploying Cassandra with a StatefulSet

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: cassandra
  name: cassandra
spec:
  clusterIP: None
  ports:
  - port: 9042
  selector:
    app: cassandra
```

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: cassandra
  labels:
    app: cassandra
spec:
  serviceName: cassandra
  replicas: 3
  selector:
    matchLabels:
      app: cassandra
  template:
    metadata:
      labels:
        app: cassandra
    spec:
      terminationGracePeriodSeconds: 1800
```

```yaml
containers:
- name: cassandra
  image: gcr.io/google-samples/cassandra:v13
  imagePullPolicy: Always
  ports:
  - containerPort: 7000
    name: intra-node
  - containerPort: 7001
    name: tls-intra-node
  - containerPort: 7199
    name: jmx
  - containerPort: 9042
    name: cql
```

Sources: https://kubernetes.io/docs/tutorials/stateful-application/cassandra/

Another example: Deploying Cassandra with a StatefulSet

```yaml
resources:
  limits:
    cpu: "500m"
    memory: 1Gi
  requests:
    cpu: "500m"
    memory: 1Gi
securityContext:
  capabilities:
    add:
      - IPC_LOCK
lifecycle:
  preStop:
    exec:
      command:
      - /bin/sh
      - -c
      - nodetool drain
```

```yaml
env:
  - name: MAX_HEAP_SIZE
    value: 512M
  - name: HEAP_NEWSIZE
    value: 100M
  - name: CASSANDRA_SEEDS
    value: "cassandra-0.cassandra.default.svc.cluster.local"
  - name: CASSANDRA_CLUSTER_NAME
    value: "K8Demo"
  - name: CASSANDRA_DC
    value: "DC1-K8Demo"
  - name: CASSANDRA_RACK
    value: "Rack1-K8Demo"
  - name: POD_IP
    valueFrom:
      fieldRef:
        fieldPath: status.podIP
```

Example: Deploying Cassandra with a StatefulSet

```yaml
readinessProbe:
  exec:
    command:
    - /bin/bash
    - -c
    - /ready-probe.sh
  initialDelaySeconds: 15
  timeoutSeconds: 5
# These volume mounts are persistent. They are like inline claims,
# but not exactly because the names need to match exactly one of
# the stateful pod volumes.
volumeMounts:
- name: cassandra-data
  mountPath: /cassandra_data
```

Example: Deploying Cassandra with a StatefulSet

```yaml
  # These are converted to volume claims by the controller
  # and mounted at the paths mentioned above.
  # do not use these in production until ssd GCEPersistentDisk or other ssd pd
  volumeClaimTemplates:
  - metadata:
      name: cassandra-data
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: fast
      resources:
        requests:
          storage: 1Gi
---
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: fast
provisioner: k8s.io/minikube-hostpath
parameters:
  type: pd-ssd
```