Technical Workshop - SUSE CaaS Platform 4 Beta

-Workbook-

Course ID: Technical Workshop – SUSE CaaSP
Version: RC1
Date: 2019-08-21

Proprietary Statement

SUSE LLC, has intellectual property rights relating to technology embodied in the product that is described in this document.

SUSE
Maxfeldstrasse 5
90409 Nuremberg
Germany
www.suse.com

If you know of illegal copying of software, contact your local Software Antipiracy Hotline.

Disclaimer

SUSE LLC, makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose.

Further, SUSE LLC, reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. Further, SUSE LLC, makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, SUSE LLC, reserves the right to make changes to any and all parts of SUSE software, at any time, without any obligation to notify any person or entity of such changes.

Any products or technical information provided under this Agreement may be subject to U.S. export controls and the trade laws of other countries. You agree to comply with all export control regulations and to obtain any required licenses or classification to export, re-export or import deliverables. You agree not to export or re-export to entities on the current U.S. export exclusion lists or to any embargoed or terrorist countries as specified in the U.S. export laws. You agree to not use deliverables for prohibited nuclear, missile, or chemical biological weaponry end uses. SUSE assumes no responsibility for your failure to obtain any necessary export approvals.

This SUSE Training Manual is published solely to instruct students in the use of SUSE networking software. Although third-party application software packages may be used in SUSE training courses, this is for demonstration purposes only and shall not constitute an endorsement of any of these software applications.

Further, SUSE LLC does not represent itself as having any particular expertise in these application software packages and any use by students of the same shall be done at the student’s own risk.

Table of Contents

Documentation Conventions:

The following typographical conventions are used in this manual:

Bold	Represents things you should pay attention to or buttons you click, text or options that you should click/select/type in a GUI.
Bold Gray	Represents the name of a Task or in the context of what is seen on the screen, the screen name, a tab name, column name, field name, etc.
Bold Red	Represents warnings or very important information.
Option > Option > Option	Represents a chain of items selected from a menu.
BOLD_UPPERCASE_ITALIC	Represents an “exercise variable” that you replace with another value.
bold monospace	Represents text displayed in a terminal or entered in a file.
bold monospace blue	Represents commands entered at the command line.
bold monospace green	Represents a file name.

1 Setup Workstation to Initialize Cluster

Description:

You will setup a workstation to initialize the cluster

1- 1 Start the Lab VMs

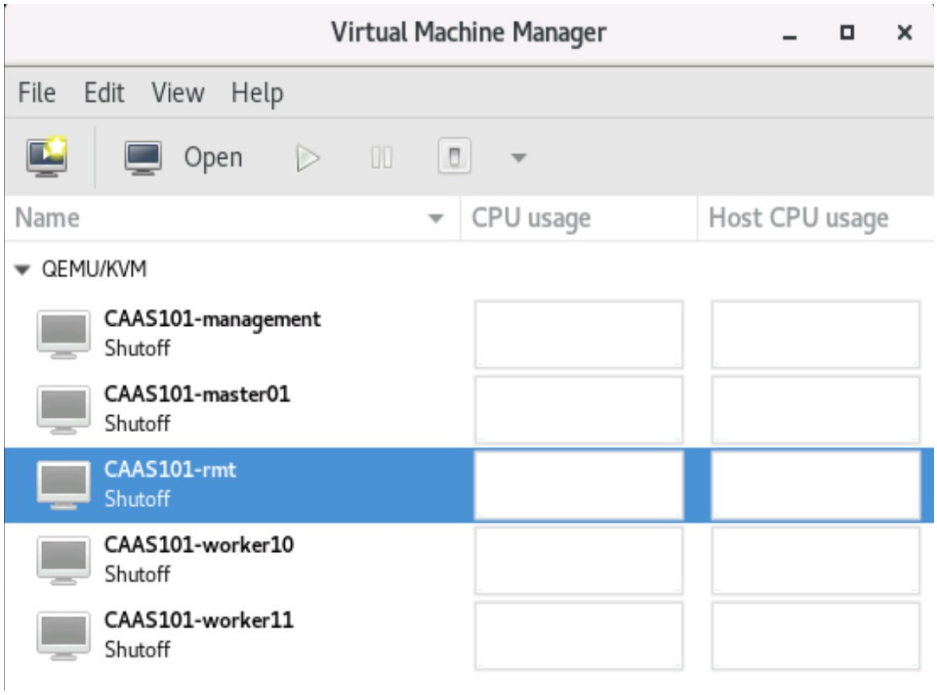
Description:

In this exercise, you use the Virt-Manager utility to start the lab VM(s) in the proper order.

Task 1: Start the Lab VMs

1. On the **VM host**, launch the Virt-Manager utility
   (4th icon from the left on the dock at the bottom of the screen)
   You should see the course VMs listed.

2. Right-click on the **CAAS101-rmt** VM and select **Run**

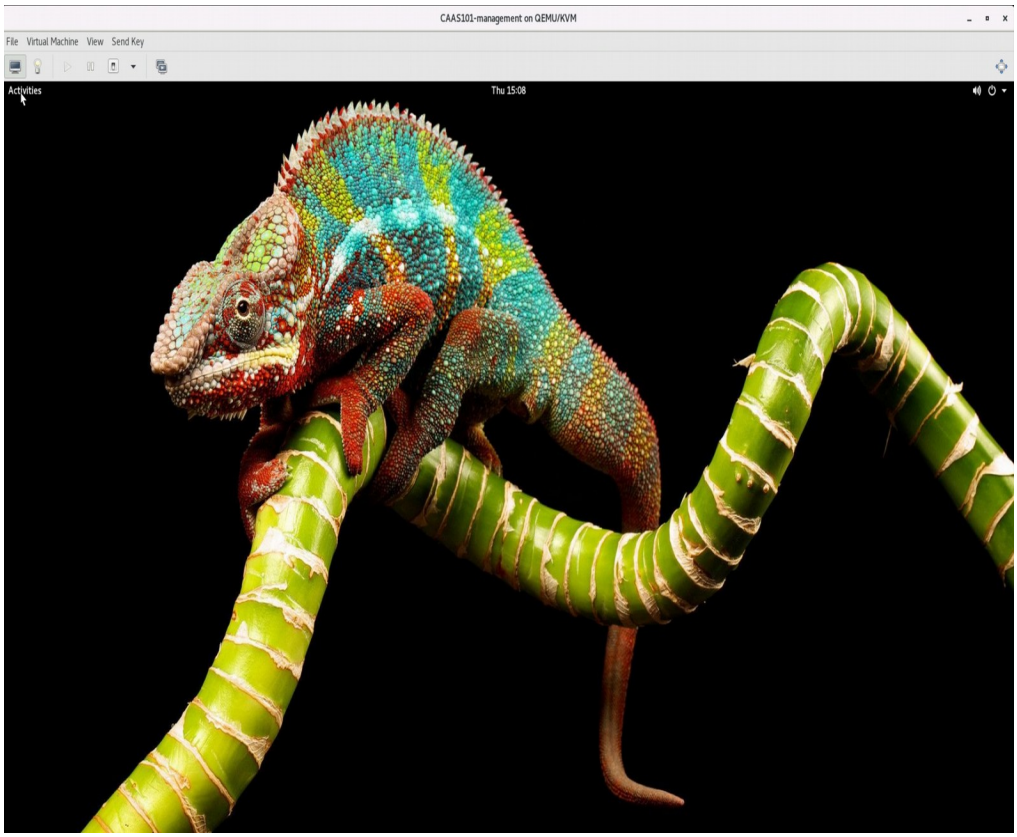


3. Double-click on the **CAAS101-rmt** VM to view its console
   When the **CAAS101-rmt** VM has finished booting completely, close its console window.
   (You can use the CPU utilization graph in Virt-Manager to see when the node is finished booting and starting services).

4. Power on the reset of the Virtual Machines **CAAS101-management, CAAS101-master, CAAS101-worker10 and CAAS101-worker10**

   *note you can power all of the on at the same time

   When the **Workstation** has finished booting you will use it as your management workstation for all tasks including managing VMs with the Virt-Manager utility.
5. Make sure you double click on the **CAAS101-management** so you view it full screen
6. Open a Terminal session by clicking on the 3rd icon down



Summary:

In this exercise, you started the required lab VM(s) in the proper order.

(End of Exercise)

1- 2 Prepare the Management Workstation

Description:

In this exercise, you prepare the Management Workstation for the CaaS Platform installation.

Dependencies:

The RMT server must be configured (Optional Docker Registry)

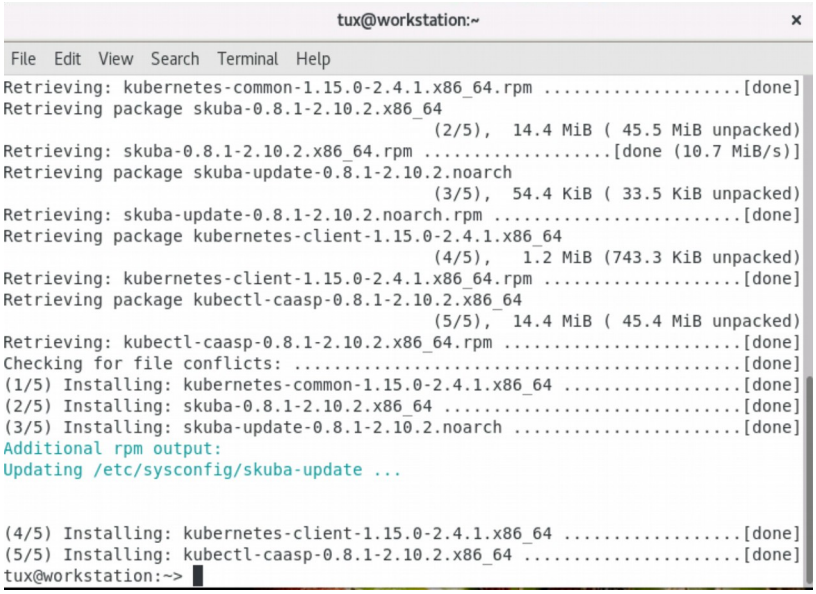
DHCP, PXE and DNS must be configured.

Task 1: Install needed software on Management Workstation

1. On the **Workstation**, open a terminal
2. Enter the following command to install skuba and the kubernetes client

   sudo zypper in -t pattern SUSE-CaaSP-Management

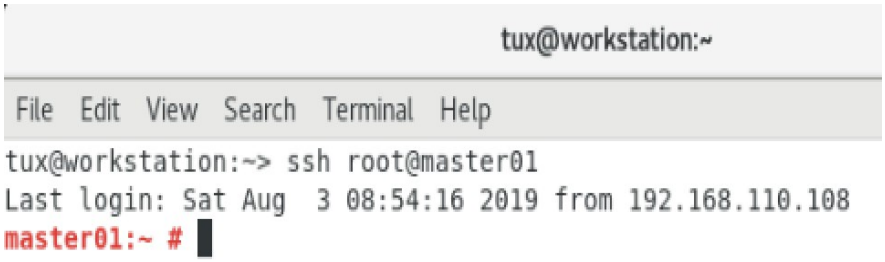
   It will tell you the amount of additional drive space needed and it will prompt you to press **y** to continue
3. Press the **'Y'** key to continue
   when it finished you should see a screen similar to the image below



Task 2: Test to ensure SSH keys are properly set

1. On the **Workstation**, open a terminal
2. Enter the following command in the terminal to test your **ssh keys** to master01

   ssh root@master01
3. You should see a red prompt that says master01



4. Notice that you did not need to type in a password or accept a key?
   This is because of the pre-work done on the workstation
5. Exit your ssh session from the terminal prompt
   exit

6. Repeat the step 2 and 3 for **worker10** and **worker11**

Summary:

In this exercise, you installed Management workstation utilities. You also tested connectivity to all of the nodes.

(End of Exercise)

1- 3 Setup SSH Keys (optional)

Description:

In this exercise, you will setup all nodes to ssh using key authentication

This is an **OPTIONAL** exercise to be performed only if you do not have SSH setup on your workstation intended to manage the cluster installation

Task 1: Open a terminal

1. On the **Workstation**, open a terminal

Task 2: Create SSH key (This has been done – information only)

1. On the **Workstation**, in the terminal opened previously
   Enter the following command to create a ssh key
   ssh-keygen
   When it asks you questions just take the defaults

Task 3: Add the SSH key to the agent

1. On the **Workstation**, in the terminal opened previously
2. Enter the following command in the terminal to add your **ssh key** to ssh-agent

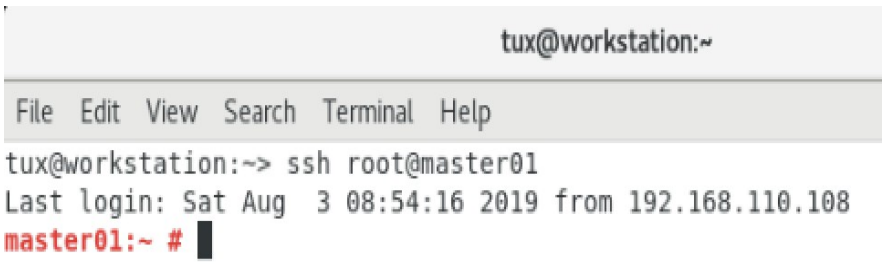
   ssh-add

Task 4: Test to ensure SSH keys are properly set

1. On the **Workstation**, open a terminal
2. Enter the following command in the terminal to test your **ssh keys** to master01

   ssh root@master01

3. You should see a red prompt: **master01:~ #**



4. Notice that you did not need to type in a password or accept a key?

This is because of the pre-work done on the workstation

5. Exit your ssh session from the terminal prompt

   exit

6. Repeat step 2 and 3 for **worker10** and **worker11**

Summary:

In this exercise, you set up and verified the SSH key to ssh from the Management Workstation to any of the nodes using key authentication.

(End of Exercise)

1- 4 Install CaaS Platform Nodes

Description:

In this exercise, you install SUSE CaaS Platform all nodes.

Dependencies:

The RMT server must be configured (Optional Docker Registry)

DHCP, PXE and DNS must be configured.

Master, and 2 worker nodes with SLES 15 SP1 installed and CaaS Platform Repos added

Task 1: Initialize the and bootstrap the cluster

1. On the **Workstation**, open a terminal
2. Enter the following command create the basic config to our **kubernetes cluster**

   skuba cluster init --control-plane master.example.com my-cluster

   ```
   tux@workstation:~> skuba cluster init --control-plane master.example.com my-cluster
   ** This is a BETA release and NOT intended for production usage. **
   [init] configuration files written to /home/tux/my-cluster
   tux@workstation:~>
   ```

3. Enter the following command to change into the newly created folder

   cd my-cluster

4. Look at the cluster config filename **my-cluster**

   less kubeadm-init.conf

```
tux@workstation:~/my-cluster> cat kubeadm-init.conf
apiVersion: kubeadm.k8s.io/v1beta1
kind: InitConfiguration
bootstrapTokens: []
localAPIEndpoint:
  advertiseAddress: ""
---
apiVersion: kubeadm.k8s.io/v1beta1
kind: ClusterConfiguration
apiServer:
  certSANs:
    - master.example.com
  extraArgs:
    oidc-issuer-url: https://master.example.com:32000
    oidc-client-id: oidc
    oidc-ca-file: /etc/kubernetes/pki/ca.crt
    oidc-username-claim: email
    oidc-groups-claim: groups
clusterName: my-cluster
controlPlaneEndpoint: master.example.com:6443
dns:
  imageRepository: registry.suse.com/caasp/v4
  imageTag: 1.3.1
  type: CoreDNS
etcd:
  local:
    imageRepository: registry.suse.com/caasp/v4
    imageTag: 3.3.11
imageRepository: registry.suse.com/caasp/v4
kubernetesVersion: 1.15.0
networking:
  podSubnet: 10.244.0.0/16
  serviceSubnet: 10.96.0.0/12
useHyperKubeImage: true
tux@workstation:~/my-cluster>
```

5. Install, configure, and bootstrap the first node as the master

   From the **~/my_cluster** directory:

   skuba node bootstrap --target master.example.com master01

```
tux@workstation:~/my-cluster> skuba node bootstrap --target master.example.com master01
** This is a BETA release and NOT intended for production usage. **
W0808 17:13:25.590213    2588 ssh.go:306]
The authenticity of host '192.168.110.7:22' can't be established.
ECDSA key fingerprint is 54:3e:9b:2e:21:fc:5c:70:ff:c4:99:7c:fa:b4:08:d0.
I0808 17:13:25.590316    2588 ssh.go:307] accepting SSH key for "master.example.com:22"
I0808 17:13:25.590327    2588 ssh.go:308] adding fingerprint for "master.example.com:22" to "known_hosts"
[bootstrap] updating init configuration with target information
[bootstrap] writing init configuration for node
[bootstrap] applying init configuration to node
[bootstrap] downloading secrets from bootstrapped node "master.example.com"
[bootstrap] successfully bootstrapped node "master.example.com" with Kubernetes: "1.15.0"
tux@workstation:~/my-cluster>
```

Task 2: Deploy the Worker nodes

1. Add a **worker** node to the cluster
   From the **~/my_cluster** directory:
   skuba node join --role worker --target worker10.example.com worker10

```
tux@workstation:~/my-cluster> skuba node join --role worker --target worker10.example.com worker10
** This is a BETA release and NOT intended for production usage. **
W0808 17:23:50.580677    2632 ssh.go:306]
The authenticity of host '192.168.110.10:22' can't be established.
ECDSA key fingerprint is 19:7c:db:38:d2:c5:34:f6:cb:90:21:4e:1a:b5:5e:d7.
I0808 17:23:50.580863    2632 ssh.go:307] accepting SSH key for "worker10.example.com:22"
I0808 17:23:50.580916    2632 ssh.go:308] adding fingerprint for "worker10.example.com:22" to "known_hosts"
[join] applying states to new node
[join] node successfully joined the cluster
tux@workstation:~/my-cluster>
```

2. Add a second **worker** node to the cluster

   skuba node join --role worker --target worker11.example.com worker11

   You should see a message very similar to the message you received in the previous step.

3. Verify cluster status
   skuba cluster status

```
tux@workstation:~/my-cluster> skuba cluster status
** This is a BETA release and NOT intended for production usage. **
NAME       OS-IMAGE                              KERNEL-VERSION           KUBELET-VERSION   CONTAINER-RUNTIME   HAS-UPDATES   HAS-DISRUPTIVE-UPDATES
master01   SUSE Linux Enterprise Server 15 SP1   4.12.14-197.10-default   v1.15.0           cri-o://1.15.0      <none>        <none>
worker10   SUSE Linux Enterprise Server 15 SP1   4.12.14-197.10-default   v1.15.0           cri-o://1.15.0      <none>        <none>
worker11   SUSE Linux Enterprise Server 15 SP1   4.12.14-197.10-default   v1.15.0           cri-o://1.15.0      <none>        <none>
tux@workstation:~/my-cluster>
```

Summary:

In this exercise, you installed Skuba and the Kubernetes Client Utilities. You also tested that the SSH keys were set up properly and you could seamlessly connect to all of the nodes.

(End of Exercise)

2 Connecting to the Cluster

Description:

In this section you will learn how to connect to the Cluster via both the command line and a GUI interface.

2- 1 Use the kubectl Command to Display Info About the Cluster

---

Description:

In this exercise, you use the **kubectl** command to display info about the Kubernetes cluster.

Task 1: Configure workstation for kubectl command

1. **On** the **Workstation**, **open** a **terminal**
2. Create official folder for kubernetes config filename

   mkdir ~/.kube

3. **On** the **Workstation**, **open** a **terminal**
4. Create config file

   cp ~/my-cluster/admin.conf ~/.kube/config

Task 2: Use the kubectl Command

1. From a terminal, enter the following command to view the URL of the Kubernetes master:

   kubectl cluster-info

   You should see the URL of the Kubernetes master displayed.
2. Enter the following command to see a more detailed output of the status of the cluster:

   kubectl cluster-info dump

   You should see a json dump of the status of cluster.
3. Enter the following command to display a list of Kubernetes nodes:

   kubectl get nodes

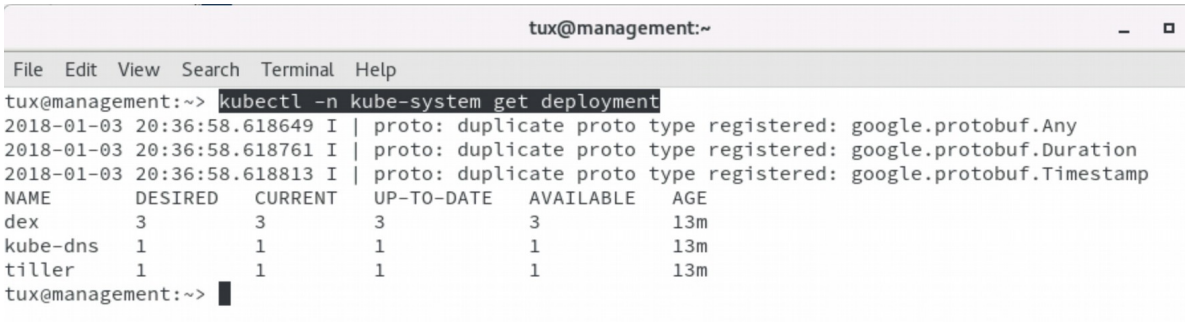
   You should see the list of nodes, their node names, status, age and version displayed.

```
tux@management:~/course_files/CAAS101/scripts> kubectl get nodes
2018-08-08 10:02:50.924418 I | proto: duplicate proto type registered: google.protobuf.Any
2018-08-08 10:02:50.924462 I | proto: duplicate proto type registered: google.protobuf.Duration
2018-08-08 10:02:50.924473 I | proto: duplicate proto type registered: google.protobuf.Timestamp
NAME       STATUS    AGE       VERSION
master01   Ready     1d        v1.9.8
master02   Ready     1d        v1.9.8
master03   Ready     1d        v1.9.8
worker10   Ready     1d        v1.9.8
worker11   Ready     1d        v1.9.8
worker12   Ready     1d        v1.9.8
tux@management:~/course_files/CAAS101/scripts>
```

4. Enter the following command to display a list of Kubernetes nodes:

   kubectl -n kube-system get deployments

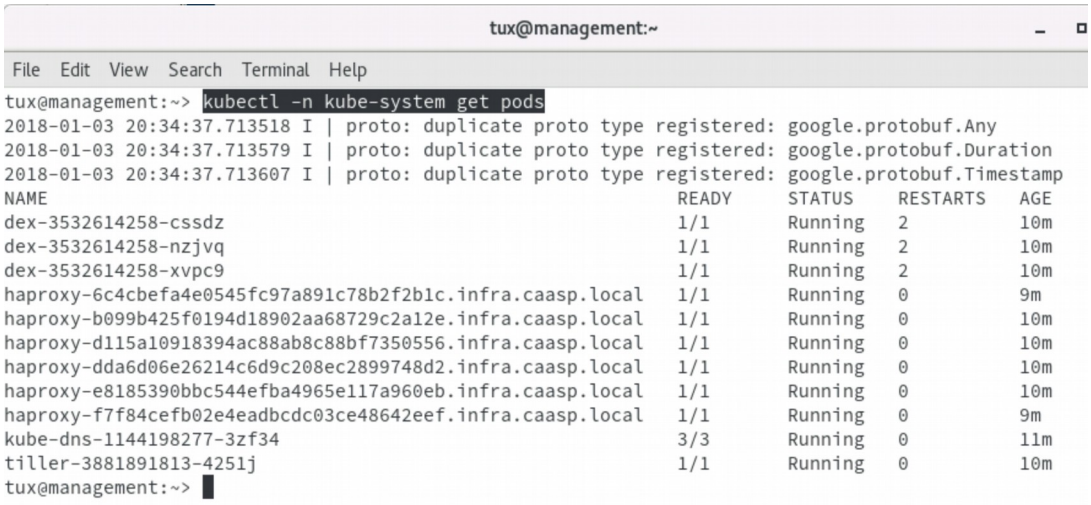
   You should see the list of deployments, their nodes names, status, age and version displayed.



5. Enter the following command to display a list of Kubernetes pods:

   kubectl -n kube-system get pods

   You should see the list of pods, their names, status, and age displayed.



Summary:

In this exercise, you downloaded the kubectl config file from the cluster and then ran some kubectl commands to view information about the cluster.

(End of Exercise)

2- 2 Deploy the Kubernetes Dashboard

Description:

In this exercise, you deploy the Kubernetes Dashboard.

We will also apply an RBAC configuration that turns off token based authentication and stops the dashboard from timing out after 10 mins.

Task 1: Deploy the Kubernetes Dashboard

1. On the **Workstation**, open a terminal
2. Enter the following command to deploy the Kubernetes Dashboard:

   ```
   kubectl apply -f ~/STW-CaaSPv4/manifests/dashboard
   ```

   You should see that a **secret**, **serviceaccount** , **role**, **rolebinding**, **deployment** and **service** were created:
3. Enter the following command to view the pods deployed in the **kube-system** namespace:

   ```
   kubectl -n kube-system get pods
   ```

   You should see the kubernetes-dashboard listed among the pods.
4. Enter the following command to view the deployments in the **kube-system** namespace:

   ```
   kubectl -n kube-system get deployments
   ```

   You should see the kubernetes-dashboard listed among the deployments.
5. Enter the following command to view the services in the **kube-system** namespace:

   ```
   kubectl -n kube-system get services
   ```

   You should see the kubernetes-dashboard listed among the services.

Task 2: Access the Kubernetes Dashboard

1. Enter the following command to start a kubectl proxy in a screen session:

   screen kubectl proxy

   You should see the proxy session started. Notice that port 8001 on the local host is the port used.
2. Enter the following keystrokes to detach from the screen session:

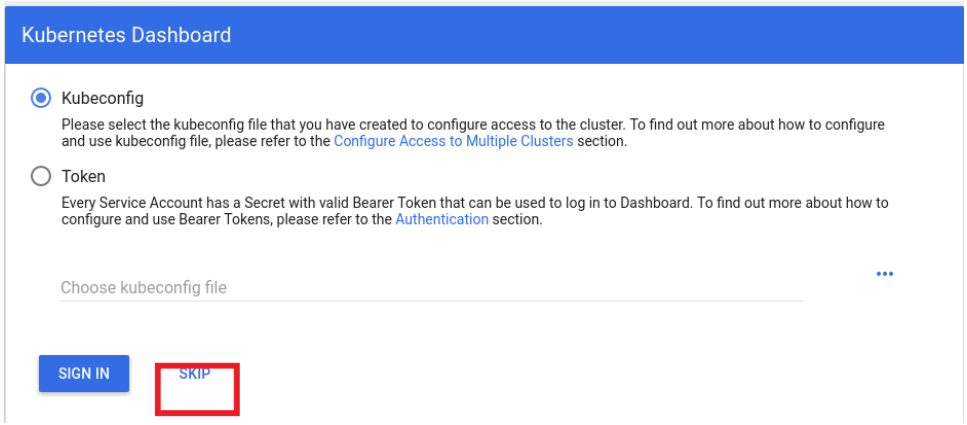
   ctrl+a ctrl+d

   You should be detached from the screen session and back at a command prompt.
3. Open a web browser and point to:

   http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/#!/login

   You should see the Kubernetes Dashboard authentication method screen.

4. Click: **Skip** to login without a token



5. From the list on the left, in the **Namespace** section, from the drop-down list (which is probably displaying the **default** namespace), select **kube-system**
   You should see the kubernetes dashboard deployment in the **kube-system** namespace.
6. From that same drop-down menu, select **default**
   You should see that there is nothing deployed in the **default** namespace.

7.

Summary:

In this exercise, you deployed the Kubernetes Dashboard to the cluster. You then started a kubectl proxy in a screen session and accessed the Kubernetes Dashboard.

(End of Exercise)

2- 3 Install and Configure Helm

Description:

In this exercise, you install install Helm and config it to pull charts from SUSE.

Task 1: Install Helm on Workstation

1. On the **Workstation**, open a terminal
2. Enter the following command to install skuba and the kubernetes client

   sudo zypper in helm

   It will tell you the amount of additional drive space needed and it will prompt you to press **y** to continue

Task 2: Create Service account and Cluster Role for Helm

1. On the **Workstation**, open a terminal
2. Enter the following command in the terminal to create a service account for Helm to use

   kubectl create serviceaccount --namespace kube-system tiller

3. Enter the following command (all one line) in the terminal to create a cluster role for Helm to use

   **kubectl create clusterrolebinding tiller --clusterrole=cluster-admin **
   --serviceaccount=kube-system:tiller

Task 3: Initialize Helm

1. On the **Workstation**, open a terminal
2. Enter the following command in the terminal to create a service account for Helm to use

   **helm init --tiller-image registry.suse.com/caasp/v4/helm-tiller:2.8.2 **
   --service-account tiller

3. Look to see if tiller is currently running (give it a minute if it is not in the **Ready** state)

   kubectl get pods –all-namespaces

```
tux@localhost:~> kubectl get pods --all-namespaces
NAMESPACE     NAME                                    READY   STATUS    RESTARTS   AGE
kube-system   cilium-82kz9                            1/1     Running   0          47m
kube-system   cilium-bzsp7                            1/1     Running   0          47m
kube-system   cilium-operator-7d6ddddbf5-qnbp5        1/1     Running   0          50m
kube-system   cilium-rwwjw                            1/1     Running   0          47m
kube-system   coredns-69c4947958-75vdp                1/1     Running   1          50m
kube-system   coredns-69c4947958-vmf6d                1/1     Running   3          50m
kube-system   etcd-master01                           1/1     Running   0          49m
kube-system   kube-apiserver-master01                 1/1     Running   0          49m
kube-system   kube-controller-manager-master01        1/1     Running   0          49m
kube-system   kube-proxy-2nhh6                        1/1     Running   0          50m
kube-system   kube-proxy-ftbck                        1/1     Running   0          47m
kube-system   kube-proxy-pj698                        1/1     Running   0          48m
kube-system   kube-scheduler-master01                 1/1     Running   0          49m
kube-system   kubernetes-dashboard-5f9c6b756f-hkwxh   1/1     Running   0          45m
kube-system   kured-4qzsf                             1/1     Running   0          45m
kube-system   kured-7tcfj                             1/1     Running   0          49m
kube-system   kured-9z4g7                             1/1     Running   0          47m
kube-system   oidc-dex-55fc689dc-mccw8                1/1     Running   1          50m
kube-system   oidc-gangway-7b7fbbdbdf-4w6z8           1/1     Running   0          50m
kube-system   tiller-deploy-7c666b7c99-7whpp          1/1     Running   0          42m
tux@localhost:~>
```

Task 4: Add SUSE Charts to Helm

1. On the **Workstation**, open a terminal
2. Enter the following command in the terminal to create a service account for Helm to use

   helm repo add suse-charts https://kubernetes-charts.suse.com

3. Look at available charts from SUSE

   helm search suse

```
tux@localhost:~> helm search suse
NAME                                    CHART VERSION   APP VERSION     DESCRIPTION
suse-charts/cf                          2.17.1          1.4.1           A Helm chart for SUSE Cloud Foundry
suse-charts/cf-usb-sidecar-mysql        1.0.1                           A Helm chart for SUSE Universal Service Broker ...
suse-charts/cf-usb-sidecar-postgres     1.0.1                           A Helm chart for SUSE Universal Service Broker ...
suse-charts/console                     2.4.0           2.4.0           A Helm chart for deploying Stratos UI Console
suse-charts/log-agent-rsyslog           1.0.1           8.39.0          Log Agent for forwarding logs of K8s control pl...
suse-charts/metrics                     1.0.0           1.0.0           A Helm chart for Stratos Metrics
suse-charts/minibroker                  0.2.0                           A minibroker for your minikube
suse-charts/nginx-ingress               0.28.4          0.15.0          An nginx Ingress controller that uses ConfigMap...
suse-charts/uaa                         2.17.1          1.4.1           A Helm chart for SUSE UAA
tux@localhost:~>
```

Task 5: Install Centralize Logging

1. On the **Workstation**, open a terminal
2. Enter the following command in the terminal to create a service account for Helm to use

```
helm install suse-charts/log-agent-rsyslog --name 1.0.1 /
--set server.host=rsyslog-server.default.svc.cluster.local /
--set server.port=514
```

Summary:

In this exercise, you installed Helm on the Management workstation. You then setup and configured it to pull charts directly from SUSE. You then installed the CaaS Platform's Centralized Logging Service.

(End of Exercise)

3 Deploying a Workload

Description:

In this section you will deploy a simple workload in Kubernetes.

3- 1 Deploy a Simple Pod on Kubernetes

Description:

In this exercise, you deploy the Nginx web server as a simple pod on the Kubernetes cluster.

Task 1: View a Manifest for the Deployment

1. On the **management workstation**, in the text editor of your choice, open the file:

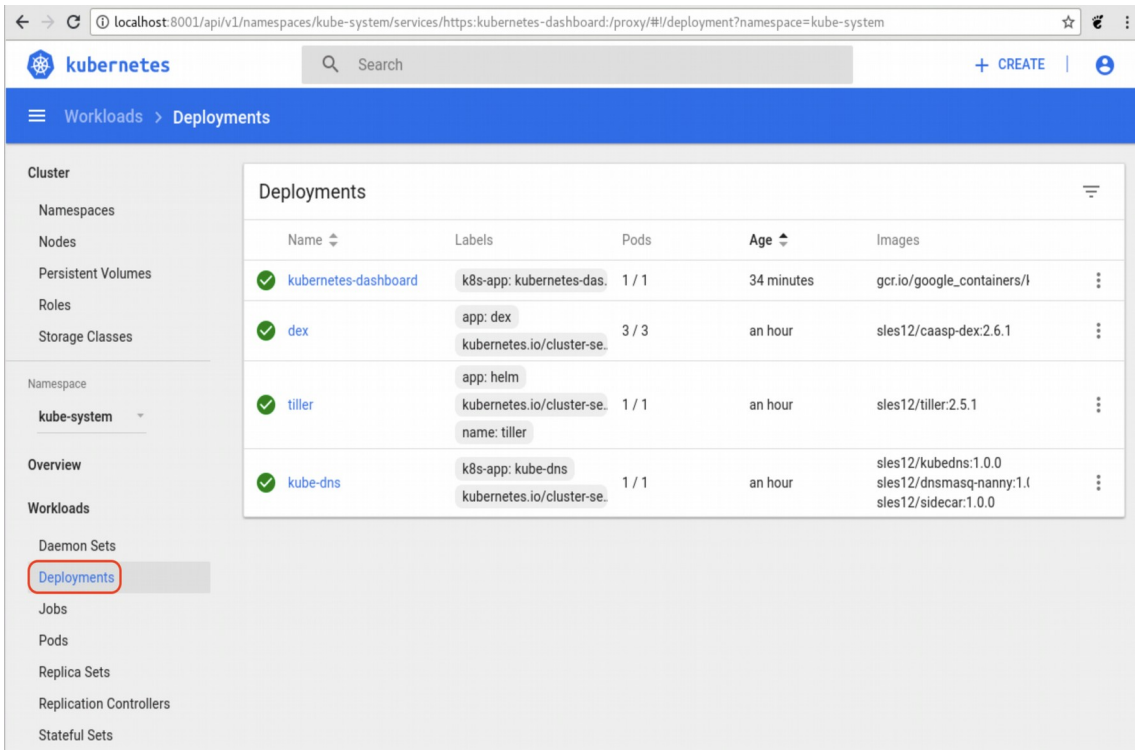
   ~/STW-CaaSPv4/labs/nginx-deployment.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 4
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: rmt.example.com:5000/sles12sp3_nginx
        ports:
        - containerPort: 80
```

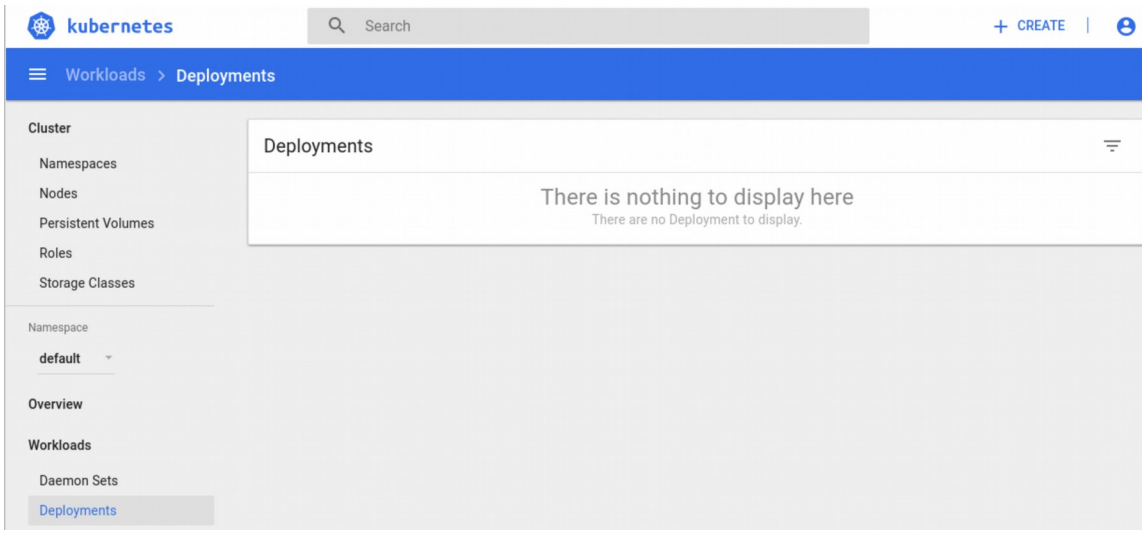
Task 2: Deploy a simple application (GUI option)

To deploy the application, go to the Kubernetes Dashboard

1. Click on **Deployments** on the left hand panel

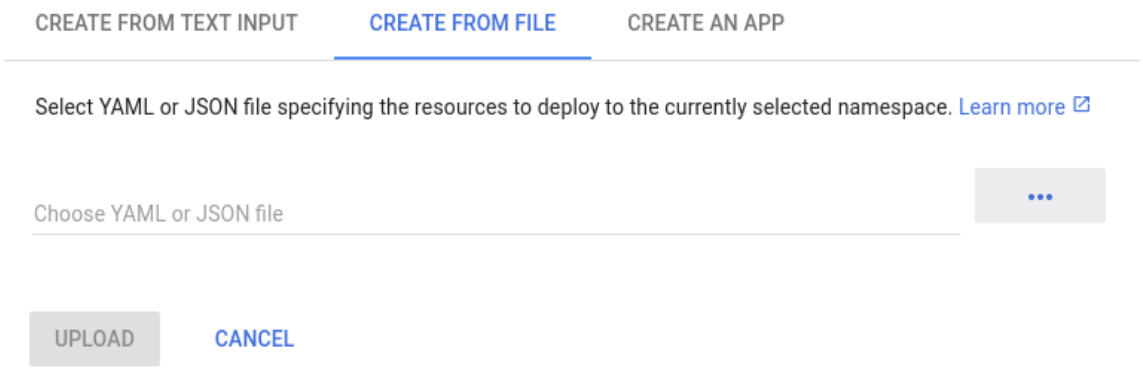


   All of the currently deployed Deployments are in the kube-system namespace
2. Select **Namespace** on the left hand panel
3. Change the **Namespace** to **Default** and select **Deployments** again

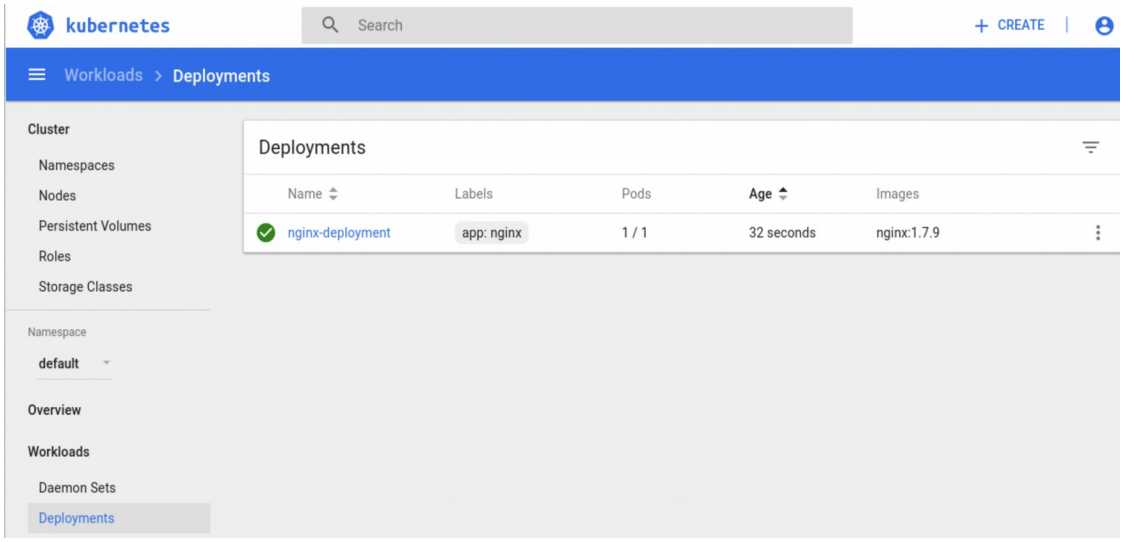


   note there are no current deployments in the **default** namespace

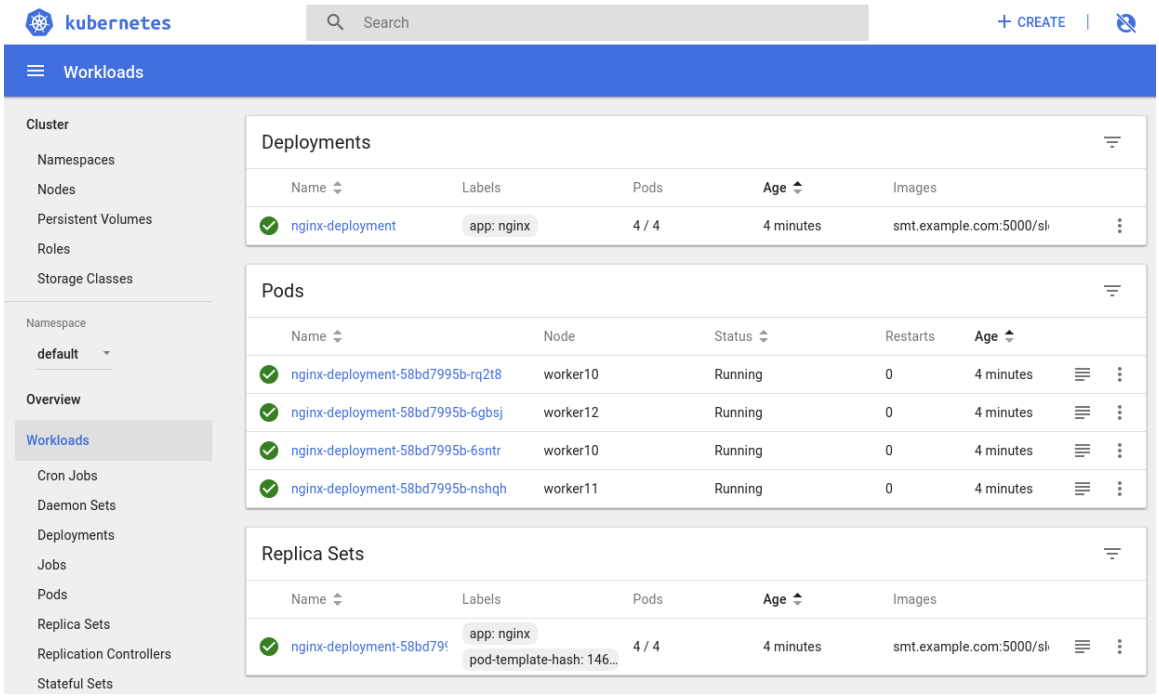
4. Click on **+ CREATE**
5. Select **Create from File** and the press '**…' to** select file



6. Browse to **~/STW-CaaSPv4/labs/nginx-deployment.yaml**
7. Press Upload
8. Select **Deployments** view and we should now see the nginx-deployment



9. Explore the Kubernetes Dashboard
   Make sure you look at the **Overview, Workloads, and Pods** views



Task 3: Deploy a simple application (Command Line option)

10. To deploy the pod, open a terminal and enter the following command:

kubectl apply -f ~/STW-CaaSPv4/labs/nginx-deployment.yaml

```
tux@management:~> kubectl apply -f ~/course_files/CAAS101/manifests/labs/nginx-deployment.yaml
2018-03-13 09:27:42.158048 I | proto: duplicate proto type registered: google.protobuf.Any
2018-03-13 09:27:42.158121 I | proto: duplicate proto type registered: google.protobuf.Duration
2018-03-13 09:27:42.158136 I | proto: duplicate proto type registered: google.protobuf.Timestamp
deployment "nginx-deployment" created
tux@management:~>
```

You should see the deployment “nginx-deployment” was created.

11. Enter the following command to view the deployments:

    kubectl get deployments

    You should see that a single instance of the **nginx-deployment** pod is running.
12. Enter the following command to view the deployed pods:

    kubectl get pods

    You should see a single instance of the **nginx-deployment** pod running.

Summary:

In this exercise, you launched a single instance of the Nginx web server as a pod in a deployment on the cluster.

(End of Exercise)

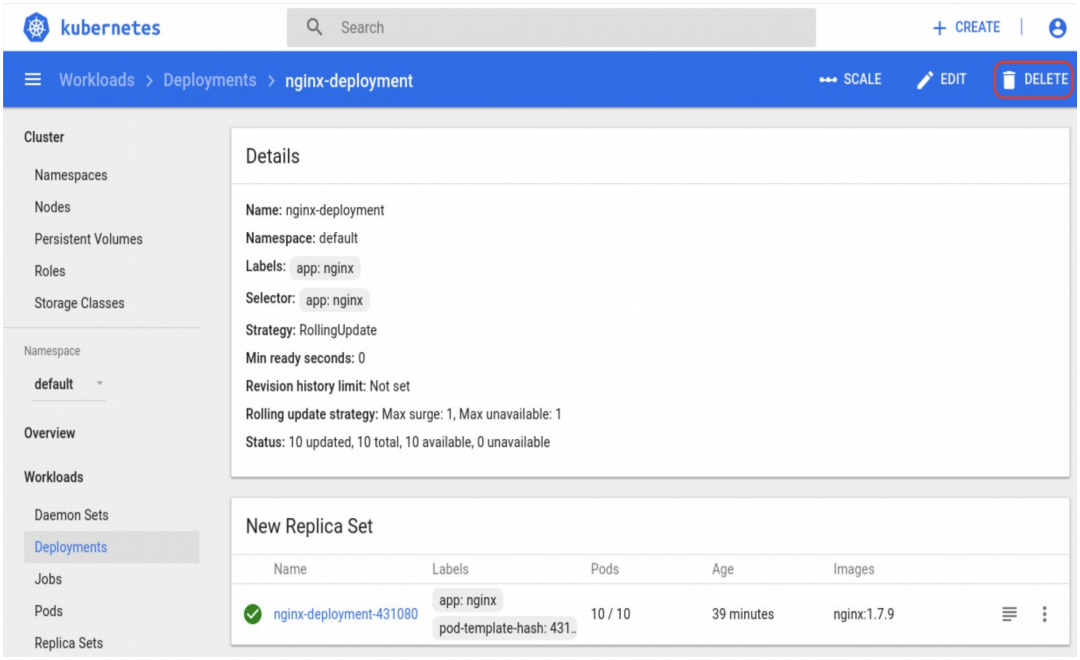
3- 2 Delete a Deployment on Kubernetes

Description:

In this exercise, you delete the Nginx web server deployment that was previously deployed on the Kubernetes cluster.

Task 1: Delete the Deployment (GUI Option)

1. Select **nginx-deployment** under Deployments
2. Select **Delete**



Task 2: Delete the Deployment (Command Line option)

3. To view the deployments, on the **Workstation**, enter the following command:

   kubectl get deployments

   You should see that one instance of the **nginx-deployment** is running.

4. Enter the following command to delete the deployment:

   kubectl delete deployment nginx-deployment

   You should see the **nginx-deployment** was deleted.

5. View the deployments again:

   kubectl get deployments

   You should see that the **nginx-deployment** is no longer running.

6. View the pods:

   kubectl get pods

   You should see that all of the pods that were part of the **nginx-deployment** are no longer running.

Summary:

In this exercise, you deleted the Nginx web server deployment that was previously deployed on the cluster.

(End of Exercise)

3- 3 Scale Out a Deployment

Description:

In this exercise, you scale out a running Deployment.

Task 1: View the Manifest for the Deployment

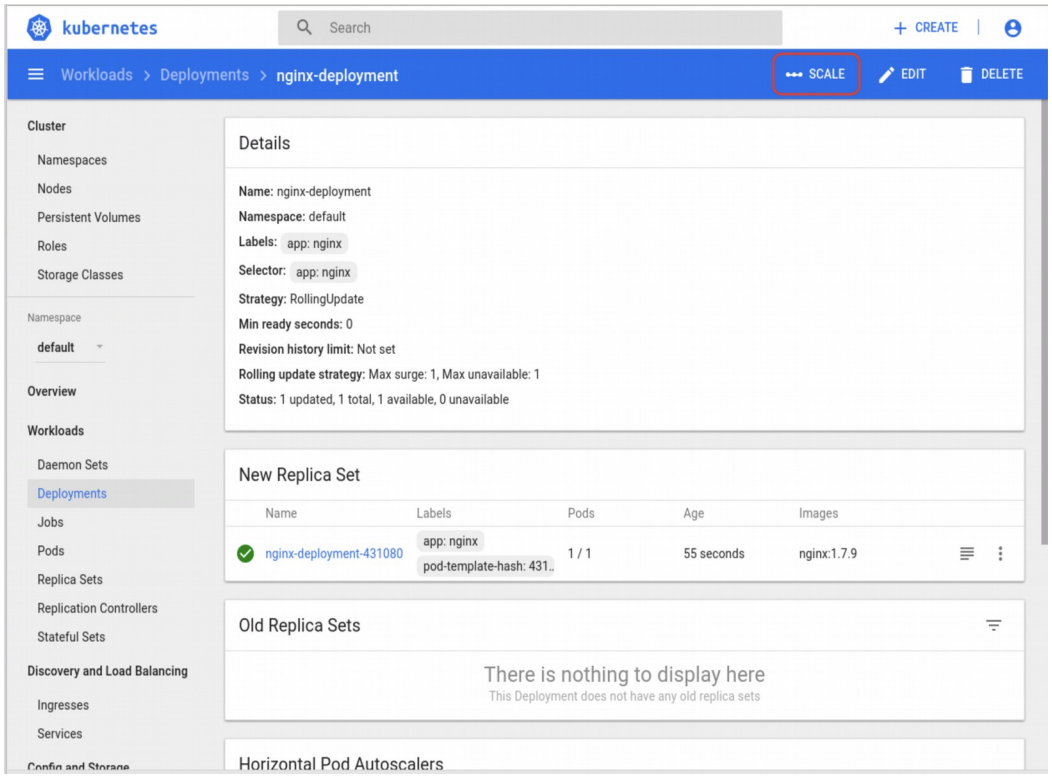
1. In the text editor of your choice, view the **~/STW-CaaSPv4/labs/nginx-deployment.yaml** file

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 4
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: rmt.example.com:5000/sles12sp3_nginx
        ports:
        - containerPort: 80
```

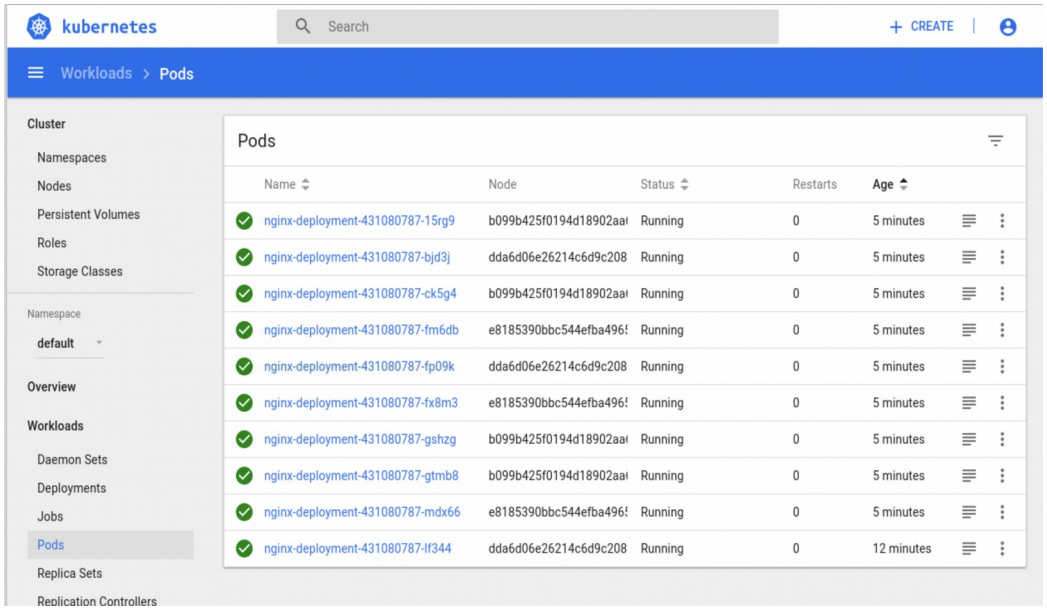
Task 2: Scale the Deployment (GUI option)

1. From **Kubernetes Dashboard** select **Create** and open **~/STW-CaaSPv4/labs/nginx-scale.yaml**
2. Press the **Deploy** button
3. Select the **nginx-deployment** under **Deployments**

4. click the **Scale** Button and change the number of **Desired pods** to 10



5. View the deployed pods by selecting **Pods** in the left hand panel



6. Scale the Pods back down to 4

Task 3: Scale the Deployment (Command Line Option)

1. In the text editor of your choice, open the **~/STW-CaaSPv4/labs/nginx-scale.yaml** file
   Note the the only difference is the number of replicas

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2
  revisionHistoryLimit: 5
  minReadySeconds: 20
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 25%
      maxSurge: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: rmt.example.com:5000/sles12sp3_nginx
        ports:
        - containerPort: 80
```

2. To scale the deployment, open a terminal and enter the following command:

   kubectl apply -f ~/STW-CaaSPv4/labs/nginx-scale.yaml

3. Enter the following command to view the deployments:

   kubectl get deployments

   You should see that one instance of the **nginx-deployment** Deployment is running.

4. Enter the following command to view the deployed pods:

   kubectl get pods

You should see 10 instances of the **nginx-deployment** pod running.

Summary:

In this exercise, you created a new manifest for an existing deployment that specified a smaller number of replicas. You then applied the updated manifest to scale in the deployment. Finally you applied the original manifest to scale the deployment back out.

(End of Exercise)

4 Working With Kubernetes

Description:

In this section you are introduced to how to work with Kubernetes. You are first introduced to Kubernetes configuration and management utilities and manifests. You then deploy and manage pods on a Kubernetes cluster.

4- 1 Update a Deployment

Description:

In this exercise, you update a running pod.

Task 1: Create a New Manifest for the Deployment

1. In the text editor of your choice, open the **~/STW-CaaSPv4/labs/nginx-update.yaml** file
2. Notice the section in **red**

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 4
  template:
    metadata:
```

```
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: rmt.example.com:5000/nginx:1.9.0
    ports:
    - containerPort: 80
```

Task 2: Update the Deployment

1. If you don't already have **nginx** deployed Deploy either through **KubeDashboard** using the file **~/STW-CaaSPv4/labs/nginx-pre-update.yaml** or the following command line:

   kubectl apply -f ~/STW-CaaSPv4/labs/nginx-pre-update.yaml
2. To display information on the current nginx deployment enter the following command:

   kubectl describe deployment -l app=nginx

   You should see the description of the nginx deployment displayed.
   Notice the image version is: nginx:1.7.9
3. Open another terminal and enter the following command to watch the running pods:

   watch kubectl get pods

   You should see a list of the running pods displayed with the list updating every 2 seconds.
4. To update the deployment, open a terminal and enter the following command:

   kubectl apply -f ~/STW-CaaSPv4/labs/nginx-update.yaml

   You should see the deployment "nginx-deployment" was configured.
5. Enter the following command to view the deployments:

   kubectl get deployments

   You should see that 4 instances of the **nginx-deployment** pod are DESIRED and, depending on when you ran the command, the values in the CURRENT, UP-TO-DATE and AVAILABLE columns my be more, fewer or the same number as the update happens .
6. In the terminal where you are watching the pods enter **ctrl+c** to stop the watch command
7. Enter the following command to display information about the running deployment of nginx:

kubectl describe pods -l app=nginx

Notice the image version is now: nginx:1.9.0

Summary:

In this exercise, you created a new manifest to update the running nginx deployment. You then updated the deployment and verified that it was updated.

(End of Exercise)

4- 2 Update a Deployment Via Rolling Updates

Description:

In this exercise, you update a running pod using rolling updates.

Task 1: Create a New Manifest for the Deployment

1. On the management workstation, In the text editor of your choice, open the **~/STW-CaaSPv4/labs/nginx-rolling_update.yaml** file
2. Notice the changes are in **red**

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 4
  revisionHistoryLimit: 5
  minReadySeconds: 20
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 25%
      maxSurge: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: rmt.example.com:/5000/nginx:1.12.0
        ports:
        - containerPort: 80
```

3. Save the file and close the text editor

Task 2: Update the Deployment

1. To display information on the current nginx deployment enter the following command:

   kubectl describe deployment -l app=nginx

   You should see the description of the nginx deployment displayed.
   Notice the image version is: **nginx:1.9.0**
2. Open another terminal and enter the following command to watch the running pods:

   watch kubectl get pods

   You should see a list of the running pods displayed with the list updating every 2 seconds.
3. To update the deployment, open a terminal and enter the following command:

   kubectl apply -f ~/STW-CaaSPv4/labs/nginx-rolling_update.yaml

   You should see the deployment “nginx-deployment” was configured.
   In the terminal where you are watching the pods you should see the number of running pods drop to 3 (because of maxUnavailable: 25%) and 3 new pods start deploying. Shortly after the 3 new pods are running you should see the number of running pods scale up to 6 (because of maxSurge: 2) and then back to 4.
4. Enter the following command to view the deployments:

   kubectl get deployments

   You should see that 4 instances of the **nginx-deployment** pod are running.
5. In the terminal where you are watching the pods enter **ctrl+c** to stop the watch command
6. Enter the following command to display information about the running deployment of nginx:

   kubectl describe deployment -l app=nginx

   Notice the image version is now: **nginx:1.12.0**

Summary:

In this exercise, you created a new manifest to update the running nginx deployment using the rolling update type. You then updated the deployment and verified that it was updated.

(End of Exercise)

4- 3 Expose a Service Running in a Pod

Description:

In this exercise, you expose a service running in a pod.

Task 1: Create a Manifest for the Service

1. On the **Workstation**, in the text editor of your choice, open the file:

   ~/STW-CaaSPv4/labs/nginx-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  type: NodePort
  ports:
    - port: 80
      nodePort: 30000
  selector:
      app: nginx
```

Task 2: Define the Service

1. To define the service in the cluster, open a terminal and enter the following command:

   kubectl apply -f ~/STW-CaaSPv4/labs/nginx-service.yaml

   You should see the service “nginx-service” was created.
2. Enter the following command to view the services:

   kubectl get services

   You should see that the **nginx-service** service is defined. Notice the 80:30000 under ports showing external port 30000 will be redirected into internal port 80.

Task 3: Access the Exposed Service

1. To access the exposed service, open a web browser and point to:

   http://worker.example.com:30000

   You should see the Welcome to nginx web page.

Note:

If desired, you can change the URL to point to a specific worker node (i.e. **worker10.example.com**) and see that the web page is still accessible. This demonstrates that the kube-proxy is working on all of the cluster nodes.

Summary:

In this exercise, you defined a service in the cluster exposing port 80 that allowed access to the nginx pod running on the cluster. You then accessed the nginx pod in a web browser.

(End of Exercise)

4- 4 Setup Readiness and Liveness Probes

Description:

In this exercise, you learn how to setup and configure Readiness and Liveness Probes

---

Task 1: View manifest for the Deployment

1. In the text editor of your choice, open the **~/STW-CaaSPv4/labs/nginx-deployment-health.yaml** file

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 4
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: rmt.example.com:5000/sles12sp3_nginx
        ports:
        - containerPort: 80
        readinessProbe:
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 5
          timeoutSeconds: 1
          periodSeconds: 15
        livenessProbe:
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 15
          timeoutSeconds: 1
          periodSeconds: 15
```

2. Notice the ReadinessProbe and LivenessProbe settings

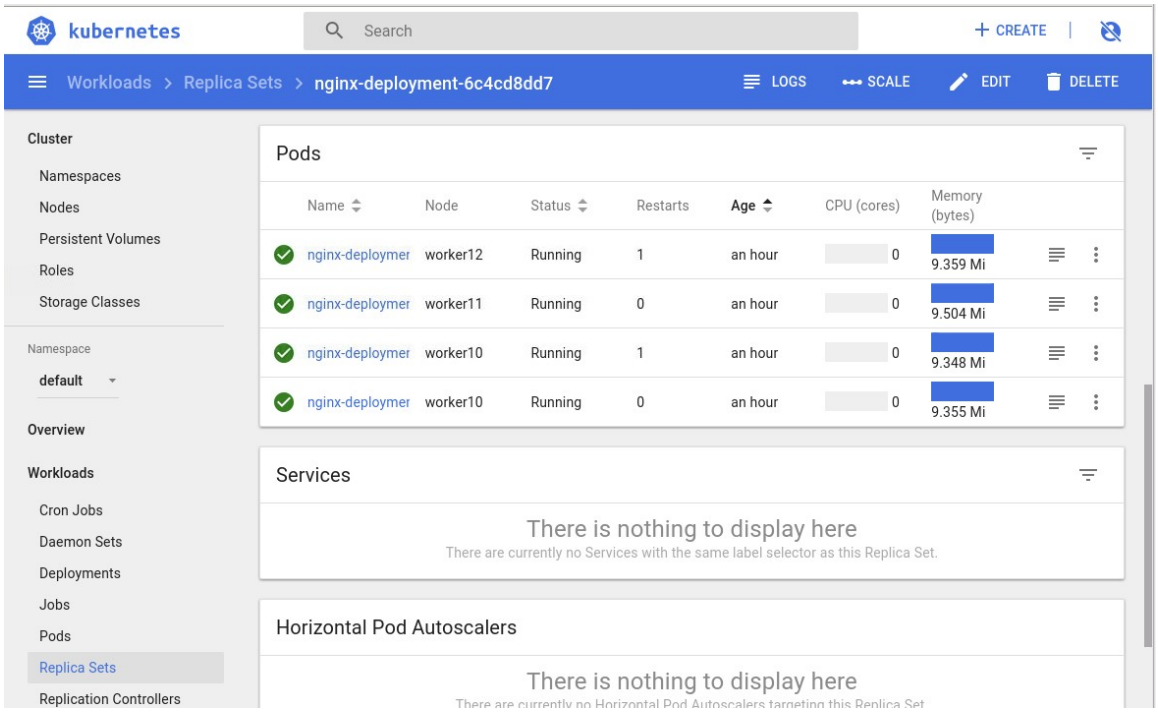
Task 2:Deploy Manifest

1. Deploy the manifest

   kubectl apply -f ~/STW-CaaSPv4/labs/nginx-deployment-health.yaml

Task 3: View the Replica sets

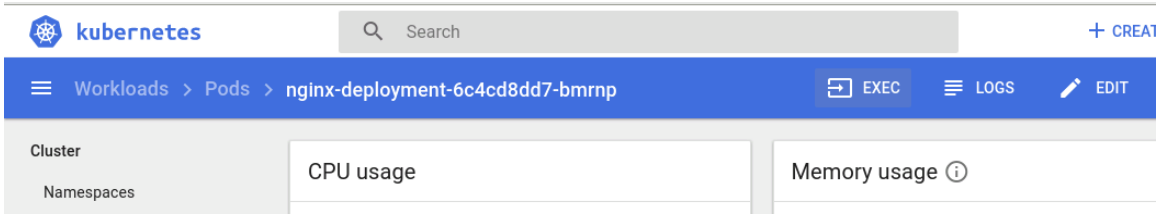
2. On the lefthand side of the Kubernetes Dashboard, under Workload, select Replica Sets



3. Notice the number of Restarts on the container

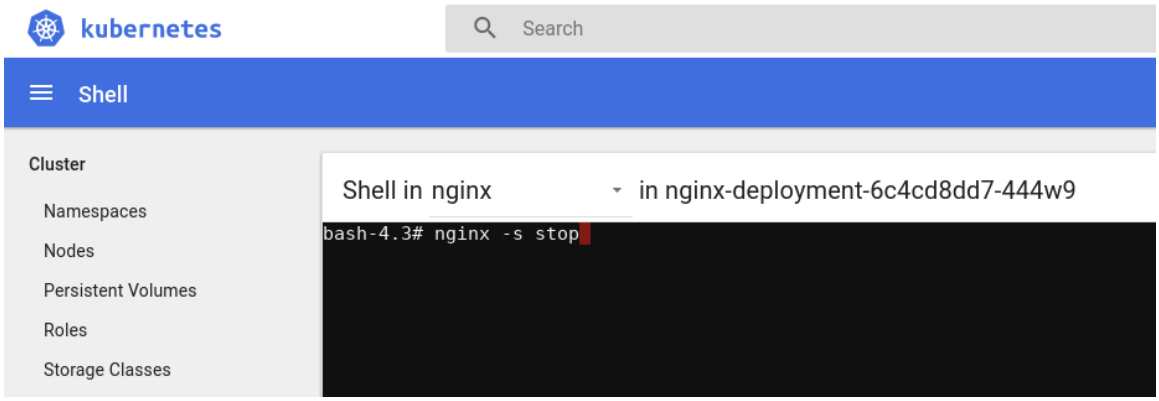
Task 4: Kill an nginx server

1. Select one of the pods labeled nginx-deployment-#### by **clicking** on it
2. **Click** the EXEC button to be taken to a shell prompt inside the running pod

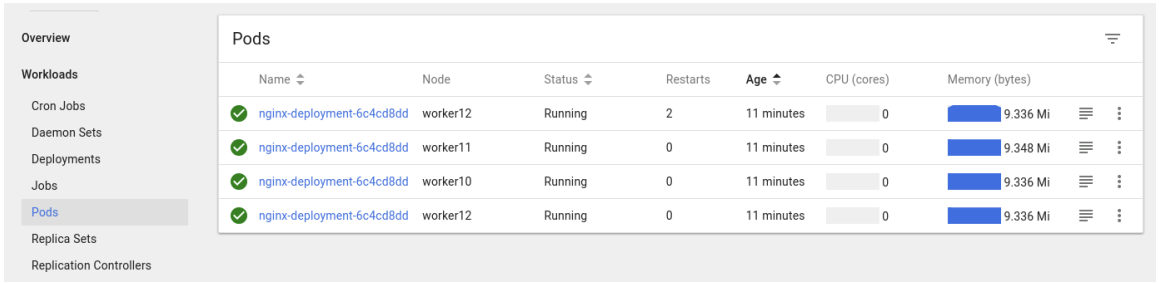


1. From the Shell enter the following command to kill the nginx process

   nginx -s stop



2. Click on Pods to view the current pods running



3. Notice the number of restarts has increased. This is because Kubernetes could no longer reach the server on port 80 os it restarted that Nginx Container inside the Pod

---

Summary:

In this exercise, you created a new manifest to update the running nginx deployment. You then updated the deployment and verified that it was updated.

(End of Exercise)

4- 5 Define Limits for Containers and Pods in Kubernetes

Description:

In this exercise, you define limits for containers and pods in the Kubernetes cluster.

Task 1: Create a New Namespace in the Cluster

1. On the **Workstation**, at the command line, enter the following command to create a new namespace in the Kubernetes cluster:

   kubectl create namespace limit-example

   You should see that a new namespace was created.
2. Enter the following command to display the namespaces:

   kubectl get namespaces

   You should see the new namespace listed.

Task 2: Create a Manifest for the Limits

1. In the text editor open the file:

   ~/STW-CaaSPv4/labs/limits.yaml

```
apiVersion: v1
kind: LimitRange
metadata:
  name: mylimits
spec:
  limits:
  - max:
      cpu: "2"
      memory: 1Gi
    min:
      cpu: 200m
      memory: 6Mi
    type: Pod
  - default:
      cpu: 300m
      memory: 200Mi
    defaultRequest:
      cpu: 200m
      memory: 100Mi
    max:
      cpu: "2"
      memory: 1Gi
    min:
      cpu: 100m
      memory: 3Mi
    type: Container
```

Task 3: Apply the Limits to the Namespace

1. To deploy the pod, open a terminal and enter the following command:

   kubectl apply –f ~/STW-CaaSPv4/labs/limits.yaml --namespace=limit-example

   You should see the limitrange “mylimits” was created.
2. Enter the following command to display the limitranges:

   kubectl describe limitranges --namespace=limit-example

   You should see the **mylimits** limitrange listed.

```
Name:           mylimits
Namespace:      limit-example
Type            Resource        Min     Max     Default Request Default Limit   Max Limit/Request Ratio
----            --------        ---     ---     --------------- -------------   -----------------------
Pod             cpu             200m    2       -               -               -
Pod             memory          6Mi     1Gi     -               -               -
Container       cpu             100m    2       200m            300m            -
Container       memory          3Mi     1Gi     100Mi           200Mi           -
tux@management:~/Downloads>
```

Summary:

In this exercise, you created a new namespace in the Kubernetes cluster. You then defined limits for containers and pods and applied them to the new namespace.

(End of Exercise)

4- 6 Introduction to Helm

Description:

In this exercise, you are introduced to the helm utility.

Task 1: Use Some Basic Helm Commands

1. In a terminal, enter the following command to source the Helm auto-completion functions into your shell environment:

   source <(helm completion bash)

   You shell environment should now have the helm auto-completion functions available.
2. Enter the following command to display a list of available helm charts:

   helm search

   You should see a list of available helm charts listed.
3. Try running the command again but rather than typing in the entire command, only type:

   helm sea[Tab]

   (Where [Tab] is the tab key)

   Notice that the “**search**” option is auto-completed.
4. Enter the following command to search for a specific helm chart:

   helm search dokuwiki

   You should see the dokuwiki chart listed.
5. Enter the following command to display the list of currently configured repos:

helm repo list

You should see a repo for **https://kubernetes-charts.storage.googleapis.com**, **kubernetes-charts.suse.com** and an addition to the local repo.

Summary:

In this exercise, you initialized helm. You then ran some basic helm commands such as configuring auto-completion, searching for helm charts and displaying repositories.

(End of Exercise)

4- 7 Deploy an Application with Helm

Description:

In this exercise, you deploy an application from a helm chart using the helm command.

Task 1: Create Helm Chart Config File

1. On the **Workstation**, in a terminal, enter the following command to search for a dokuwiki helm chart:

   helm search dokuwiki

   You should see a helm chart named stable/dokuwiki listed with its available chart version.
2. Enter the following command to view the default configuration for the dokuwiki chart:

   helm inspect stable/dokuwiki | less

   You should see the configuration displayed in the less pager. Page through the configuration to see what variables are being set.
3. In the text editor of your choice, create/open the **~/STW-CaaSPv4/labs/dokuwiki-config.yaml** file
4. Enter the following in the file:

```
dokuwikiPassword: password123
serviceType: NodePort
persistence:
  enabled: false
```

5. Save the file and close the text editor

Task 2: Deploy the Helm Chart

1. In a terminal, enter the following command to view the current helm releases:

   helm list

   You should not see the dokuwiki application listed.
2. Enter the following command to deploy the chart:

   helm install -f ~/STW-CaaSPv4/labs/dokuwiki-config.yaml --name mywiki stable/dokuwiki

   You should see the chart was deployed.
   In the output of the chart deployment, in the **==> v1/Service** section, notice the IP and port(s) the application is listening on. The NodePort(s) that the application is listening on are the number after the colon (**:**) in the **PORT(S)** column.

   Example: **80:32313/TCP,443:31034/TCP**

   In this example the NodePorts are **32313** for http and **31034** for https.

   Record the http NodePort here:
   DOKUWIKI_PORT=_____
3. Enter the following command to display the current helm releases:

   helm list

   You should now see the dokuwiki chart listed with a name of **mywiki**. Notice the REVISION column shows the number **1** as this is the initial deployment of this release.
4. Enter the following command to view the release history for the application:

   helm history mywiki

   You should see that only a single release of **mywiki** exists.
5. Enter the following command to view the status of the **mywiki** release:

   helm status mywiki

   You should see output similar to what was displayed when the chart was first deployed.
6. You can also enter the following **kubectl** commands:

   kubectl get deployments
   kubectl get pods
   kubectl get services

   Notice that you see that same info about the deployed deployments/pods/services as if you were to have deployed them from manifests.

Task 3: Access the Application Deployed by the Helm Chart

1. Open a web browser and point to:

http://worker.example.com:*DOKUWIKI_PORT*

You should see the My Wiki page displayed.

2. On the top right of the page click: **Login**
3. Enter the following credentials:

   Username: **user**
   Password: **password123**

   You should be logged in.

   After logging in you probably see a number of warnings of available hotfixes and/or updates.

Task 4: Update the Application Release

1. On the Workstation, in a terminal, make a copy of the chart config file you created at the beginning of this exercise:

   cp ~/dokuwiki-config.yaml ~/dokuwiki-config-update.yaml

2. In the text editor of your choice, open the **~/dokuwiki-config-update.yaml** file
3. Edit the file to match the following (changes are in **red**):

   images: bitnami/dokuwiki:latest
   dokuwikiPassword: password123
   serviceType: NodePort
   persistence:
   ** enabled: false**

4. Save the file and close the text editor
5. Enter the following command to upgrade the mywiki release:

   helm upgrade -f ~/dokuwiki-config-update.yaml mywiki stable/dokuwiki

   You should see the chart was successfully deployed.
6. Enter the following command to view the current helm releases:

   helm list

   You should see the dokuwiki chart named **mywiki** listed. Notice the REVISION column now shows the number **2** as the release has been updated once.
7. Enter the following command to display the release history for the mywiki release:

   helm history mywiki

   Notice that there are 2 revisions. Also notice the the first revision's STATUS is **SUPERSEDED** and the second revision's STATUS is **DEPLOYED**.
8. In the web browser, refresh the web page (or log back in if you have logged out)

   You should no longer see the warning messages about available hotfixes and/or

updates.

Task 5: Delete a Deployed Helm Chart Release

1. In a terminal, enter the following command to delete the mywiki release:

   ```
   helm delete mywiki
   ```

   You should see that the release was deleted.
2. Enter the following command to list the current helm releases:

   ```
   helm list
   ```

   You should no longer see the dokuwiki chart named **mywiki** displayed.
3. Enter the following command to display the status of the mywiki release:

   ```
   helm status mywiki
   ```

   Notice that the STATUS is **DELETED** but also that it remembered that it had been deployed as it has a date listed in LAST DEPLOYED.

Summary:

In this exercise, you first deployed a helm chart. You then accessed the application that was deployed, Next you updated the release, verifying is was updated. Finally you deleted the release.

(End of Exercise)

5 Storage

Description:

How to handle persistent storage in pods.

5- 1 Configure NFS Persistent Storage

Description:

In this exercise, you configure a persistent volume on an NFS server. You then create a pod that updates a file on the persistent volume and a pod that exports the file via http

Task 1: Create the Persistent Volume on the NFS Server

1. On the **RMT server**, as the **root** user with the password of **Linux** enter the following commands in a terminal session to create the persistent volume and **index.html** file:

   mkdir -p /export/vol-01
   touch /export/vol-01/index.html

   The directory and file should now exist.

Task 2: Create the Manifest for the Persistent Volume

2. On the **CAAS101-management VM,** in the text editor of your choice, open the file:

   ~/STW-CaaSPv4/labs/pv/nfs-pv-vol-01.yaml

   Enter the following in the file:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-vol-01
spec:
  capacity:
    storage: 1Mi
  accessModes:
    - ReadWriteMany
  nfs:
    # FIXME: use the right IP
    server: 192.168.110.2
    path: "/export/vol-01"
```

Task 3: Create the Manifest for the Persistent Volume Claim

1. In the text editor of your choice, open the file:

~/STW-CaaSPv4/labs/pv/nfs-pvc-vol-01.yaml

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: nfs-vol-01
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Mi
```

Task 4: Create the Manifest for the Busybox Instance

1. In the text editor of your choice, open the file:

   ~/STW-CaaSPv4/labs/pv/nfs-busybox-deployment.yaml

```yaml
# This mounts the nfs volume claim into /mnt and continuously
# overwrites /mnt/index.html with the time and hostname of the pod.

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nfs-busybox
spec:
  replicas: 1
  template:
    metadata:
      labels:
        name: nfs-busybox
    spec:
      containers:
      - image: smt.example.com:5000/busybox
        command:
          - sh
          - -c
          - 'while true; do date > /mnt/index.html; hostname >> /mnt/index.html; sleep $(($RANDOM % 5 + 5)); done'
        imagePullPolicy: IfNotPresent
        name: busybox
        volumeMounts:
          # name must match the volume name below
          - name: nfs-vol-01
            mountPath: "/mnt"
      volumes:
      - name: nfs-vol-01
        persistentVolumeClaim:
          claimName: nfs-vol-01
```

Task 5: Create the Manifests for the Web Frontend

1. In the text editor of your choice, create/open the file:

   ~/STW-CaaSPv4/labs/pv/nfs-web-deployment.yaml

```
# This pod mounts the nfs volume claim into /usr/share/nginx/html and
# serves a simple web page.

apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nfs-web
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: web-frontend
    spec:
      containers:
      - name: web
        image: rmt.example.com:5000/nginx:1.9.0
        ports:
          - name: web
            containerPort: 80
        volumeMounts:
            - name: nfs-vol-01
              mountPath: "/usr/share/nginx/html"
      volumes:
      - name: nfs-vol-01
        persistentVolumeClaim:
          claimName: nfs-vol-01
```

2. Next, open the file:

   ~/STW-CaaSPv4/labs/pv/nfs-web-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nfs-web
spec:
  type: NodePort
  ports:
    - port: 80
      nodePort: 30080
  selector:
    app: web-frontend
```

Task 6: Deploy the Objects

1. To deploy the volumes/pods/service, open a terminal and enter the following command:

   kubectl apply -f ~/STW-CaaSPv4/labs/pv

You should see the following were created (not necessarily in this order):

deployment “nfs-busybox”
persistentvolume “nfs-vol-01”
persistenvolumeclaim “nfs-vol-01”
deployment “nfs-web”
deployment “nfs-web”

2. Enter the following command to view the deployments:

   kubectl get deployments

   You should see the **nfs-busybox** and **nfs-web** deployments listed.

3. Enter the following command to view the pods:

   watch kubectl get pods

   You should see the pods for the **nfs-busybox** and **nfs-web** deployments listed.
4. Enter the following command to view the persistent volumes:

   kubectl get pv

   You should see the persistent volume **nfs-vol-01** listed.
5. Enter the following command to view the persistent volumes:

   kubectl get pvc

   You should see the persistent volume claim **nfs-vol-01** listed.

Task 7: Test the Persistent Data

1. On the **SMT server**, enter the following command to view the content of the **index.html** file on the NFS server:

   cat /export/vol-01/index.html

   You should see a line with the date and time and a line with the pod name of the **nfs-busybox** pod. If you rerun the command you should see that the time stamp is updating.
2. On the **management workstation**, open a web browser and point to:

   http://worker10.example.com:30080

   You should see the content of the **index.html** file. When you refresh the page you should see the time stamp updating. (Also notice that if you change the URL to the different worker nodes you will see the same thing.)

Task 8: Remove the Objects from the Cluster

1. In the first terminal, enter the following commands to delete all of the objects:

   kubectl delete service nfs-web
   kubectl delete deployment nfs-web
   kubectl delete deployment nfs-busybox
   kubectl delete pv nfs-vol-01
   kubectl delete pvc nfs-vol-01

   You should see that the objects were deleted.

Summary:

In this exercise, you created manifests for a persistent volume, a persistent volume claim, a pod to that attached to the volume and writes data to an index.html file in the volume, and a web server that attaches to the volume and displays the index.html file. You then verified that the index.html file was being updated by looking at the file both on the NFS volume and the web server.

(End of Exercise)

5- 2 Configure Persistent Storage with a NFS StorageClass

Description:

In this exercise, you define a StorageClass that will automatically create volumes on an NFS server. You then create a volume claim that will cause the volume to be created on the NFS server and a pod that will write a file in the volume.

Task 1: Create the Directory for the Persistent Volumes on the NFS Server

1. On the **SMT server**, as the **root** user with a password of **Linux**, enter the following commands to create the directory that will contain the volumes:

   mkdir -p /export/volumes
   chmod 777 /export/volumes

   The directory and file should now exist.

Task 2: Define a Service Account, Cluster Role and Cluster Role Binding

1. On the **CAAS101-management VM,** in the text editor of your choice, open the file:
   ~/STW-CaaSPv4/labs/nfs-client-storageclass/auth/serviceaccount.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: nfs-web
spec:
  type: NodePort
  ports:
    - port: 80
      nodePort: 30080
  selector:
    app: web-frontend
```

2. In the text editor of your choice, open the file:
3. **~/STW-CaaSPv4/labs/nfs-client-storageclass/auth/clusterrole.yaml**

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: nfs-client-provisioner-runner
rules:
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["list", "watch", "create", "update", "patch"]
```

4. In the text editor of your choice, open the file:
   ~/STW-CaaSPv4/labs/nfs-client-storageclass/auth/clusterrolebinding.yaml

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: run-nfs-client-provisioner
subjects:
  - kind: ServiceAccount
    name: nfs-client-provisioner
    namespace: default
roleRef:
  kind: ClusterRole
  name: nfs-client-provisioner-runner
  apiGroup: rbac.authorization.k8s.io
```

5. Enter the following command to open the service, clusterrole and clusterrolebinding:

kubectl apply -f ~/STW-CaaSPv4/labs/nfs-client-storageclass/auth

You should see the following were created (not necessarily in this order):

serviceaccount “nfs-client-provisioner”
clusterrole “nfs-client-provisioner-runner”
clusterrolebinding “run-nfs-client-provisioner”

6. Enter the following command to view the service accounts:

   kubectl get serviceaccounts

   You should see **nfs-client-provisioner** listed.
7. Enter the following command to view the cluster roles:

   kubectl get clusterroles

   You should see **nfs-client-provisioner-runner** listed.

Task 3: Define a Storage Class

1. On the **CAAS101-management VM**, in the text editor of your choice, open the file:

   ~/STW-CaaSPv4/labs/nfs-client-storageclass/deploy/class.yaml

```
apiVersion: v1beta1
kind: ServiceAccount
metadata:
  name: nfs-client-provisioner
```

2. In the text editor of your choice, open the file:

   ~/STW-CaaSPv4/labs/nfs-client-storageclass/deploy/deployment.yaml

```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: nfs-client-provisioner
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: nfs-client-provisioner
    spec:
      containers:
        - name: nfs-client-provisioner
          image: rmt.example.com:5000/quay.io/external_storage/nfs-client-provisioner:latest
          volumeMounts:
            - name: nfs-client-root
              mountPath: /persistentvolumes
          env:
            - name: PROVISIONER_NAME
              value: rmt.example.com
            - name: NFS_SERVER
              value: 192.168.110.2
            - name: NFS_PATH
              value: /export/volumes
      serviceAccount: nfs-client-provisioner
      serviceAccountName: nfs-client-provisioner
      volumes:
        - name: nfs-client-root
          nfs:
            server: 192.168.110.2
            path: /export/volumes
```

3. Enter the following command to create the storage class and provisioner:

   kubectl apply -f ~/STW-CaaSPv4/labs/nfs-client-storageclass/deploy

   You should see the following were created (not necessarily in this order):

   storageclass “managed-nfs-storage”
   deployment “nfs-client-provisioner”

4. Enter the following command to view the service accounts:

   kubectl get storageclasses

   You should see **managed-nfs-storage** listed.

5. Enter the following command to view the cluster roles:

   kubectl get deployments

   You should see **nfs-client-provisioner** listed.

Task 4: Create a Persistent Volume Claim and Pod that Uses It

6. On the **CAAS101-management VM**, in the text editor of your choice, open the file:

   ~/STW-CaaSPv4/labs/nfs-client-storageclass/test/test-claim.yaml

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: test-claim
  annotations:
    volume.beta.kubernetes.io/storage-class: "managed-nfs-storage"
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Mi
```

7. In the text editor of your choice, open the file:

   ~/STW-CaaSPv4/labs/nfs-client-storageclass/test/test-pod.yaml

```
kind: Pod
apiVersion: v1
metadata:
  name: test-pod
spec:
  containers:
  - name: test-pod
    image: rmt.example.com:5000/gcr.io/google_containers/busybox:1.24
    command:
      - "/bin/sh"
    args:
      - "-c"
      - "touch /mnt/SUCCESS && exit 0 || exit 1"
    volumeMounts:
      - name: nfs-pvc
        mountPath: "/mnt"
  restartPolicy: "Never"
  volumes:
    - name: nfs-pvc
      persistentVolumeClaim:
        claimName: test-claim
```

8. Enter the following command to create the volume claim and the pod:

   kubectl apply -f ~/STW-CaaSPv4/labs/nfs-client-storageclass/test

   You should see the following were created (not necessarily in this order):

   persistentvolumeclaim “test-claim”
   pod “test-pod”

9. Enter the following command to view the persistent volume claims:

kubectl get pvc

You should see **test-claim** listed.

10. Enter the following command to view the details about the volume claim:

    kubectl describe pvc test-claim

    Under the **Events** section you should see that the provisioner successfully provisioned the volume followed by the name of the volume. Record the name of the volume here:

    ***NFS_VOLUME_NAME*=_____**

    This will be in the directory name you will see on the NFS server

11. Enter the following command to view the pods:

    kubectl get pods

    You should note see **test-pod** listed because it was defined to deploy, run a command to write a file to the volume and then exit.

12. On the **SMT server**, enter the following command to view the directory that was created for the volume:

    ls -l /export/volumes/

    You should see a directory named:

    default-test-claim-*NFS_VOLUME_NAME*

13. Enter the following command:

    ls -l /export/volumes/default-test-claim-*NFS_VOLUME_NAME*

    You should see a file named: **SUCCESS**

Task 5: Create a Persistent Volume Claim and Pod that Uses It

14. On the **CAAS101-management VM**, enter the following command to remove the volume claim:

    kubectl delete pvc test-claim

    Yous should see that the persistentvolumeclaim was deleted.

15. Enter the following command to view the persistent volume claims:

    kubectl get pvc

    You should no longer see **test-claim** listed.

16. On the **SMT server**, enter the following command to view the directory that was created for the volume:

ls -l /export/volumes/

You should see that the directory for the volume has been renamed to:

archived-default-test-claim-*NFS_VOLUME_NAME*

If desired, you could apply the manifests in the test directory again to see another directory created.

Task 6: Remove the Objects from the Cluster

1. On the **CAAS101-management VM, i**n the first terminal, enter the following commands to delete all of the objects:

   kubectl delete -f ~/STW-CaaSPv4/labs/nfs-client-storageclass/test
   kubectl delete -f ~/STW-CaaSPv4/labs/nfs-client-storageclass/deploy
   kubectl delete -f ~/STW-CaaSPv4/labs/nfs-client-storageclass/auth

   You should see that the objects were deleted.

Summary:

In this exercise, you created manifests for a service account, a role, a role binding, a storageclass, a persistent volume claim, a pod to that attached to the volume and writes data to a file in the volume. You then verified that the volume was being created and the file in it was created on the NFS server.

(End of Exercise)

6 Deploying Advanced Workload

Description:

How to deploy a workload from an existing Docker Image

6- 1 Run Heimdall as a standalone container

Description:

In this exercise, you configure and run Heimdall as standalone container

Task 1: Create a place for Heimdall to store it's files

1. On the Workstation, enter the following commands in a terminal session to create the ~/docker/heimdall folder:

   sudo mkdir -p /docker/heimdall
   sudo chmod 777 /docker -R
   cd /docker/heimdall

   The directory and file should now exist.

Task 2: Run Heimdall as a container

1. On the Workstation, enter the following commands in a terminal session to create

   sudo docker run -p 80:80 -v /docker/heimdall:/config -e TZ=Europe/London rmt.example.com:5000/heimdall

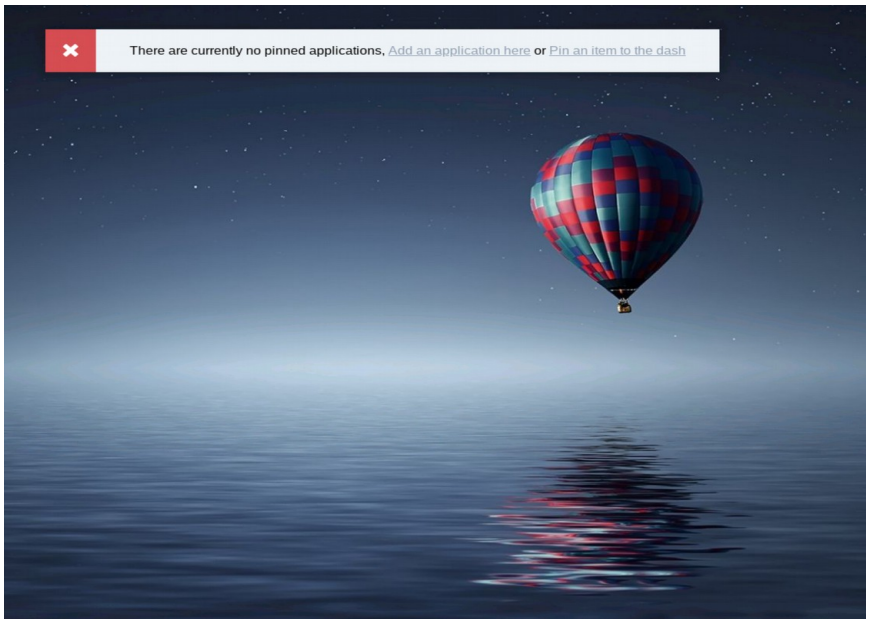
   This tells docker to run the heimdall container with the following options:
   Map port 80 in the container to 80 on the host
   mount ~/docker/heimdall in the container in /config
   Set the timezone within the container
2. Notice that the first time it runs it created a default installation

```
-------------------------------------
          _         ()
         | |  ___   _    __
         | | / __| | |  /  \
         | | \__ \ | | | () |
         |_| |___/ |_|  \__/


Brought to you by linuxserver.io
We gratefully accept donations at:
https://www.linuxserver.io/donate/
-------------------------------------
GID/UID
-------------------------------------

User uid:    911
User gid:    911
-------------------------------------

[cont-init.d] 10-adduser: exited 0.
[cont-init.d] 20-config: executing...
[cont-init.d] 20-config: exited 0.
[cont-init.d] 30-keygen: executing...
generating self-signed keys in /config/keys, you can rep
Generating a RSA private key
.....................................+++++
.....................................................
writing new private key to '/config/keys/cert.key'
-----
[cont-init.d] 30-keygen: exited 0.
[cont-init.d] 50-config: executing...
New container detected, installing Heimdall
Creating app key. This may take a while on slower system
Application key set successfully.
Setting permissions
[cont-init.d] 50-config: exited 0.
[cont-init.d] 99-custom-scripts: executing...
[custom-init] no custom scripts found exiting...
[cont-init.d] 99-custom-scripts: exited 0.
[cont-init.d] done.
[services.d] starting services
[services.d] done.
```

Task 3:On the Workstation launch the Chrome Browser

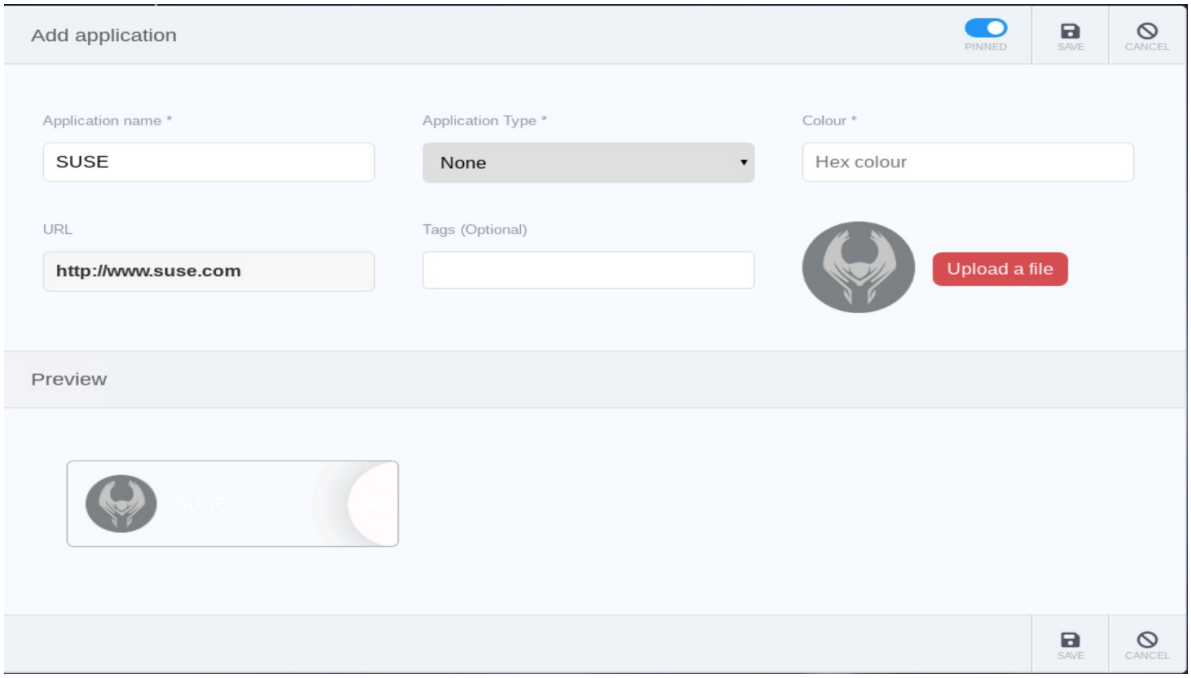
1. I On the **Workstation** **launch the Chrome Browser**

   http://127.0.0.1
2. You should see a default screen with no Apps Defined



Task 4: Create an App in Helm

1. Click on **'Add an application here'**
   Application Name: SUSE
   URL: http://www.suse.com
   Select: Pinned



2. Click Save to Save the App

Task 5: Stop and restart container

1. On the Workstation, go back to the terminal session running the container and press the following Keys to stop the container instance

   <ctrl>C

   This will terminate the container instance

2. Restart the Docker Container

   sudo docker run -p 80:80 -v ~/docker/heimdall:/config -e TZ=Europe/London smt.example.com:5000/heimdall

   Notice How it start also immediately because it's using the configuration from the last time we launched the Application

Task 6: Verify with Browser

1. On the Workstation **launch the Chrome Browser**

   http://12.0.0.1
2. You should see the App you previously defined

Task 7: Stop Container Instance

1. **On** the Workstation, go back to the terminal session running the container and press the following Keys to stop the container instance

   <ctrl>C

   This will terminate the container instance

Task 8: Try and run multiple versions of Heimdall

1. **On** the Workstation, go back to the terminal session running the container and press the following Keys to stop the container instance

   sudo docker run -d -p 80:80 -v ~/docker/heimdall:/config -e TZ=Europe/London smt.example.com:5000/heimdall

*** notice we added the -d command to tell docker to run as a Daemon rather than in the foreground**

1. Try and run a second instance of the Heimdall

   sudo docker run -d -p 80:80 -v ~/docker/heimdall:/config -e TZ=Europe/London smt.example.com:5000/heimdall

   Notice is fails because the port is already in use

Summary:

In this exercise, you launched a container from the command line to see how it behaved

(End of Exercise)

6- 2 Launch Heimdall

Description:

In this exercise, we will Heimdall without dedicated storage

Task 1: Create the Manifest for the Persistent Volume

1. On the **Workstation,** in the text editor of your choice, open the file:

   ~/STW-CaaSPv4/labs/heimdall/app-health/heimdall-deployment_ns.yaml

   Enter the following in the file:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: heimdall-shared-deployment
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: heimdall-shared
    spec:
      containers:
        - name: heimdall-pod
          image: linuxserver/heimdall
          ports:
           - containerPort: 80
```

2. On the **Workstation,** in the text editor of your choice, open the file:

   ~/STW-CaaSPv4/labs/heimdall/app-health/heimdall-service.yaml

   Enter the following in the file:

```
apiVersion: v1
kind: Service
metadata:
  name: heimdall-shared-service
spec:
  type: NodePort
  ports:
    - port: 80
      nodePort: 33000
  selector:
      app: heimdall-shared
```

Task 2: Deploy Heimdall

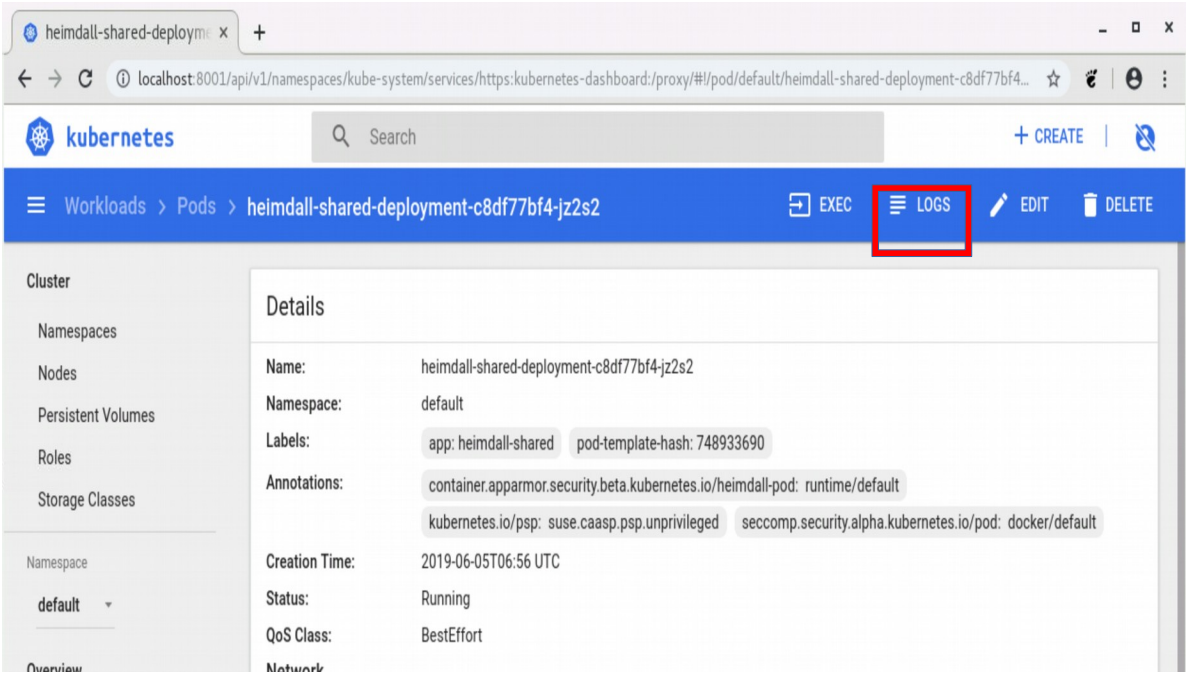
1. From a Terminal prompt on the Management workstation

   kubectl apply -f ~/STW-CaaSPv4/labs/heimdall/app-health/heimdall-deployment_ns.yaml

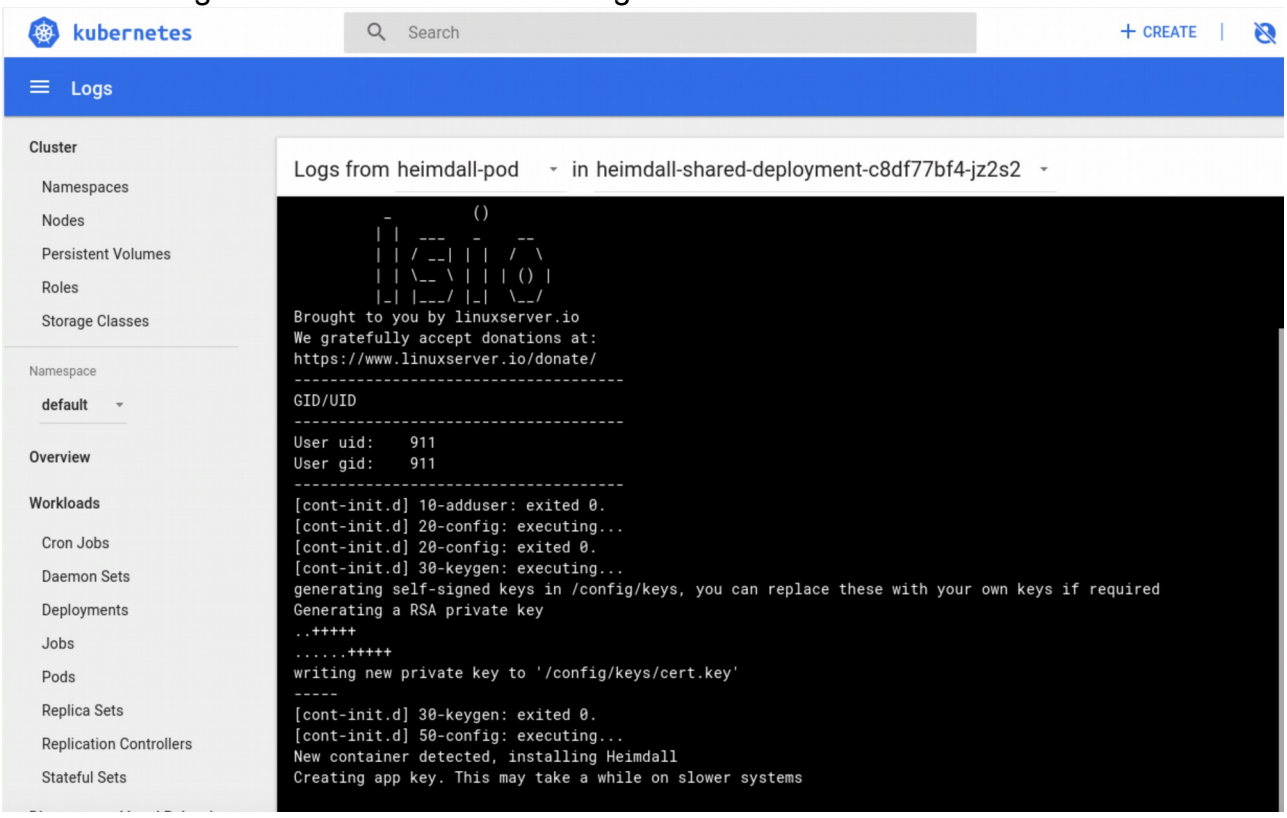
   kubectl apply -f ~/STW-CaaSPv4/labs/heimdall/app-health/heimdall-service.yaml

Task 3: View the Deployment

1. From Kubernetes Dashboard select Pods and then click on the Pod that was just deployed
2. Click on the Log button so we can see what is happening on that Pod



3. Look at the logs and notice how it is creating a new installation



4. Let that deployment continue to run until it is completely finished (Refresh the Browser to check Status)

Task 4: View Heimdall in a Browser

1. On the Workstation open the **Chrome browser** and go to

   worker.example.com:33000

2. Notice how ever time it is deployed it has to rebuild the installation
3. Stop the deployment by either killing it on the Kubernetes Dashboard from from the command line by typing in the following command

   kubectl delete -f ~/STW-CaaSPv4/labs/heimdall/app-health/heimdall-deployment_ns.yaml

Task 4: Re-deploy Heimdall

1. Repeat all of the steps in **Task 2 and Task 3**
2. Notice how ever time it is deployed it has to rebuild the installation

---

Summary:

In this exercise, you created manifests for a Heimdall. We them deployed it and watched it build the default configuration. We learned that every time we deployed the app it had to create the default installation.

(End of Exercise)

6- 3 Configure Heimdall to use NFS Persistent Storage

Description:

In this exercise, you configure a persistent volume on an NFS server. You then launch a pod that use the persistent volume

---

Task 1: Create the Manifest for the Persistent Volume

1. On the Workstation, in the text editor of your choice, open the file:

   ~/STW-CaaSPv4/labs/heimdall/app-shared/heimdall-pv-shared.yaml

Enter the following in the file:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: heimdall-shared
spec:
  capacity:
    storage: 200Mi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 192.168.110.2
    path: "/export/heimdall-shared"
```

Task 2: Create the Manifest for the Persistent Volume Claim

1. In the text editor of your choice, open the file:

   ~/STW-CaaSPv4/labs/heimdall/app-shared/heimdall-pvc-shared.yaml

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: heimdall-shared-claim
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 200Mi
```

Task 3: Create the Manifest for the Service so we can access the deployment

1. In the text editor of your choice, open the file:
2. **~/STW-CaaSPv4/labs/heimdall/app-shared/heimdall-service.yaml**

```
apiVersion: v1
kind: Service
metadata:
  name: heimdall-shared-service
spec:
  type: NodePort
  ports:
    - port: 80
      nodePort: 30000
  selector:
      app: heimdall-shared
```

Task 4: Create the Manifests for the Web Frontend

1. In the text editor of your choice, create/open the file:
2. **~/STW-CaaSPv4/labs/heimdall/app-shared/heimdall-deployment.yaml**

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: heimdall-shared-deployment
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: heimdall-shared
    spec:
      containers:
        - name: heimdall-pod
          image: linuxserver/heimdall
          volumeMounts:
            - name: heimdall-pvc-shared
              mountPath: "/config"
      volumes:
        - name: heimdall-pvc-shared
          persistentVolumeClaim:
            claimName: heimdall-shared-claim
```

Task 5: Deploy the Objects

1. To deploy the volumes/pods/service, open a terminal and enter the following command:

   kubectl apply -f ~/STW-CaaSPv4/labs/heimdall/app-shared

   You should see the following were created (not necessarily in this order):

   deployment.extensions "heimdall-shared-deployment" created
   persistentvolume "heimdall-shared" created
   persistentvolumeclaim "heimdall-shared-claim" created
   service "heimdall-shared-service" created

2. Enter the following command to view the deployments:

   kubectl get deployments

   You should see the **heimdall-shared-deployment** deployments listed.

3. Enter the following command to view the persistent volumes:

   kubectl get pv

   You should see the persistent volume **heimdall-shared** listed.
4. Enter the following command to view the persistent volumes:

   kubectl get pvc

   You should see the persistent volume claim **heimdall-shared-claim** listed.

Task 6: Test the Persistent Data

1. On the **management workstation**, open a web browser and point to:

   http://worker10.example.com:30000

   Notice we have a Heimdall session pre-populated with with 2 Applications

Task 7: Remove the Objects from the Cluster

1. In the first terminal, enter the following commands to delete all of the objects:

   kubectl delete -f ~/STW-CaaSPv4/labs/heimdall/app-shared

   You should see that the objects were deleted.

Summary:

In this exercise, you created manifests for a persistent volume, a persistent volume claim, a pod to that attached to the volume and writes data to an index.html file in the volume, and a web server that attaches to the volume and displays the index.html file. You then verified that the index.html file was being updated by looking at the file both on the NFS volume and the web server.

(End of Exercise)