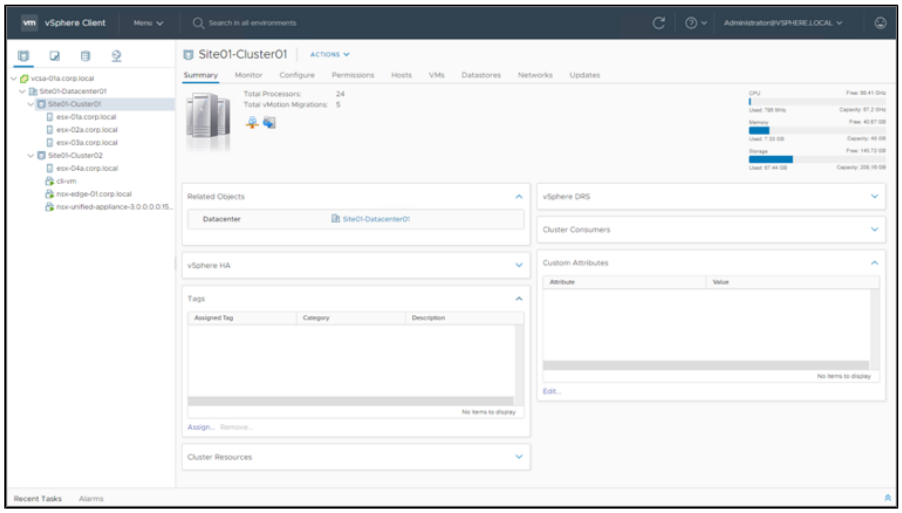
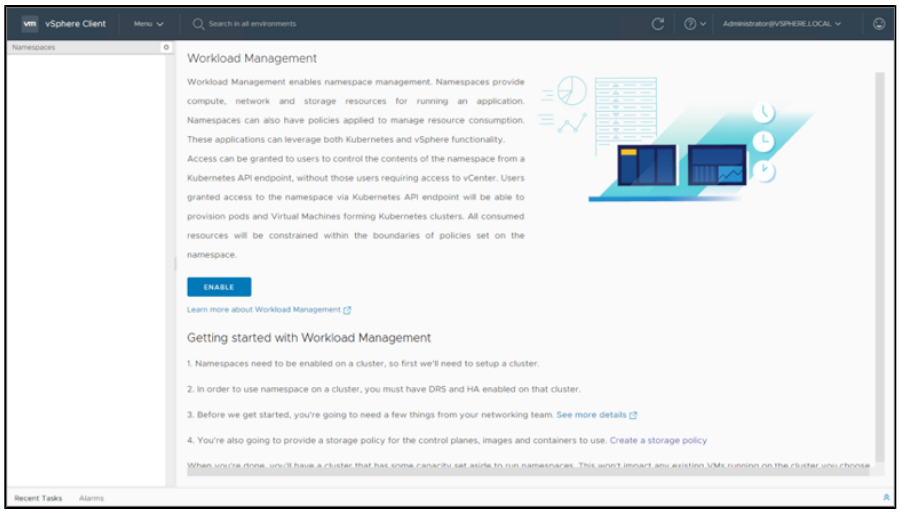
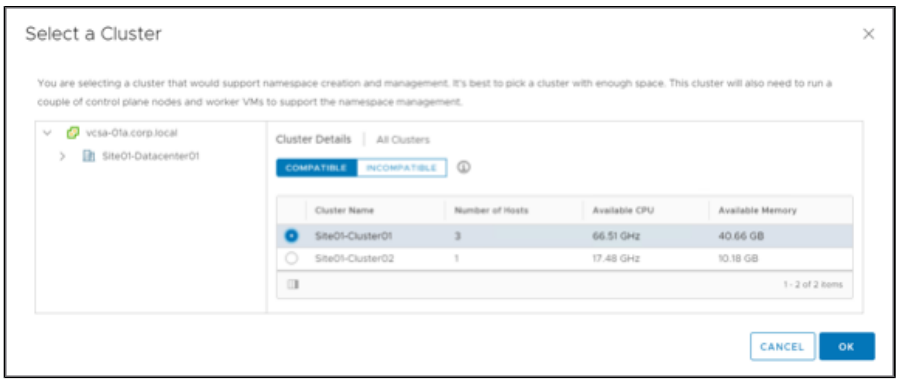
Project Pacific - Lab Manual

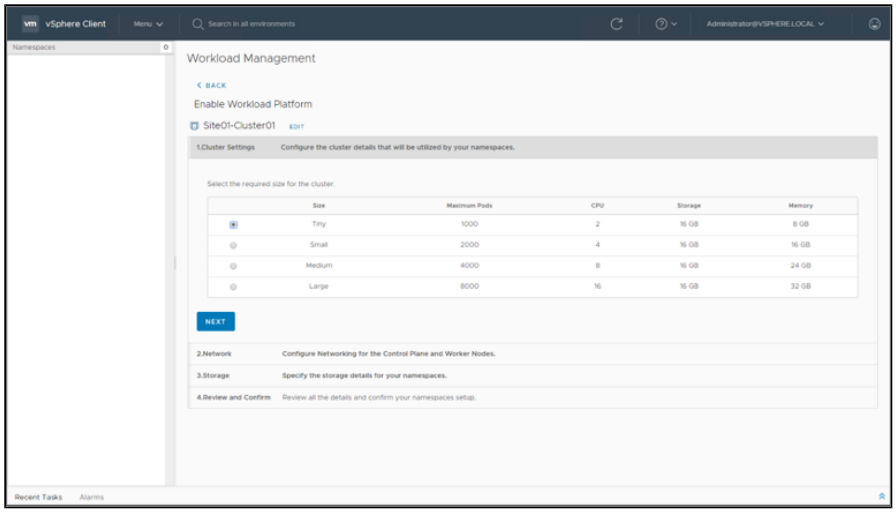
Connect to vCenter	1) Connect to vCenter: Open Chrome and click on `vcsa-01a.corp.local` Bookmark Username: `administrator@vsphere.local` Password: `VMware1!`  2) Go to Hosts and Clusters - Show vSphere clusters `Site01-Cluster01` and `Site01-Cluster02`  3) Note that there are two vSphere Clusters, but the Supervisor Cluster has not been enabled.
Enable Supervisor Cluster	1) Click `Menu` -> `Workload Management` 

2) Click `Enable`

3) Select `Site01-Cluster01` -> `OK`

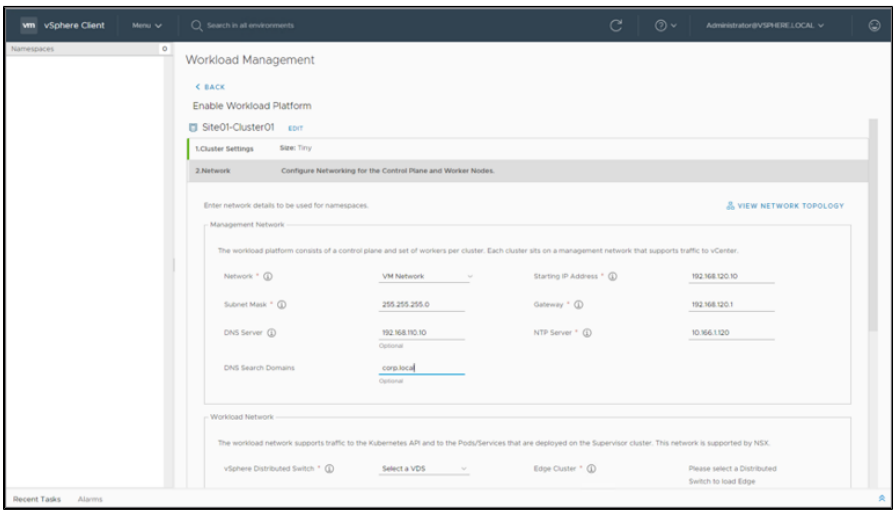


4) Select `Tiny` -> `Next`



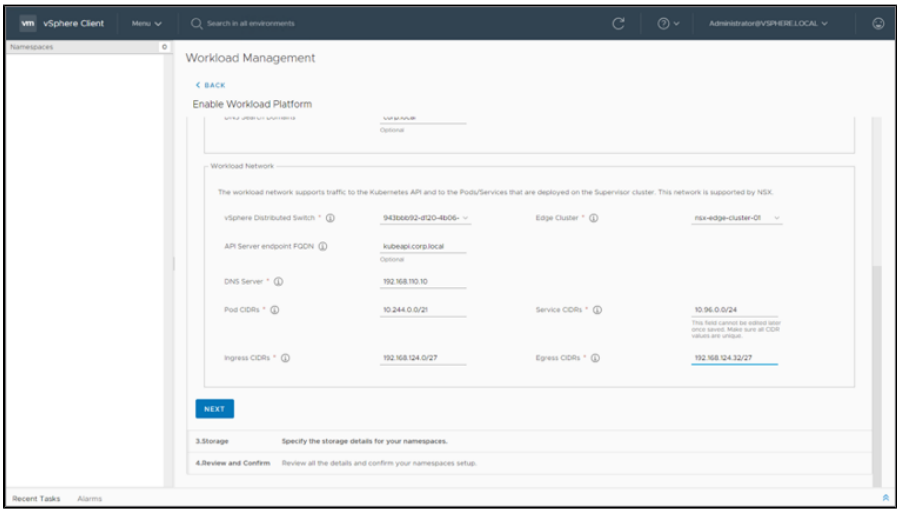
5) For Management Network

a. Network: `VM Network`
b. Starting Master IP: `192.168.120.10`
c. Subnet Mask: `255.255.255.0`
d. Gateway: `192.168.120.1`
e. DNS Server: `192.168.110.10`
f. NTP Servers: `192.168.110.10`
g. DNS Search: `corp.local`



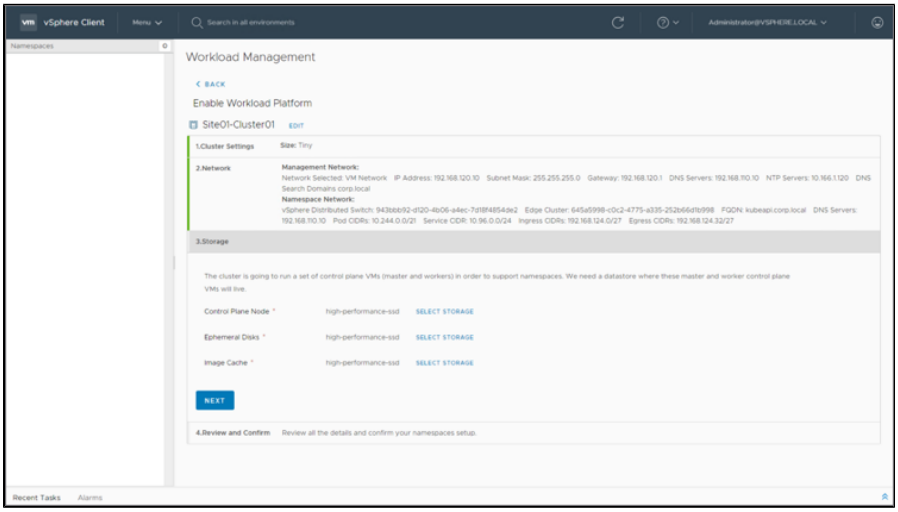
6) For Workload Network

a. vSphere Distributed Switch: Only One in List
b. Edge Cluster: `NSX-Edge-Cluster01`
b. API Server Endpoint: `kubeapi.corp.local`
d. DNS: `192.168.110.10`
e. Ingress CIDRs: `192.168.124.0/27`
f. Egress CIDRs: `192.168.124.32/27`
g. Click Next



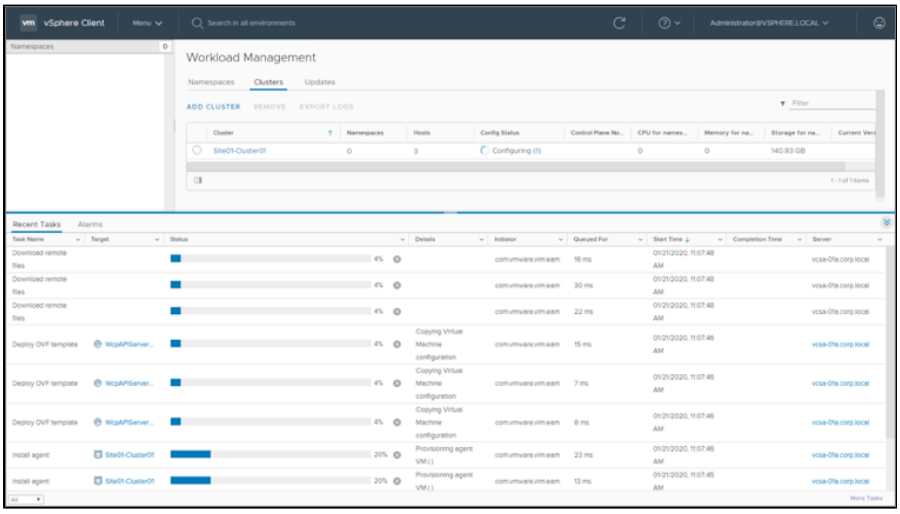
7) Storage

a. Control Plane Node: `high-performance-ssd`
b. Ephemeral Disks: `high-performance-ssd`
c. Image Cache: `high-performance-ssd`
d. Click Next and Finish

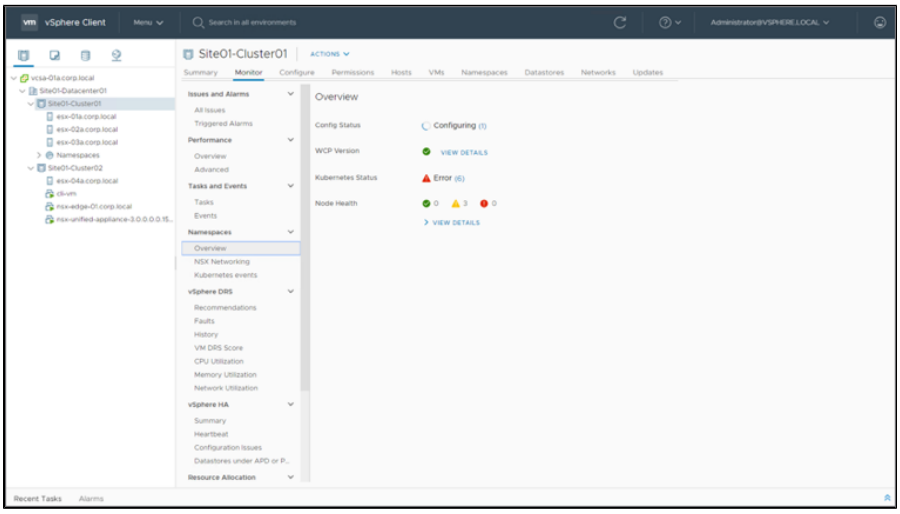


e. Wait 28 up to 50 mins

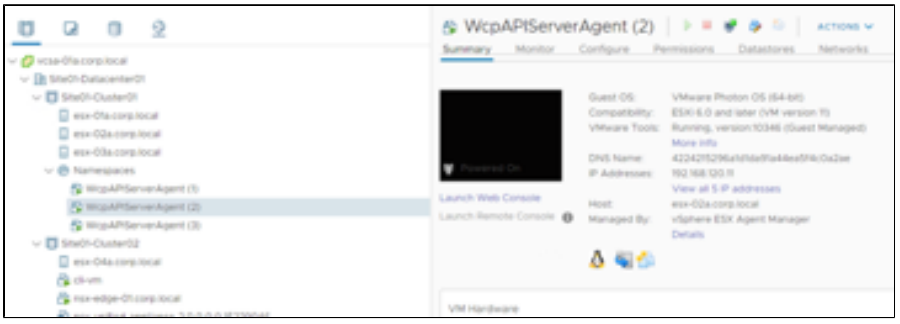
i. You can check tasks and see `WcpAPIServerAgent` VMs deploying.



ii. You can check progress by going to
`Host & Clusters` -> `Site01-Cluster01` -> `Monitor` -> `Namespace` -> `Overview`

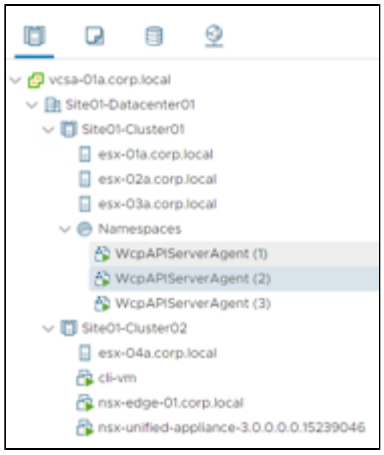


iii. You can also note progress by checking the IPs on the management VMs.
At completion, two of them will have 4 IPs and one will have 5 IPs.



iv. Note, there may be errors and warnings seen during enabling the supervisor cluster.
They will go away after a successful enablement.

8) Click on `Namespaces` in Hosts and Clusters Inventory.
This is a new Resource Pool that will include all of the namespaces you create.
Note the Kubernetes ControlPlaneAPI VMs are running.



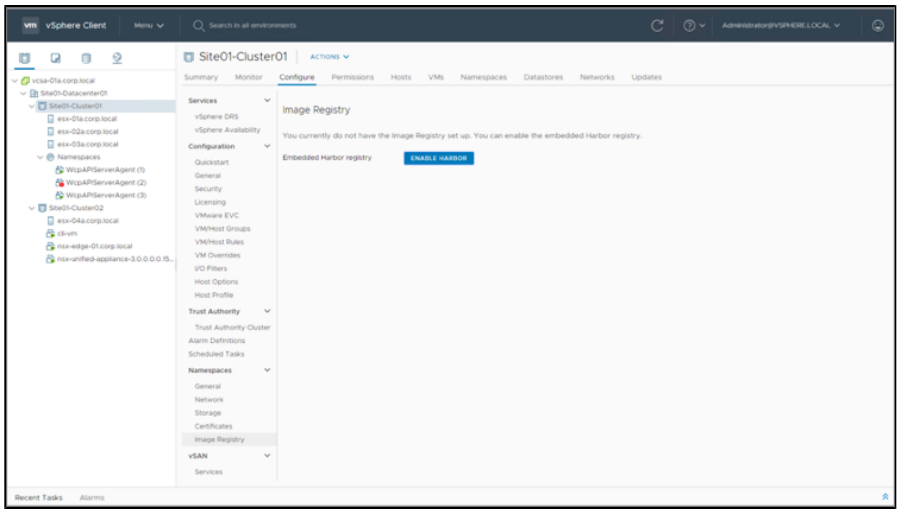
Stop Here!! We will go back to the presentation while the cluster is being created.

Enable Harbor Registry

1) Click on `Hosts and Clusters`

2) Select `Site01-Cluster01`

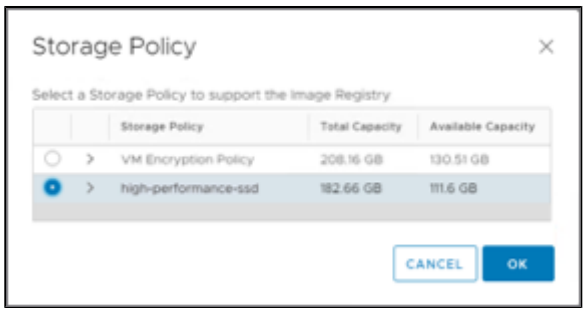
3) Click on `Configure`



4) Under Namespace select `Image Registry`

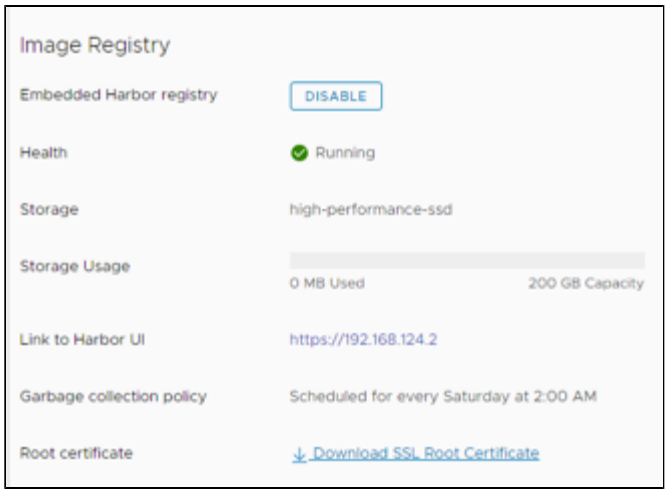
5) Click `Enable`

6) Select `high-performance-ssd` storage policy

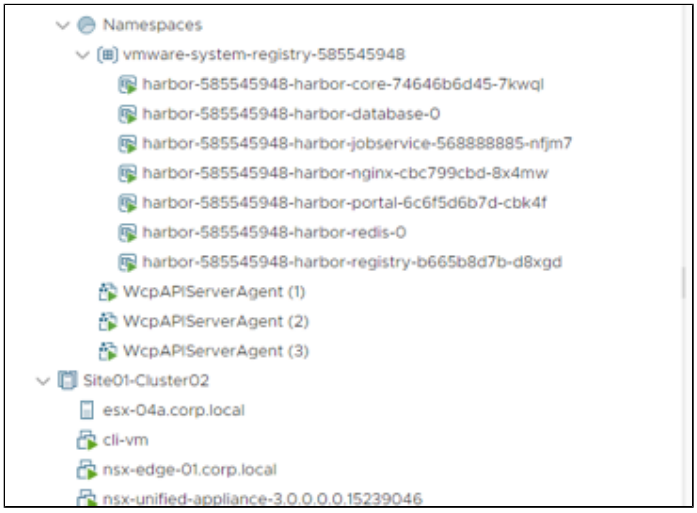


7) Click `OK`

8) When complete, note the Virtual Server IP to access Harbor.
(Beta UI issue means you need to log out and back in to see updated screen)



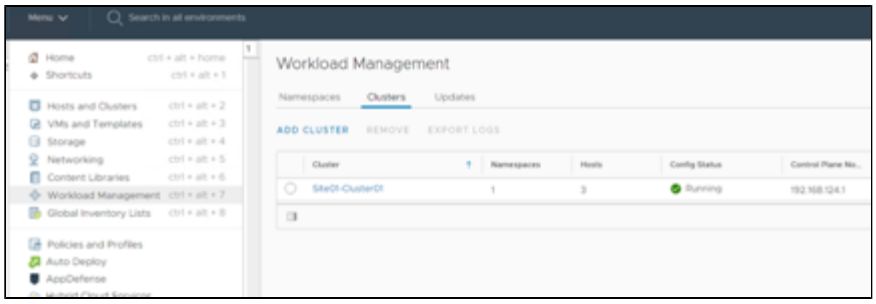
You will see a new Resource Pool created under Namespaces and a set of Pods deployed on ESXi to enable the Harbor Registry service. It will take about 10 minutes to complete.



Create Namespace

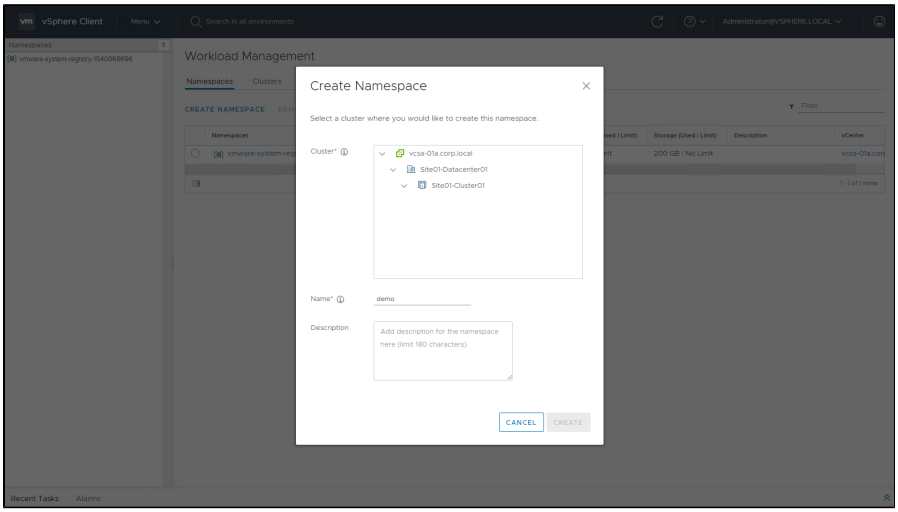
1) Click on Menu -> `Workload Management`

2) Click on `Clusters`. This is a view of your existing clusters with IP of Kubernetes API LB endpoint

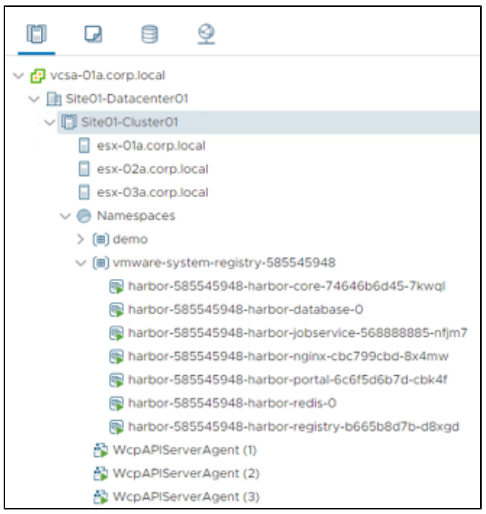


3) Click on `Namespaces` -> `Create Namespace` -> `Site01-Datacenter01` -> `Site01-Cluster01`

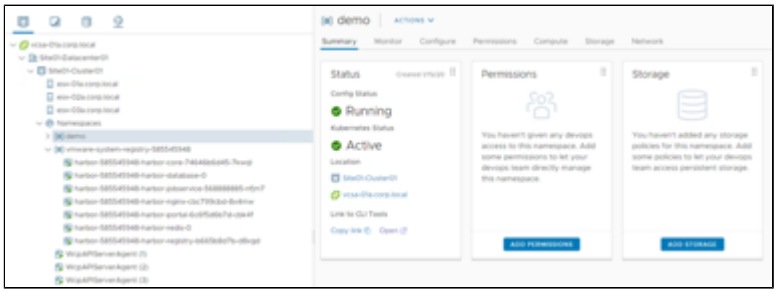
a. Make sure to highlight `Site01-Cluster01` before adding Name
b. Type in demo (Lower case because it will be k8 namespace name) as the name
c. Click Create
d. This has created a Namespace in the Supervisor cluster and has also created a VC namespace object.

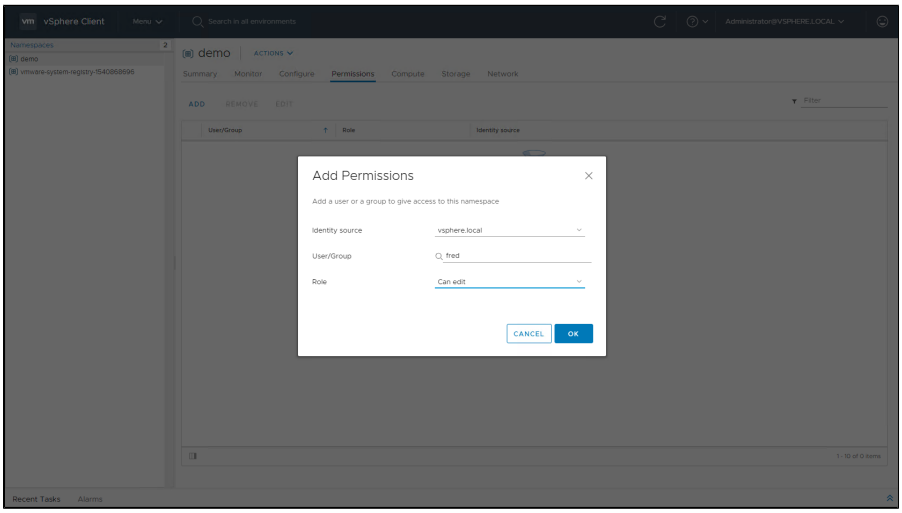
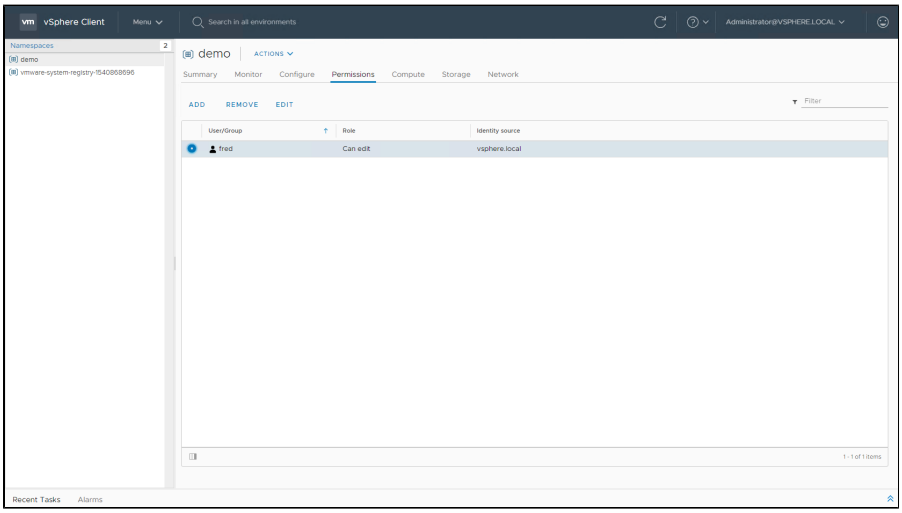


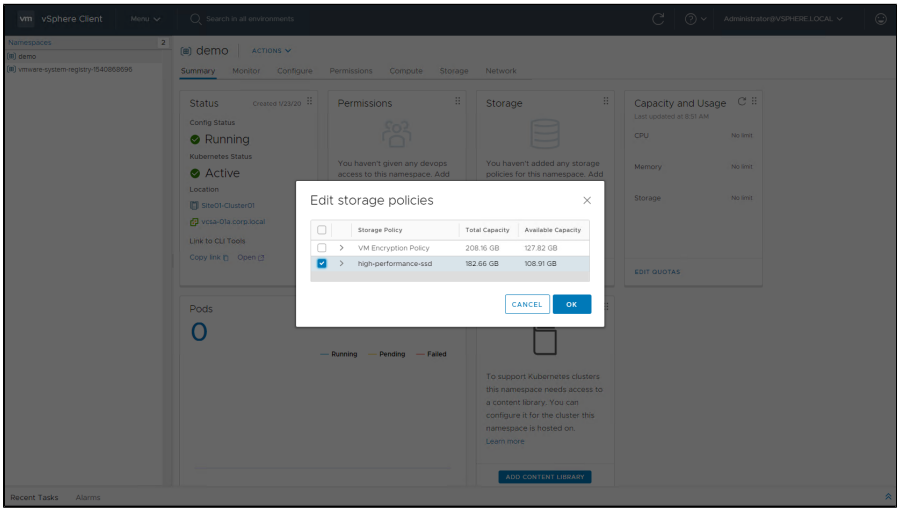
4) Go back to `Hosts and Clusters` View and note the Resource Pool under `Namespaces`.

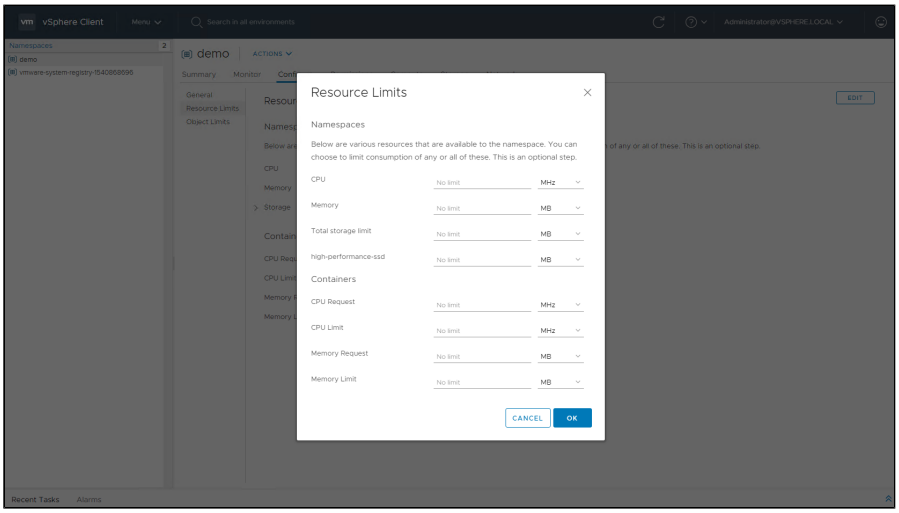
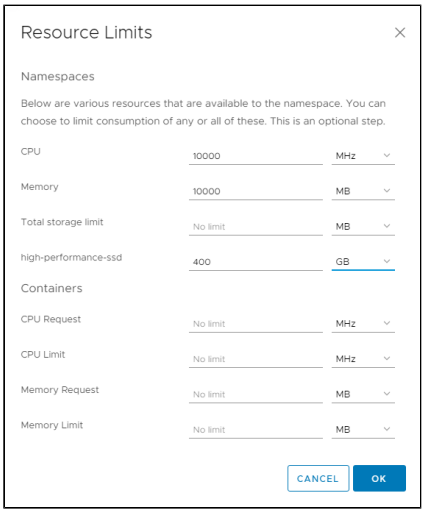


Click on the Resource Pool demo to go back to Namespace `Summary` page.

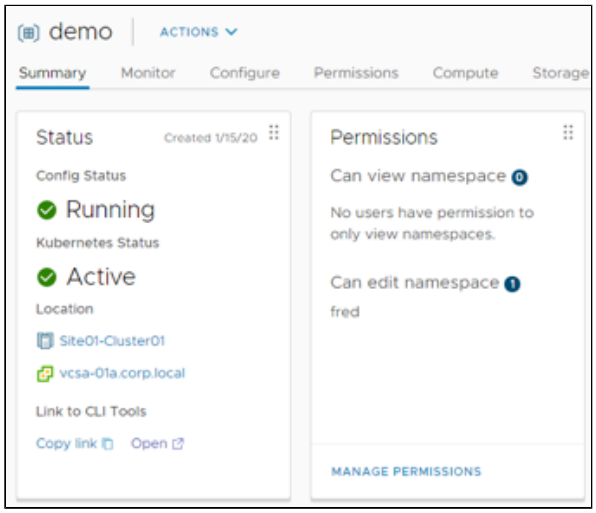


Add Permissions to Namespace	1) Add Developer access to the Cluster a. If you aren't on the Namespace `Summary` Page: Click `Menu` -> `Workload Management` -> `demo Namespace` b. Click `Add Permissions` c. Identity Source: Choose `vsphere.local` d. User/Group: Type `Fred` e. Role: `Can Edit` -> OK   This created a RoleBinding on your Namespace in the Supervisor Cluster that binds the user Fred to a ClusterRole called Edit. If you can not log in as user Fred, change the password and add the e-mail address fred@vsphere.local in the vSphere Client.

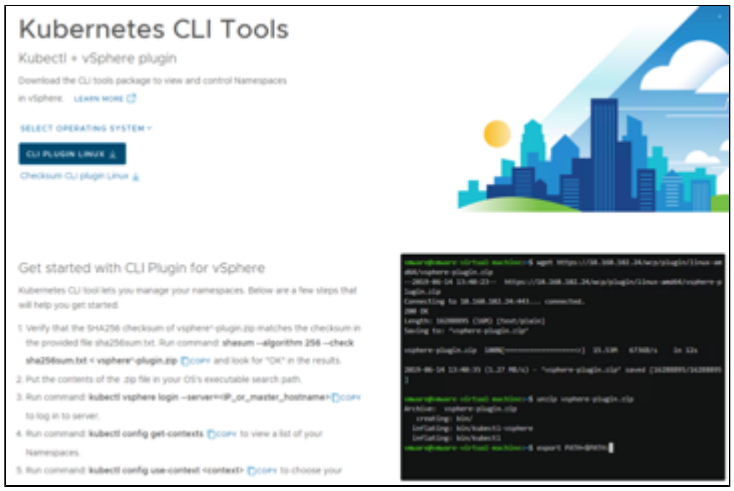
Add Storage Policy to Namespace	1) Click `Add Storage` 2) Select `high-performance-ssd` 3) Click Arrow to Expand View and see the Datastores defined by the Storage Policy 4) Click OK  This causes a Storage Class to be created on the Supervisor Cluster. It also causes an Unlimited ResourceQuota to be assigned for the Namespace on that storage Class. You can change that limit in the Config section.

Set CPU, Memory & Storage Limits on Namespace	1) Click `Configure` 2) Click `Object Limits` -> Note Kubernetes object limits can be set  3) Click `Resource Limits` 4) Click `Edit` 5) Set `CPU` = `10000 MHz` 6) Set `Memory` = `10000 MB` 7) Set `high-performance-ssd` = `400 GB` 8) Click `OK`  Set Container Limits at your own risk. Let Developers define in their Podspec or let WCP service do it.
Download & install kubectl and vSphere Plugin	The download contains standard Opensource Kubectl. Kubectl has a pluggable architecture. We have created a plugin called `kubectl-vsphere` that extends the commands available to `kubectl`. You will use this extension to authenticate to vSphere SSO. 1) Click `Summary`

2) Click Open from the Status Pane under Link to CLI Tools

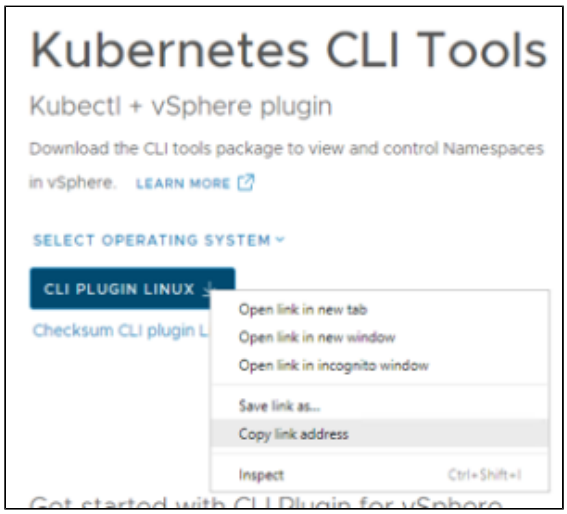


3) Click `Select Operating System`

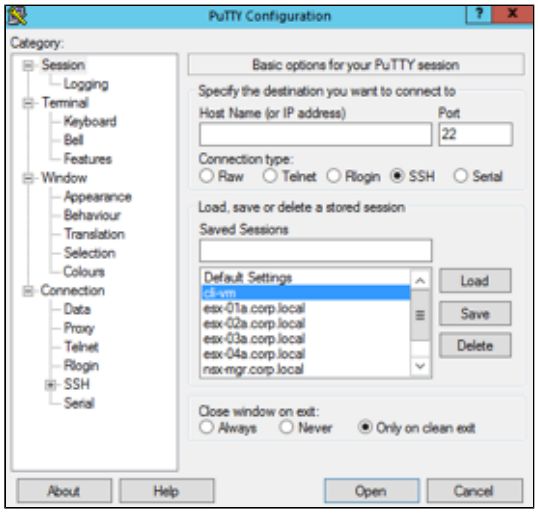


4) Select `Linux`

5) Right Click `CLI Plugin for Linux`



6) Click `Copy Link Address`

	7) Open `PuTTY` and select `cli-vm` and then `Open`  8) Download `kubectl` zip. Note: Make sure you are in the `$HOME` directory 9) `wget https://192.168.124.1/wcp/plugin/linux-amd64/vsphere-plugin.zip \` `--no-check-certificate` 10) `unzip vsphere-plugin.zip` 11) `sudo cp bin/* /usr/local/bin` (Password: `VMware1!`)

Authentication & Kubernetes Context	1) Type `kubectl vsphere login --server=kubeapi.corp.local \`
	`--vsphere-username fred@vsphere.local \`
	`--insecure-skip-tls-verify`
	
	2) Password: `VMware1!`
	
	3) This login created a config file with the JWT token needed to authenticate to the Kubernetes API. Notice that you can see the Namespaces that you have access to.
	
	4) `cat ~/.kube/config` to see the contexts that were set up for you

5) Notice that there is a token for Fred on the LB IP 192.168.124.1 and for the API FQDN you specified at deployment.

```
Pacific-Demo:~: cat .kube/config
apiVersion: v1
clusters:
- cluster:
    insecure-skip-tls-verify: true
    server: https://192.168.124.1:6443
  name: 192.168.124.1
- cluster:
    insecure-skip-tls-verify: true
    server: https://kubeapi.corp.local:6443
  name: kubeapi.corp.local
contexts:
- context:
    cluster: 192.168.124.1
    namespace: demo
    user: wcp:192.168.124.1:fred@vsphere.local
  name: demo
- context:
    cluster: kubeapi.corp.local
    user: wcp:kubeapi.corp.local:fred@vsphere.local
  name: kubeapi.corp.local
current-context: demo
kind: Config
preferences: {}
users:
- name: wcp:192.168.124.1:fred@vsphere.local
  user:
    token: eyJraWQiOiJERkU3ODM2RTJGMUREMzU0NkU4NjdDQzkzRT
3Zjc2EtMDFhLmNvcnAubG9jYWxcL29wZW5pZGNvbm5lY3RcL3ZzcGhlcm
c2VybmFtZSI6ImZyZWQifQ.VaUiPPJZFBw3w6NlKEMVBubrySkmZhPbHw
C0O6E8S60m7qyG-IrO6Msh5x9ECNZV_5DoB_FwhGgzor-W6mYf7kzWUBH
- name: wcp:kubeapi.corp.local:fred@vsphere.local
  user:
    token: eyJraWQiOiJERkU3ODM2RTJGMUREMzU0NkU4NjdDQzkzRT
3Zjc2EtMDFhLmNvcnAubG9jYWxcL29wZW5pZGNvbm5lY3RcL3ZzcGhlcm
c2VybmFtZSI6ImZyZWQifQ.NQ1_5zRJj0ELNjdw8P5kQFgJmkLhw-Fdsz
nD7fSj7FAw20cde3D9pfC2sZAanqt2I5nkT9wxGHqDNkGNbq1ZaiaHhtY
Pacific-Demo:~:
```

6) Type `kubectl config get-contexts`

7) Type `kubectl config use-context demo`
This sets the context to point to your namespace in the Supervisor Cluster

```
Pacific-Demo:~: kubectl config use-context demo
Switched to context "demo".
Pacific-Demo:~:
```

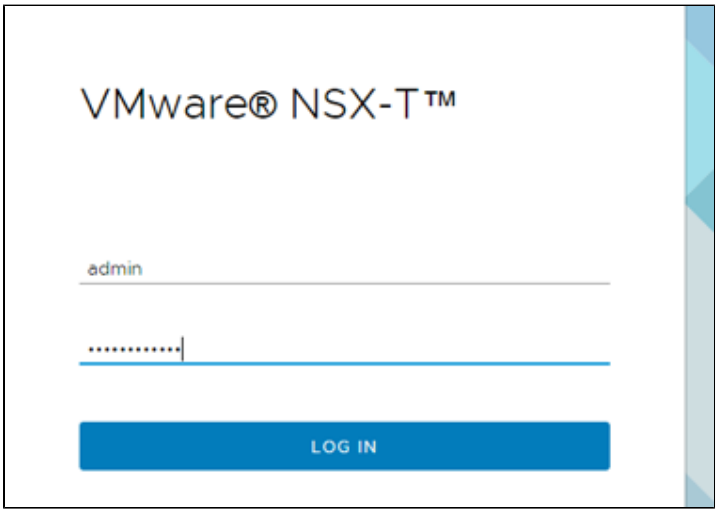
8) Type `kubectl get all`
You should get a `No Resources Found` response

```
Pacific-Demo:~: kubectl config use-context demo
Switched to context "demo".
Pacific-Demo:~: kubectl get all



No resources found.
Pacific-Demo:~:
```

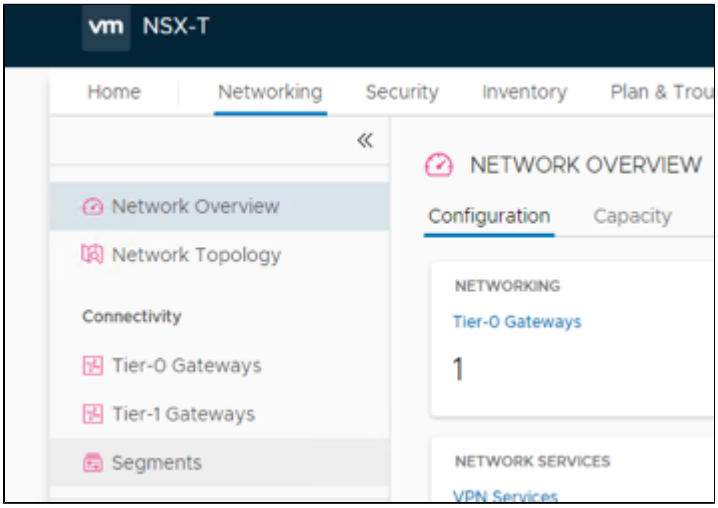
NSX Objects created through Pacific

1) From Chrome, Open a new Tab and Select `nsx-mgr.corp.local` bookmark

2) Username: `admin` Password: `VMware1!base`



3) Click `Networking`



4) Click on `Segments`

5) You will see Logical Switches (Now called Segments) for each Namespace.

6) Seems to be an issue with expanding this column to get the full segment name.
Click the arrow to expand each segment.
Check the the IP-address-Pool for a name with demo in it.
Was the third Segment in my test environment.

7) Notice the CIDR for your Namespace. This segment came from the Pod CIDR defined on cluster create.

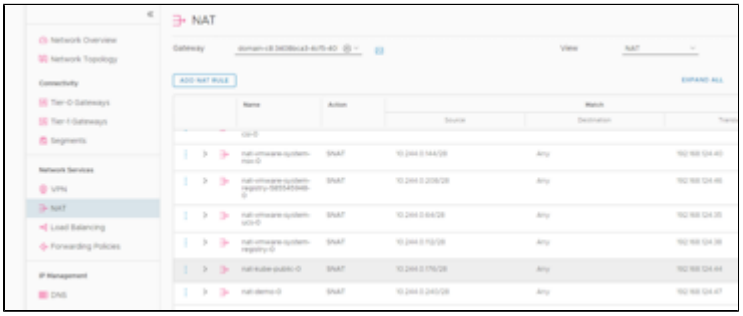
8) Click on `Tier1-Gateways`

9) Note the single T1 Gateway for your cluster – Linked to a Single T0 Gateway and all of your segments

10) Click on NAT, then Select the Logical Router starting with `domain-c8:####`.

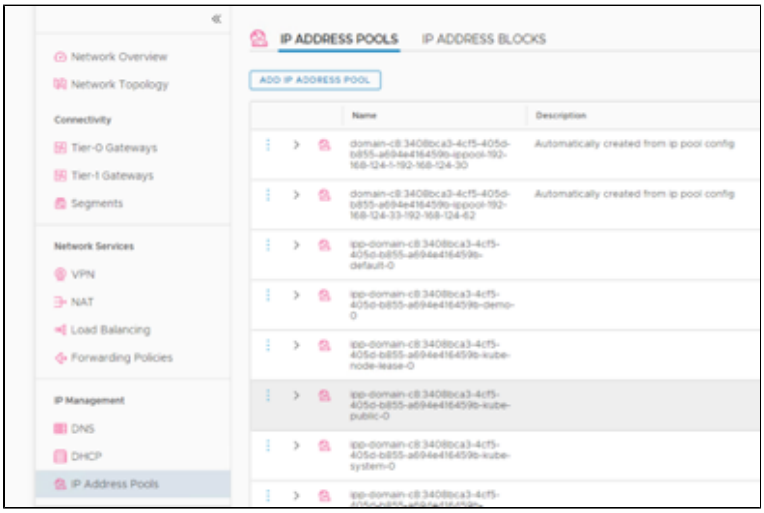
11) Note that there is an SNAT IP for each Namespace in the cluster and for the Master.

12) Find the CIDR for your Namespace and note that all traffic from Pods in this Namespace will have that translated IP. These IPs come from the Egress CIDR range and must be routable from your corporate network.



13) DHCP not implemented in the Beta

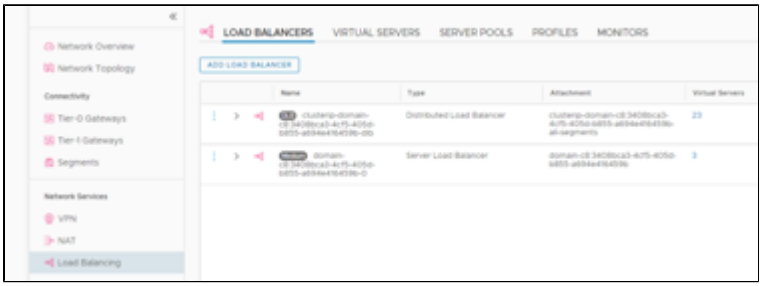
14) Click `IP Address Pools IP Block` to see your Pod CIDR subnets.



15) DHCP not implemented in the Beta

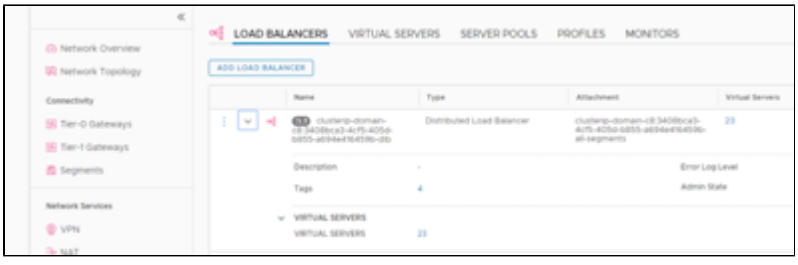
16) Click on `Load Balancers`

17) Select the Medium `domain-c8###` Load Balancer.
This is the Load Balancer for access to the MasterAPI and Harbor from external users.

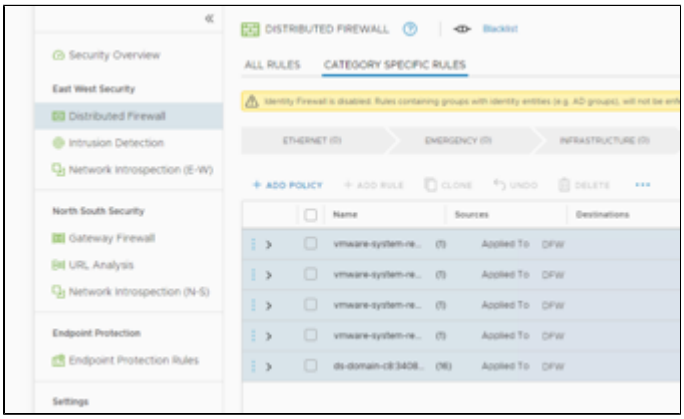


18) Click on `Virtual Servers` and note the IPs. These are the IPs you logged in with for Harbor and the API Master.
Note: Beta bug screen errors. You can find this under Advanced Network and Security -> Load Balancers -> Virtual Servers

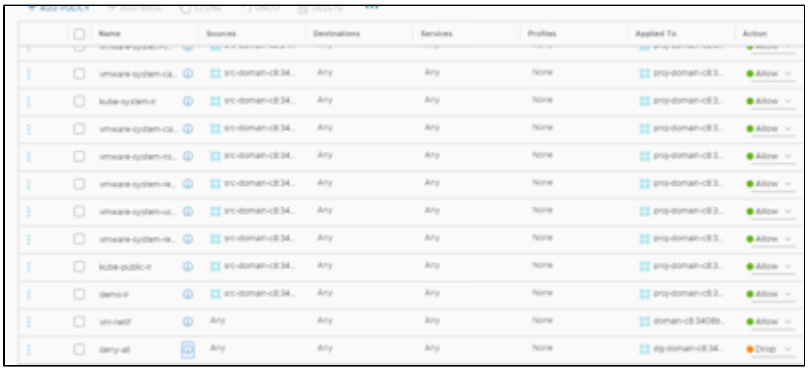
19) The LB labelled `clusterip-domain###`.
It is used for internal cluster routing between services.
This is a new Distributed Load Balancer capability in NSX 3.0.



20) Click on Security -> Distributed Firewall



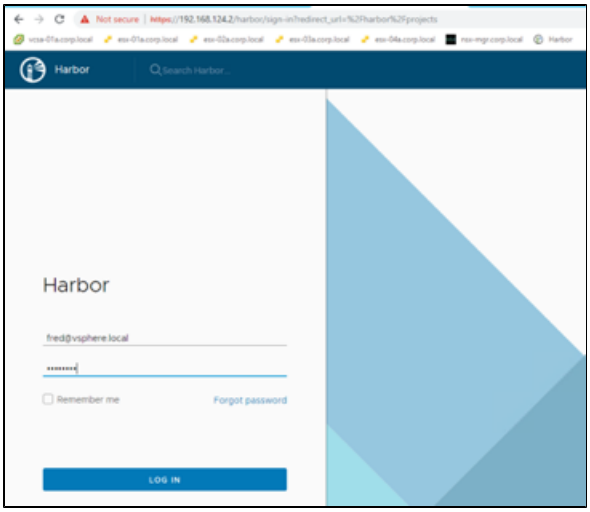
21) Expand the line that starts `ds-domain-c8:####`.
Notice the Deny-all Drop rule that will prevent all ingress/egress by default.
Keep this in mind as we deploy an application.



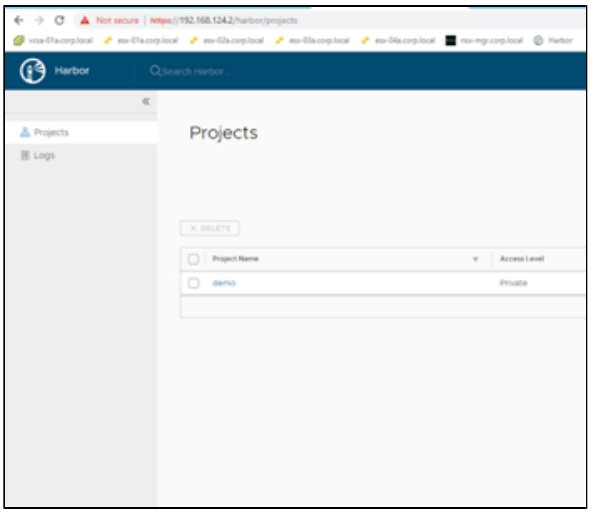
Access Harbor Registry

1) Note the IP from the enable page & click the link - or choose the Harbor bookmark in Chrome.
If you did the lab in order, the url should be `http://192.168.124.2:443`

2) Log in with the credentials you used for your Namespace:
`fred@vsphere.local`
Password: `VMmware1!`



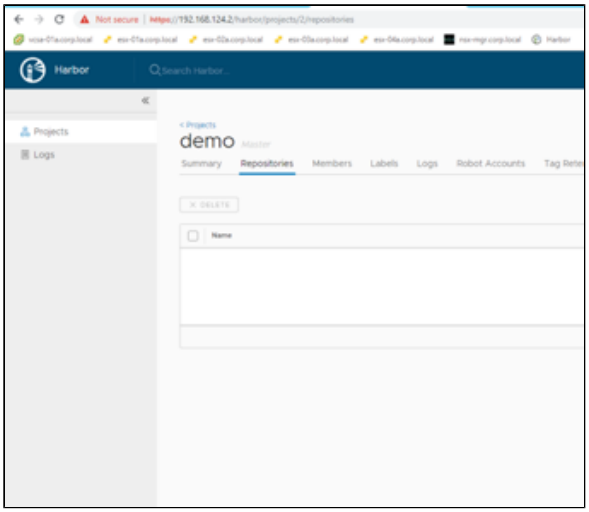
3) Note that the project for the demo Namespace was created and that your Namespace credentials were updated in Harbor



4) Click on `demo` Project

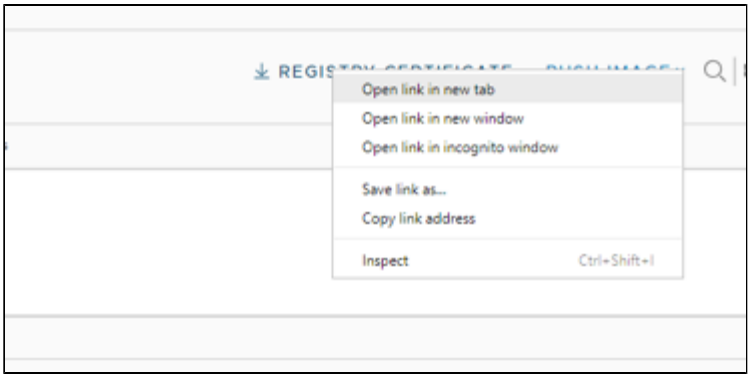
5) You see that we have not yet added any Repos

6) Click on `Repositories`



7) We are going to add a Repository to Harbor.

8) Right Click on `Registry Certificates`



9) Click `Copy Link Address`

Setup Docker Credentials and add Harbor Certificate:

Setup Docker Access and deploy Application image

1) From the `cli-vm`, Type `sudo -i` Password: `VMmware1!`

2) Type `mkdir /etc/docker/certs.d/192.168.124.2`

3) Type `cd /etc/docker/certs.d/192.168.124.2`

4) Type `wget “paste cert link address” -O ca.crt --no-check-certificate`

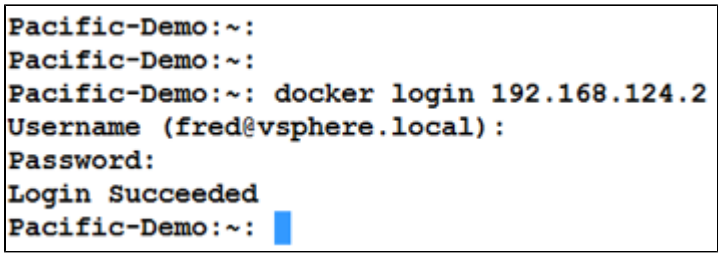
5)Type `systemctl restart docker`

6) Type `exit`

7) Type `docker login 192.168.124.2`

8) Username: `fred@vsphere.local`

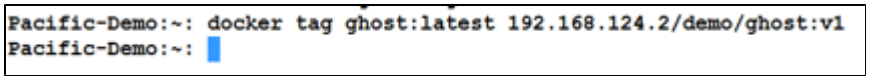
9) Password: `VMmware1!`



Tag the local image and push it to the Harbor Registry

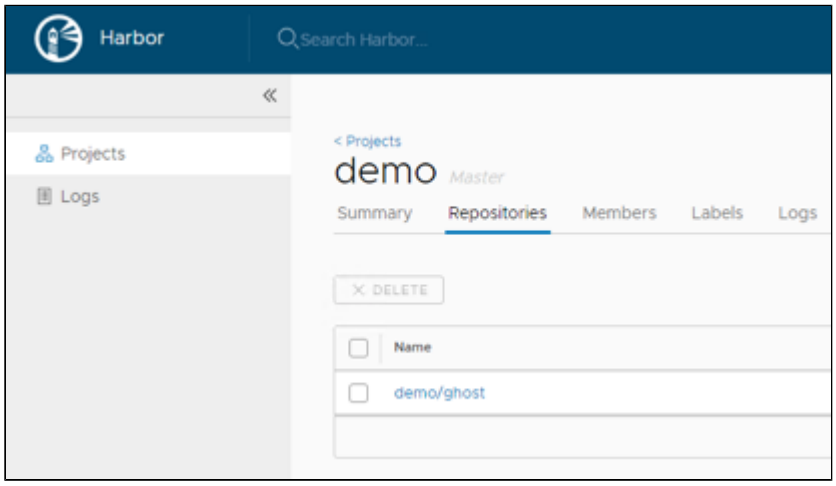
10) `docker pull ghost:latest`

11) `docker tag ghost:latest 192.168.124.2/demo/ghost:v1`



12) `docker push 192.168.124.2/demo/ghost:v1`

13) Verify its there through the Harbor UI. Refresh the page



Create Persistent Volume and Deploy Ghost app from that image.

14) `cd $HOME/labs/ghost`

15) vi `ghost-claim.yaml, set storage: 300Mi (must be <400Mb)`

```
Pacific-Demo:~: cd $HOME/labs/ghost
Pacific-Demo:~/labs/ghost: cat ghost-claim.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: blog-content
  annotations:
    volume.beta.kubernetes.io/storage-class: high-performance-ssd
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi

Pacific-Demo:~/labs/ghost:
```

16) `kubectl apply -f ghost-claim.yaml`

```
Pacific-Demo:~/labs/ghost: kubectl apply -f ghost-claim.yaml
persistentvolumeclaim/blog-content created
Pacific-Demo:~/labs/ghost:
```

17) `kubectl get pvc` (Notice volume bound to the claim)

```
Pacific-Demo:~/labs/ghost: kubectl get pvc
NAME           STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS           AGE
blog-content   Bound    pvc-25da454d-9fac-4806-9402-c1c4bf1d07ff   2Gi        RWO            high-performance-ssd   33s
Pacific-Demo:~/labs/ghost:
```

18) `cat ghost.yaml`. Note the Load Balancer Service, the image pulled from harbor and the use of the persistent volume claim.

```
Pacific-Demo:~/labs/ghost: cat ghost.yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    name: blog
  name: blog
spec:
  ports:
    - port: 80
      targetPort: 2368
  selector:
    app: blog
  type: LoadBalancer
---
```

```
    spec:
      containers:
      - name: blog
        image: 192.168.124.2/demo/ghost:v1
        imagePullPolicy: Always
        ports:
        - containerPort: 2368
        env:
        - name: url
          value: http://my-blog.corp.local
        volumeMounts:
        - mountPath: /var/lib/ghost/content
          name: content
      volumes:
      - name: content
        persistentVolumeClaim:
          claimName: blog-content
```

19) `kubectl apply -f ghost.yaml`

```
Pacific-Demo:~/labs/ghost: kubectl apply -f ghost.yaml
service/blog created
deployment.apps/blog created
Pacific-Demo:~/labs/ghost:
```

20) `kubectl get all` (Note that Pod will be Pending while image is pulled and container is started)

```
Pacific-Demo:~/labs/ghost: kubectl get all
NAME                        READY   STATUS    RESTARTS   AGE
pod/blog-8599498b5b-fwnc6   0/1     Pending   0          38s

NAME           TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)        AGE
service/blog   LoadBalancer   10.96.0.128    192.168.124.3   80:32566/TCP   38s

NAME                   READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/blog   0/1     1            0           38s

NAME                              DESIRED   CURRENT   READY   AGE
replicaset.apps/blog-8599498b5b   1         1         0       38s
```

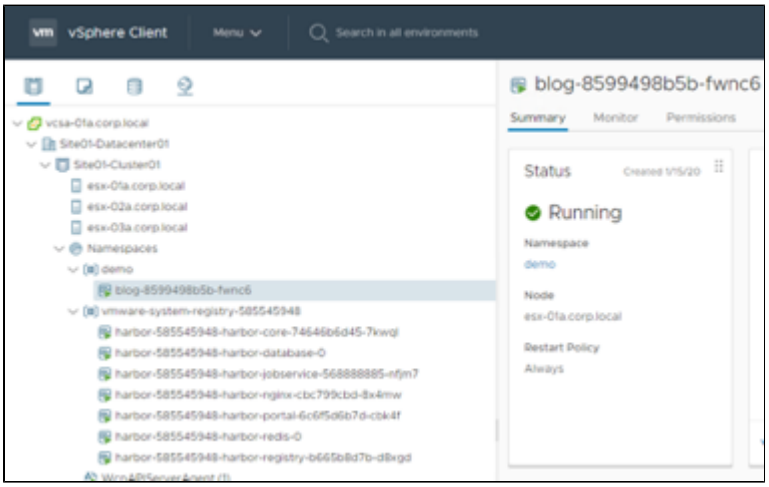
21) `kubectl describe pod/blog-#######`
(Check Events section for Pod Startup Steps and Status.)

This will show you status of the pod startup

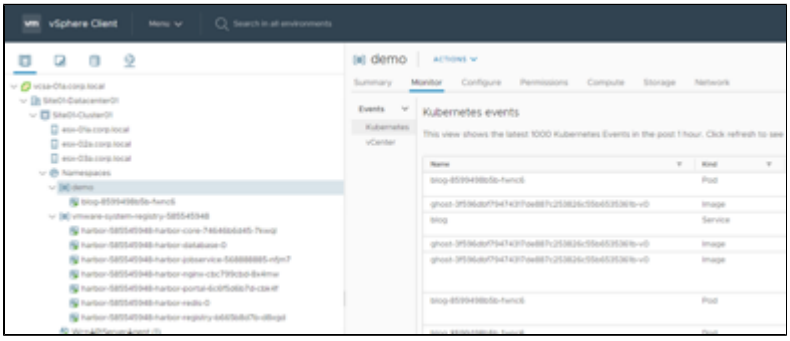
Manage Kubernetes Objects from vCenter

1. From Chrome: Got to vCenter tab: Click on `Hosts and Clusters`
2. Click on `Namespaces` -> `demo` Namespace Resource Pool
3. Click on `blog-######` Native Pod
    a. Note Pending/Running status

b. This is a new summary page for vSphere Native Pods. Notice that there is no console and that you cannot take action on these objects except through the Kubernetes API



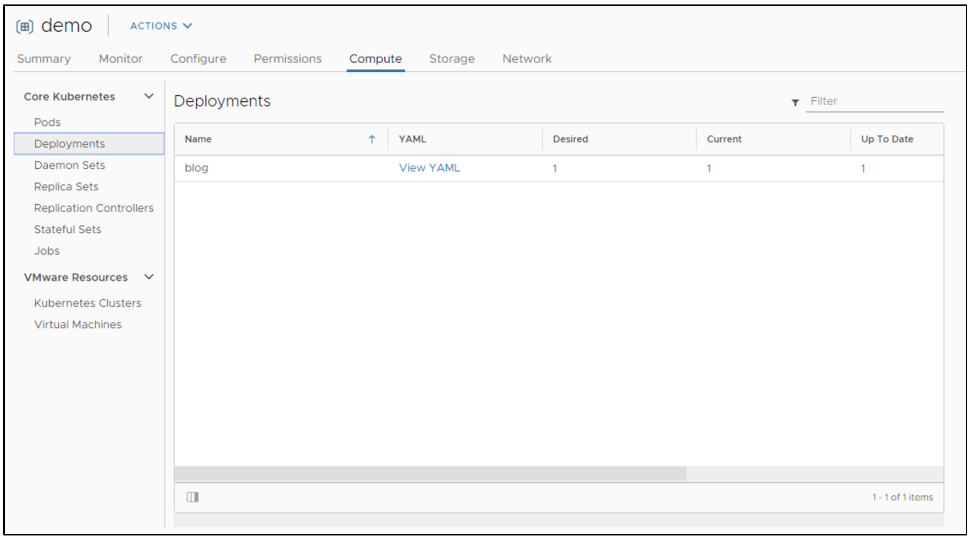
c. Click on demo Namespace in Inventory then -> `Monitor` -> `Kubernetes`.
Note Kubernetes events in the vSphere client.



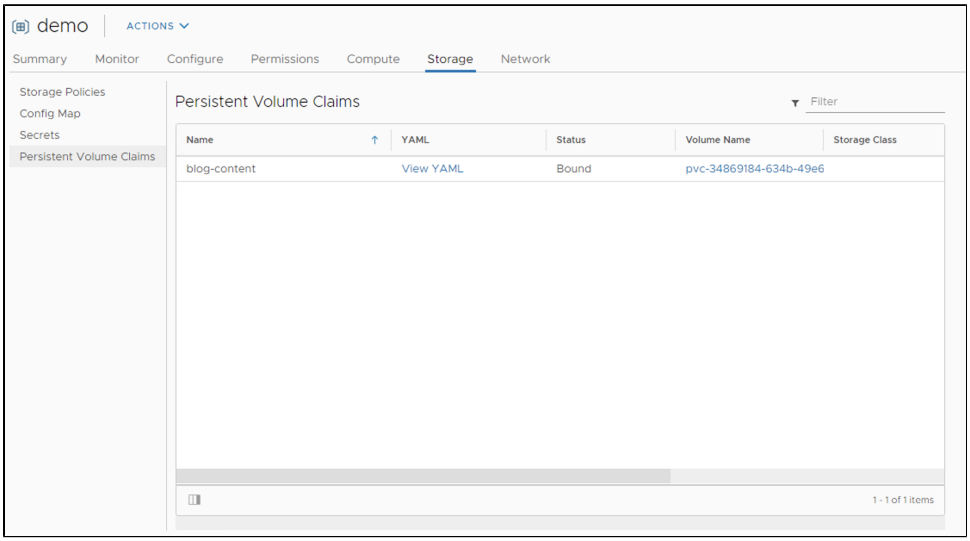
Compare that with the output of the following command:

```
kubectl get events --sort-by=.metadata.creationTimestamp
```

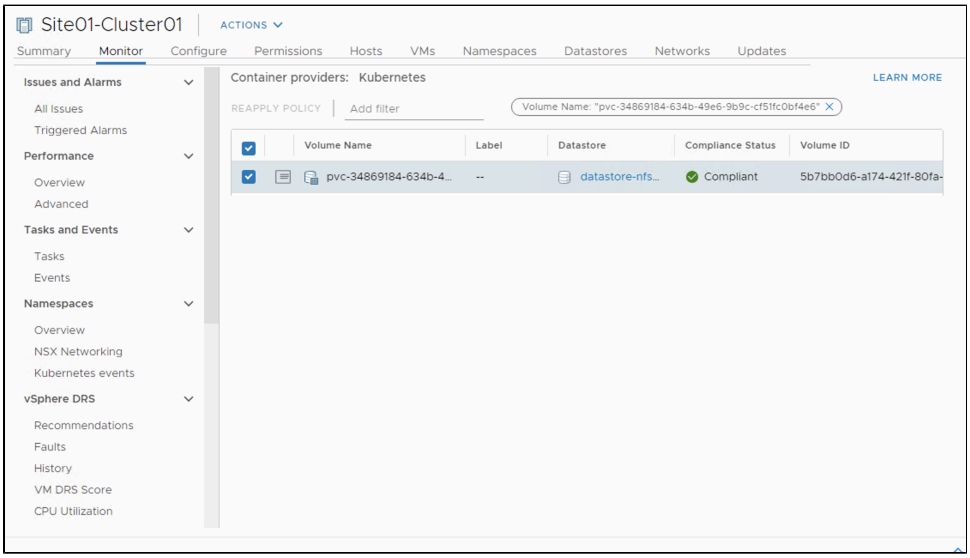
d. Browse other tabs to see the Kubernetes objects populated in the Namespace



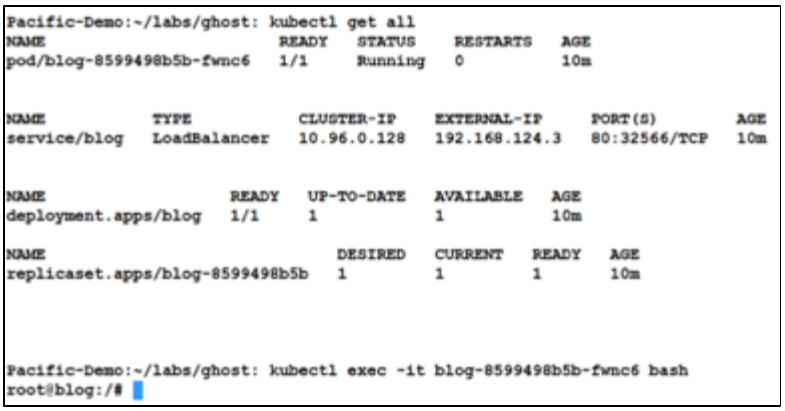
e. Click on `Storage` -> `Persistent Volumes` -> `pvc-###` Volume Name

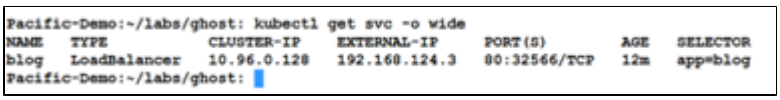


f. Click the icon next to volume name and Kubernetes objects.
Note: In Beta this view isn't opening correctly but should show PVC and Datastore details.



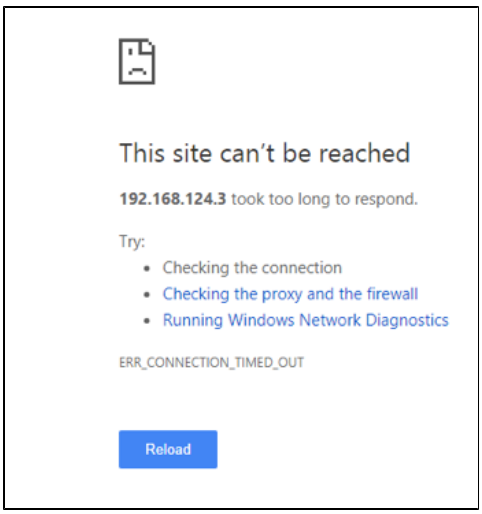
Developer: Manage Application	1. Back to the CLI. To enter the Blog Pod type the following command: `kubectl exec -it blog-#### bash` 2. Type `help` to see the Toybox Shell commands 3. Type `exit` 4. `kubectl get all` 5. `kubectl get svc -o wide` Wait for EXTERNAL-IP to be assigned. It may take a couple of minutes.





Create Network Policy to enable access to Ghost Blog App

1. From Browser go to http://<external- *IP*>.



Note that you are blocked. Namespace Ingress Deny by default,

1. Go back to CLI-VM
   a. `cat $HOME/labs/guestcluster/enable-policy.yaml`

```
Pacific-Demo:~/labs/guestcluster: cat $HOME/labs/guestcluster/enable-policy.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all
spec:
  podSelector: {}
  ingress:
  - {}
  egress:
  - {}
  policyTypes:
  - Ingress
  - Egress
Pacific-Demo:~/labs/guestcluster:
```

   b. This will create a Network Policy that enables Ingress/Egress to/from all Pods.
   c. Type `kubectl apply -f $HOME/labs/guestcluster/enable-policy.yaml`

```
Pacific-Demo:~/labs/guestcluster: kubectl apply -f $HOME/labs/guestcluster/enable-policy.yaml
networkpolicy.networking.k8s.io/allow-all created
Pacific-Demo:~/labs/guestcluster:
```

   d. `kubectl get netpol`
      `kubectl describe netpol allow-all`

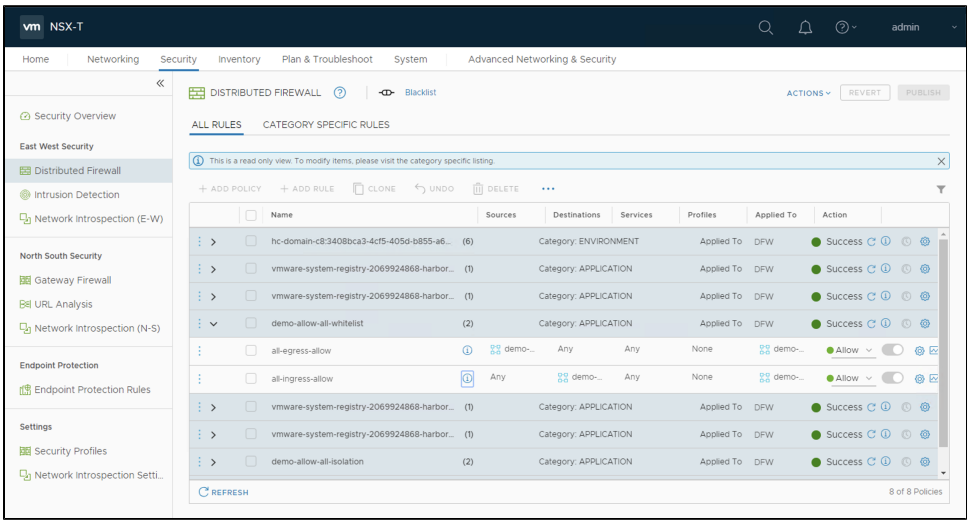
```
ubuntu@ubuntu: ~/labs/ghost
Pacific-Demo:~/labs/ghost: kubectl get netpol
NAME        POD-SELECTOR   AGE
allow-all   <none>         42s
Pacific-Demo:~/labs/ghost: kubectl describe netpol allow-all
Name:         allow-all
Namespace:    demo
Created on:   2020-01-26 13:12:04 -0500 EST
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"networking.k8s.io/v1","kind":"NetworkPolicy","metadata":{"annotations":{},"name":"allow-all","name
space":"demo"},"spec":{"e...
Spec:
  PodSelector:     <none> (Allowing the specific traffic to all pods in this namespace)
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From: <any> (traffic not restricted by source)
  Allowing egress traffic:
    To Port: <any> (traffic allowed to all ports)
    To: <any> (traffic not restricted by source)
  Policy Types: Ingress, Egress
Pacific-Demo:~/labs/ghost:
```

View Firewall Rules in NSX

Go Back to NSX-Mgr. and the Distributed Firewall.

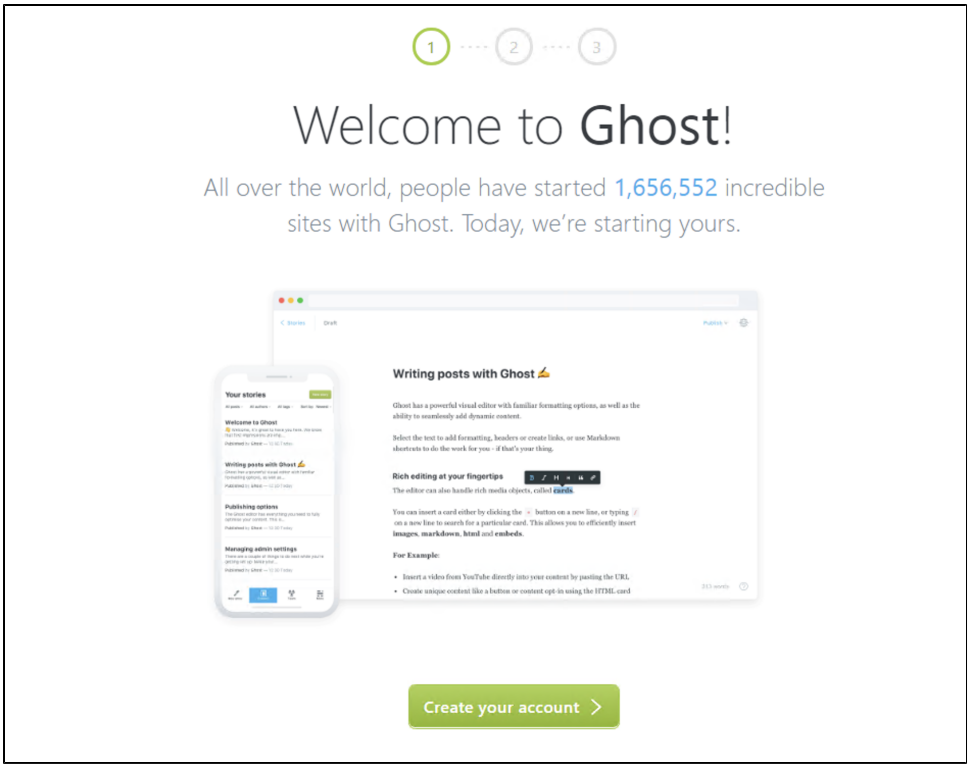
1. Refresh the Distributed Firewall page.
   Login: `admin`/`VMware1!base`
   Click `Security` -> `Distributed Firewall`

2. Expand "`demo`" `demo-allow-all-whitelist`.
   Note that `all-ingress-allow` and `all-egress-allow` has been added.



Access Ghost Blog App and create Blog

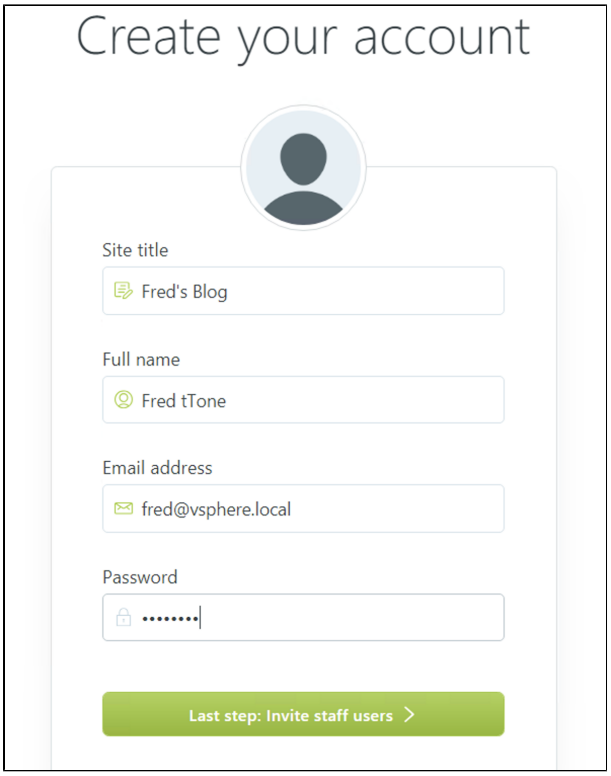
1. From Browser go to `http://"external- IP"/ghost/#/setup/one/`



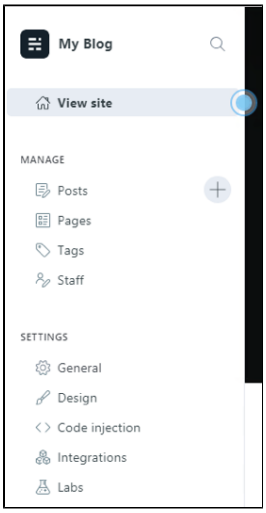
2. You now have access and can create an account and write a blog

3. Click Create an Account

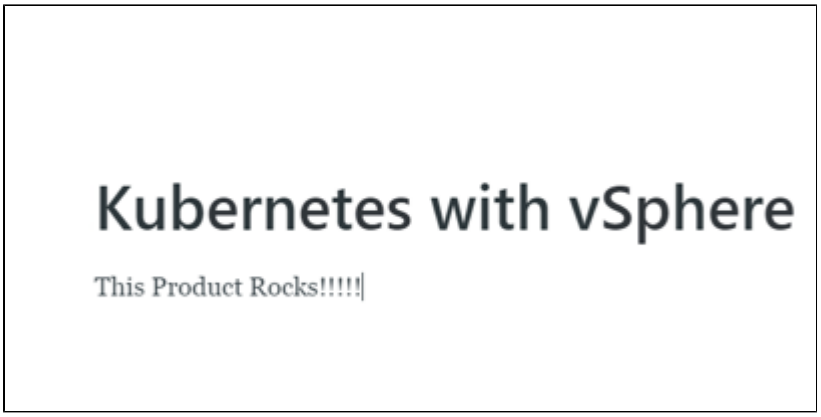
4. Add user `fred` with email of `Fred@vmware.com`



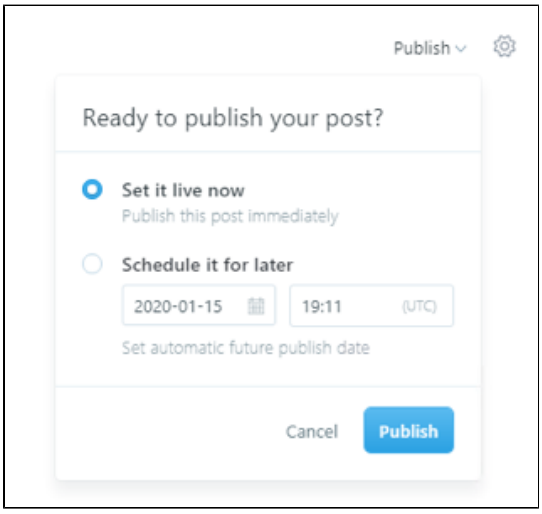
5. Do not invite coworkers to the blog
6. Click on `Manage` -> `Posts` -> `+` sign



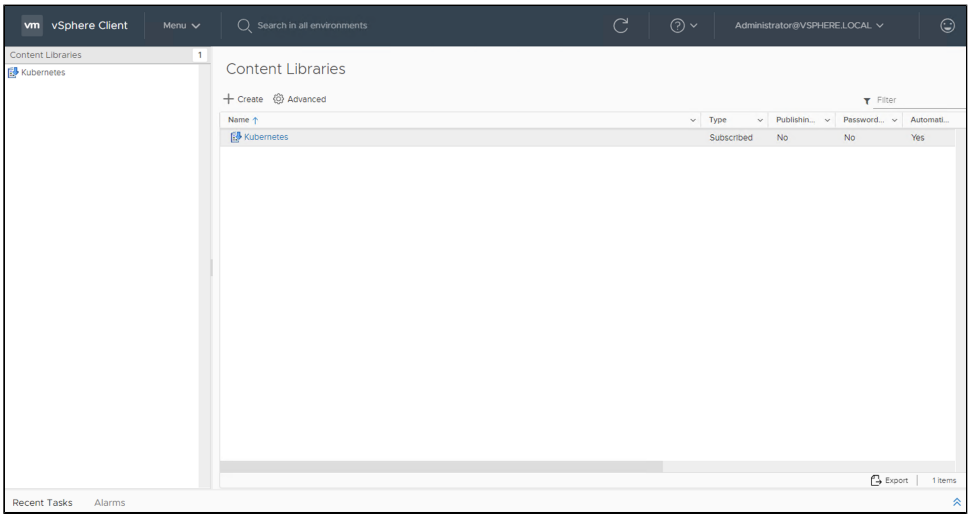
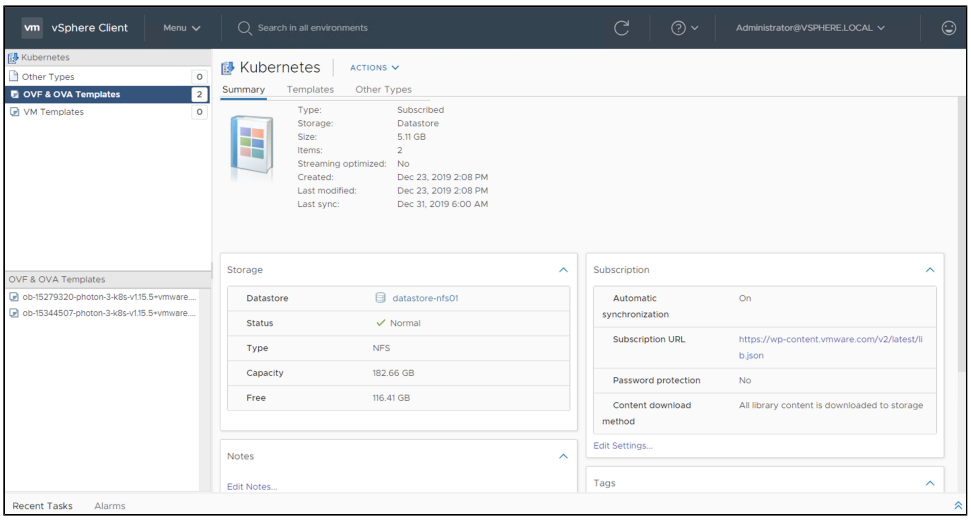
7. Now type in a Blog Title and write some text.



8. Click on Publish in Upper Right corner and select Publish.



9. You have now published your first blog post.

Update Content Library Image	We are now moving on to Tanzu Kubernetes Grid Clusters. TKG Cluster VMs will use an image stored in a content Library. The Kubernetes Content Library has already been created and subscribes to the VMware CDN. It automatically updates with the latest TKG Cluster images. From vCenter 1) Click Menu 2) Click on `Content Libraries` 3) Click on `Kubernetes` 4) And Click on Kubernetes again  4) Click on `OVF and OVA Templates`  5) Notice the templates available. If you do not see an image ending in `guest.1.37` then a. Click on `Actions` b. Click on `Synchronize` You should already have the correct version of the image. This is how we will provide upgrades to TKG Clusters
Deploy Guest Cluster	

1) Go to the CLI

```
Pacific-Demo:~: cd $HOME/labs/guestcluster
Pacific-Demo:~/labs/guestcluster:
```

2) `cd $HOME/labs/guestcluster`

3) Type: `cat guestcluster.yaml`

```
ubuntu@ubuntu: ~/labs/guestcluster
Pacific-Demo:~/labs/guestcluster: cat guestcluster.yaml
apiVersion: gcm.vmware.com/v1alpha1
kind: ManagedCluster
metadata:
  name: my-tanzu-k8-cluster
  namespace: demo
spec:
  topology:
    controlPlane:
      count: 1
      class: guaranteed-xsmall # vmclass to be used for master(s)
      storageClass: high-performance-ssd
    workers:
      count: 2
      class: guaranteed-xsmall # vmclass to be used for workers(s)
      storageClass: high-performance-ssd
  distribution:
    version: v1.15.5+vmware.1.66-guest.1.37
  settings:
    network:
      cni:
        name: calico
      services:
        cidrBlocks: ["198.51.100.0/12"]
      pods:
        cidrBlocks: ["192.0.2.0/16"]
Pacific-Demo:~/labs/guestcluster:
```

4) Notice the distribution version and the virtual machine class configuration

5) Type `kubectl apply -f guestcluster.yaml`
The cluster could take as long as 30 minutes to complete

```
Pacific-Demo:~/labs/guestcluster: kubectl apply -f guestcluster.yaml
managedcluster.gcm.vmware.com/my-tanzu-k8-cluster created
Pacific-Demo:~/labs/guestcluster:
```

6) Type `kubectl get managedcluster,cluster,machines,virtualmachines`

```
ubuntu@ubuntu: ~/labs/guestcluster
Pacific-Demo:~/labs/guestcluster: kubectl get managedcluster,cluster,machines,virtualmachines
NAME                                                CONTROL PLANE   WORKER   DISTRIBUTION                  AGE
managedcluster.gcm.vmware.com/my-tanzu-k8-cluster   1               2        v1.15.5+vmware.1.66-guest.1.37   21s


Pacific-Demo:~/labs/guestcluster:
```

These are the custom resources that are created in support of the cluster.
Describe each one individually to see what data is retained for each resource.
You will see more items created as the cluster create progresses. Keep trying this command.

7) Type `kubectl describe managedcluster my-tanzu-k8-cluster`
Look for an LB VIP at API Endpoints when cluster creation is complete.

```
Status:
  Cloudprovider:
    Name:  vmware-guest-cluster
  Cluster API Status:
    API Endpoints:
      Host:         192.168.124.4
      Port:         6443
    Error Reason:
    Phase:          provisioned
```

8) Type `kubectl describe clusters.cluster.x-k8s.io my-tanzu-k8-cluster`

```
Pacific-Demo:~/labs/guestcluster: kubectl describe clusters.cluster.x-k8s.io
Name:         my-tanzu-k8-cluster
Namespace:    demo
Labels:       <none>
Annotations:  <none>
API Version:  cluster.x-k8s.io/v1alpha2
Kind:         Cluster
Metadata:
  Creation Timestamp:  2020-01-15T22:47:57Z
  Finalizers:
    cluster.cluster.x-k8s.io
  Generation:  1
  Owner References:
    API Version:           gcm.vmware.com/v1alpha1
    Block Owner Deletion:  true
    Controller:            true
    Kind:                  ManagedCluster
    Name:                  my-tanzu-k8-cluster
    UID:                   e3c1e451-e4e6-4848-b905-e2c2dbdd3573
  Resource Version:        167454
  Self Link:               /apis/cluster.x-k8s.io/v1alpha2/namespaces/demo/cl
  UID:                     6e04b0d8-b6e4-4093-a951-34c21952e242
```

This is the Cluster API cluster resource that is created by the managedcluster controller

9) Type `kubectl get machine.cluster.x-k8s.io`

```
Pacific-Demo:~/labs/guestcluster: kubectl get machine.cluster.x-k8s.io
NAME                                PROVIDERID                                       PHASE
my-tanzu-k8-cluster-control-plane-0   vsphere://42240bf0-39bd-c300-d4f8-36596c9d4f3a   running
my-tanzu-k8-cluster-md-0-64694b75c5-krzt4   vsphere://422446a8-c56a-927d-f0b9-f639c276da60   running
my-tanzu-k8-cluster-md-0-64694b75c5-tz9rh   vsphere://4224b36e-ae5a-b1da-28b1-d254775204de   running
Pacific-Demo:~/labs/guestcluster:
```

These are the Cluster API machine resources that is created by the managedcluster controller

10) Type `kubectl describe virtualmachines`

```
Pacific-Demo:~/labs/guestcluster: kubectl get machine.cluster.x-k8s.io
NAME                                PROVIDERID                                       PHASE
my-tanzu-k8-cluster-control-plane-0   vsphere://42240bf0-39bd-c300-d4f8-36596c9d4f3a   running
my-tanzu-k8-cluster-md-0-64694b75c5-krzt4   vsphere://422446a8-c56a-927d-f0b9-f639c276da60   running
my-tanzu-k8-cluster-md-0-64694b75c5-tz9rh   vsphere://4224b36e-ae5a-b1da-28b1-d254775204de   running
Pacific-Demo:~/labs/guestcluster:
```

These are the VirtualMachine resources created by the CAPW ClusterAPI provider. The Virtualmachine Operator calls the vCenter API to create the Virtual Machines cased on the config stored in this custom resource.

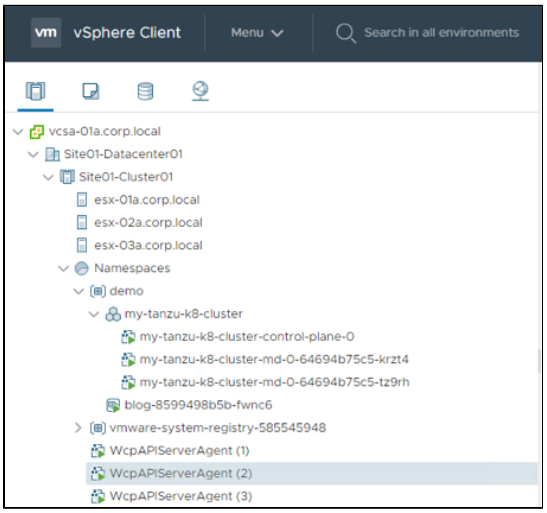
A bigger list of objects to dig through:

Cluster API v1a2 introduced different custom resources and we have internal aliases for some of them.

```
kubectl get managedclusters,clusters.cluster.x-k8s.io,wcpcluster
kubectl get wcpmachinetemplate,kubeadmconfigtemplate
kubectl get machinedeployments.cluster.x-k8s.io,machines.cluster.x-k8s.io,wcpmachine
kubectl get kubeadmconfig,virtualmachinesetresourcepolicies
kubectl get virtualmachines,virtualmachineservices
kubectl get configmaps,secrets
```

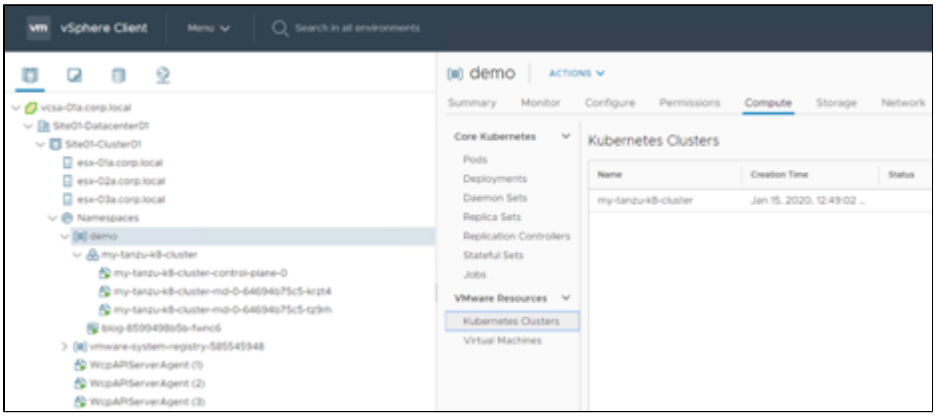
VI Admin: Manage Guest Clusters

1. Go back to vCenter
2. From the "`Hosts and Clusters`" View: Notice a new Cluster object for your k8 cluster and a set of VMs



3. It takes a few minutes to begin creating VMs

4. Click on `demo` Namespace and then "`Compute`".
   Under "`VMware Resources`": Notice Kubernetes Clusters and Virtual Machines that have been created.



Connect to Guest Cluster

1. From CLI: `kubectl vsphere login --server kubeapi.corp.local \`
   `--vsphere-username fred@vsphere.local \`
   `--managed-cluster-namespace demo \`
   `--managed-cluster-name my-tanzu-k8-cluster \`
   `--insecure-skip-tls-verify`

```
Pacific-Demo:~: kubectl vsphere login --server kubeapi.corp.local --vsphere-u

Password:
Logged in successfully.

You have access to the following contexts:
   demo
   kubeapi.corp.local
   my-tanzu-k8-cluster

If the context you wish to use is not in this list, you may need to try
logging in again later, or contact your cluster administrator.

To change context, use `kubectl config use-context <workload name>`
Pacific-Demo:~:
```

   This will generate the context from your new cluster. Notice that it is the same login you used for your supervisor Namespace but you are adding the cluster name to build that new context. You can view it in the `.kube/config` file.

2. Type `kubectl config use-context my-tanzu-k8-cluster`

```
Pacific-Demo:~: kubectl config use-context my-tanzu-k8-cluster
Switched to context "my-tanzu-k8-cluster".
Pacific-Demo:~:
```

3. Type `kubectl get all -A` to show all the objects running in all Namespaces in your K8 Cluster

```
Pacific-Demo:~: kubectl get all


NAME                 TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)    AGE
service/kubernetes   ClusterIP   198.48.0.1    <none>        443/TCP    11h
service/supervisor   ClusterIP   None          <none>        6443/TCP   11h
```

Deploy App on Guest Cluster

1. Type `cd $HOME/labs/nginx`

2. Type `wget https://github.com/mwest44/pacific/raw/master/authorize-psp-for-gc-service-accounts.yaml`

```
Pacific-Demo:~: cd $HOME/labs/nginx
Pacific-Demo:~/labs/nginx: wget https://github.com/mwest44/pacific/raw/master/auth
--2020-01-16 05:50:27--  https://github.com/mwest44/pacific/raw/master/authorize-p
Resolving github.com (github.com)... 192.30.255.112
Connecting to github.com (github.com)|192.30.255.112|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/mwest44/pacific/master/authorize-psp-f
--2020-01-16 05:50:34--  https://raw.githubusercontent.com/mwest44/pacific/master/
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.188.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.188.13
HTTP request sent, awaiting response... 200 OK
Length: 517 [text/plain]
Saving to: 'authorize-psp-for-gc-service-accounts.yaml'

authorize-psp-for-gc-service-accounts.yaml                100%[==================

2020-01-16 05:50:48 (57.3 MB/s) - 'authorize-psp-for-gc-service-accounts.yaml' save
```

Kubernetes Pod Security Policies (PSP) are enabled by default on TKG clusters.
Containers that run without any security context defined with run as root by default.
By enabling PSP, RunasRoot Pods will fail. We are creating a Deployment, which uses a service account to create the underlying pods. We will create a ClusterRole to allow privileged pods and then a ClusterRoleBinding to Bind that Role to Service accounts.
Now Pods can be created with explicitly defining the security context.

3. Type `kubectl apply -f authorize-psp-for-gc-service-accounts.yaml`

```
Pacific-Demo:~/labs/nginx: kubectl apply -f authorize-psp-for-gc-service-accounts.yaml
clusterrole.rbac.authorization.k8s.io/psp:privileged created
clusterrolebinding.rbac.authorization.k8s.io/all:psp:privileged created
Pacific-Demo:~/labs/nginx:
```

4. Type `kubectl apply -f nginx-lbsvc.yaml`

```
Pacific-Demo:~/labs/nginx: kubectl apply -f nginx-lbsvc.yaml
service/nginx created
deployment.extensions/nginx created
Pacific-Demo:~/labs/nginx:
```

5. Type `kubectl get all`

```
Pacific-Demo:~/labs/nginx: kubectl get all
NAME                         READY   STATUS    RESTARTS   AGE
pod/nginx-7bffc778db-7bhbq   1/1     Running   0          37s
pod/nginx-7bffc778db-s95g8   1/1     Running   0          36s
pod/nginx-7bffc778db-xcstt   1/1     Running   0          36s


NAME                 TYPE           CLUSTER-IP       EXTERNAL-IP     PORT(S)        AGE
service/kubernetes   ClusterIP      198.48.0.1       <none>          443/TCP        12h
service/nginx        LoadBalancer   198.51.225.125   192.168.124.6   80:31949/TCP   37s
service/supervisor   ClusterIP      None             <none>          6443/TCP       12h


NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx   3/3     3            3           37s

NAME                               DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-7bffc778db   3         3         3       37s
```

6. Open Chrome and type In LB VIP to reach nginx web page.