# vSphere 7 with Kubernetes Environment and Prerequisites

*July 14, 2020*[*2*](https://theithollow.com/2020/07/14/vsphere-7-with-kubernetes-environment-and-prerequisites/#comments)*By* [ERIC SHANKS](https://theithollow.com/author/eshanks/)

This post describes the lab environment we’ll be working with to build our vSphere 7 with Kubernetes lab and additional prerequisites that you’ll need to be aware of before starting. This is not the only topology that would work for vSphere 7 with Kubernetes, but it is a robust homelab that would mimic many production deployments except for the HA features. For example, we’ll only install one (singular) NSX Manager for the lab where in a production environment would have three.

This post will describe my vSphere home lab environment so that you can correlate what I’ve built, with your own environment.

## vSphere Environment – Cluster Layout

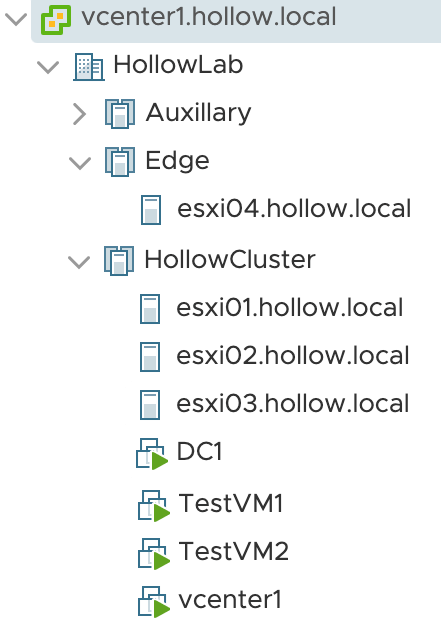
My home lab has three clusters in it. Each of which consists of ESXi hosts on version 7 and a single vCenter server running the version 7 GA build.

The lab has a single auxilary cluster that I use to run non-essential VMs that I don’t want hogging resources from my main cluster. We’ll ignore that cluster for this series. The other two cluster will be important.

I have a three node hollowlab cluster that I’ll use to host my workloads and general purpose VMs. This cluster must have vSphere HA turned on, and VMware Distributed Resource Scheduler (DRS) enabled in Fully Automated mode. This means that you’ll need to have vMotion working between nodes in your workload cluster. HA and DRS are requirements for vSphere 7 with Kubernetes clusters. More details about size/speed/capacities are found [here](https://theithollow.com/2020/02/15/2020-home-lab/).

Lastly, I have a third cluster, named edge, with a single ESXi host that will run my NSX-T Edge Virtual Machine. This VM is needed to bridge traffic between the physical network and the overlay tunnels created by NSX. You can deploy your edge VM within your workload cluster if you need to, but highly recommended to have edge nodes on their own hardware. Edge nodes can become network hotspots since overlay traffic has to flow through this VM for North/South traffic and load balancing.

Here is a look at my cluster layout for reference.



## Physical Switches

In my lab I’m running an HP switch that has some layer three capabilities. To be honest, it doesn’t matter what you’re running, but you’ll need to be able to create VLANs, and route between them. However you want to do this is fine, but if you follow along with this series, you’ll need VLANs for the purposes in the table below. I’ve listed my VLAN numbers and VLAN Interfaces for each of the networks so you can compare with your own.

NOTE: I see you judging me for my gateway addresses being .254 instead of .1. Just let it go.

|  |  |  |
| --- | --- | --- |
| **VLAN Purpose** | **VLAN #** | **Interface IP** |
| Management | 150 | 10.10.50.254/24 |
| Tunnel Endpoint (TEP) | 200 | 10.10.200.254/24 |
| NSX Edge VLAN | 201 | 10.10.201.254/24 |

v7wk8s VLANS

I have trunked (802.1q) the VLANs down to my ESXi hosts so that they can be used with my virtual switches. You will need to make sure to configure your ports so that the ESXi hosts can sent tagged packets on these vlans.

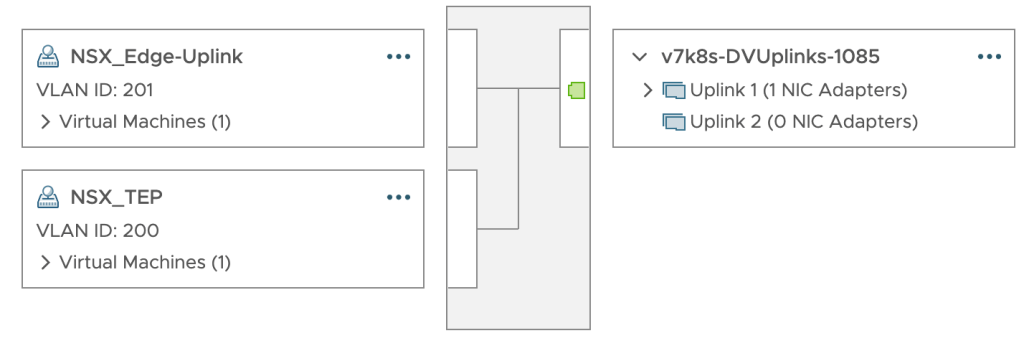
Now, for any Overlay networks, you will need to ensure that you have Jumbo Frames enabled. This means, that your Interfaces, Switches, or Distributed virtual switches must accept an frames of 1600 mtu or larger. This is an NSX requirement. Be sure to enable jumbo frames across your infrastructure.

## vSphere Virtual Switches

There is more than one way to setup the NSX-T virtual switching environment. For my lab, I’ve setup management, vMotion, vSAN, and NFS networks on some virtual switches. For those items, it doesn’t matter what kind of switches they’re on. These could be on standard switches and these are probably portgroups you’ve setup for all vSphere environments.

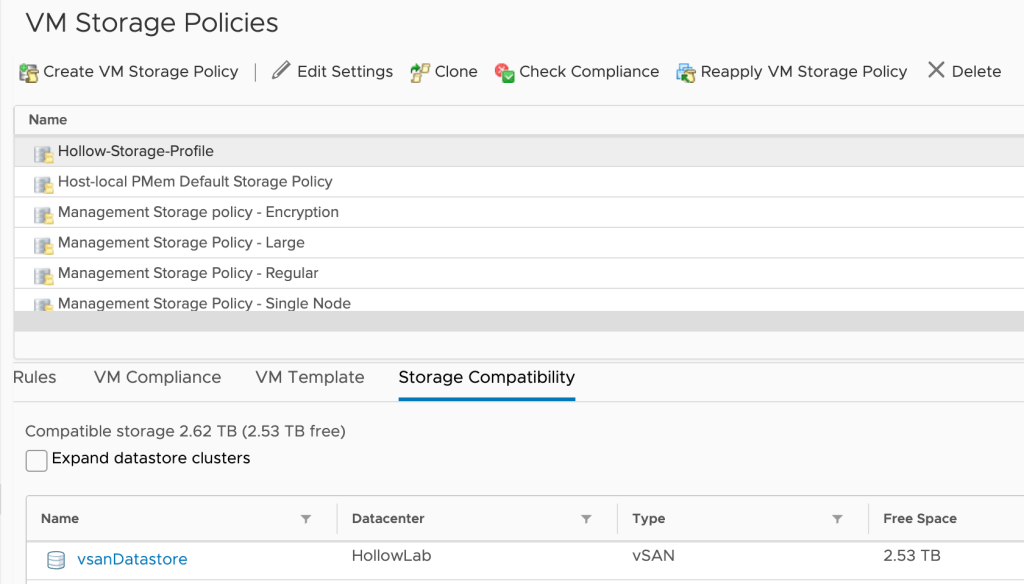
For the NSX components, I’ve deployed a single vSphere 7 distributed switch (VDS) across both my workload cluster and the edge cluster. I’ve created two portgroups on the VDS which will be used by the edge nodes deployed later in the series.

From the screenshot below, You can see I’ve created an NSX\_Edge-Uplink portgroup and an NSX\_TEP porgroup. Each of these portgroups are VLAN tagged with the VLANs shown in the table from the previous section. You can have more portgroups on this switch if you’d like. As you create new NSX-T “segments” they will appear as portgroups on this switch.



## Storage

This lab is using VMware VSAN for storage of the workload virtual machines and thereby the Supervisor Cluster VMs. You should be able to use other storage solutions, but you’ll need to have a VMware storage policy that works with your datastores. I’ve created a storage policy named Hollow-Storage-Profile that will be used for my build.



## Licenses

You will need to have some advanced licenses for some of the components. Specifically, NSX-T requires an NSX-T Data Center Advanced or higher license. Also, the ESXi hosts will need a VMware vSphere 7 Enterprise Plus with Add-on for Kubernetes license for proper configuration.

# NSX Installation

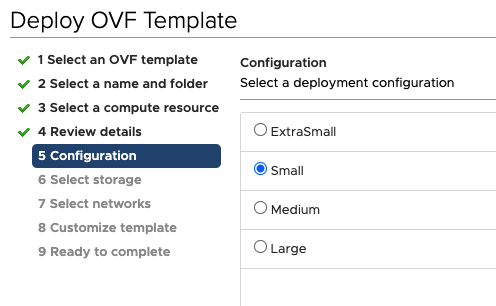
*July 14, 2020*[*2*](https://theithollow.com/2020/07/14/nsx-installation/#comments)*By* [ERIC SHANKS](https://theithollow.com/author/eshanks/)

This post will focus on getting the NSX-T Manager deployed and minimally configured in the lab. NSX-T is a pre-requisite for configuring vSphere 7 with Kubernetes as of the time of this writing.

### Deploy the NSX Manager

The first step in our build is to deploy the NSX Manager from an OVA template into our lab. The NSX Manager is the brains of the solution and what you’ll be interacting with as a user. Each time you configure a route, segment, firewall rule, etc., you’ll be communicating with the NSX Manager. Download and deploy the OVA into your vSphere lab.

As you deploy the template you’ll need to specify the size of the deployment. This is important, but for a lab environment less so. I’ve found that the Small size works well for my lab and doesn’t take up too many resources.



Fill out the rest of the deployment information. The configurations that I used are listed below, customized for my lab environment. I deployed the NSX-T Manager in the management VLAN outlined in the [previous post](https://theithollow.com/2020/07/14/vsphere-7-with-kubernetes-environment-and-prerequisites/).

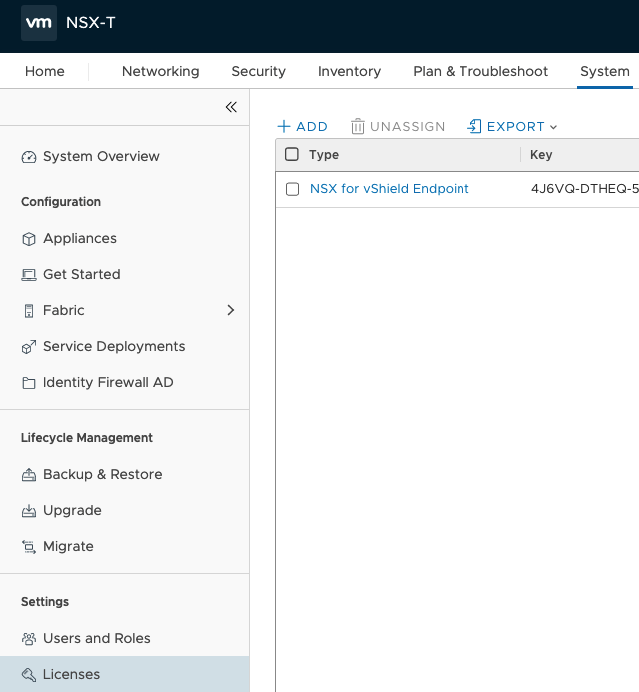
* **Hostname:** nsx
* **Rolename:** NSX Manager
* **NSX Site Name:**HollowLab
* **Default IPv4 Gateway:** 10.10.50.254
* **Management Network IPv4 Address:** 10.10.50.19
* **Management Network Netmask:** 255.255.255.0
* **DNS Server list:**10.10.50.12, 10.10.50.9
* **Domain Search List:** hollow.local
* **NTP Server List:** pool.ntp.org
* **Enable SSH:** no
* **Allow root logins:** no

Finish the installation and when complete, power on the NSX vm that was just deployed.

## Initialize NSX Manager

Once your NSX Manager appliance has been deployed and powered on, its time to do some basic initialization. The first thing you’ll do is open a web browser and navigate to the FQDN of your NSX Manager appliance you just deployed. Once you authenticate to the appliance using the credentials specified in your OVA deployment from above, you’ll probably see some pop-up screens asking you to accept a EULA, join the CEIP program, etc. Check the boxes and close any getting started windows. We don’t need that stuff.

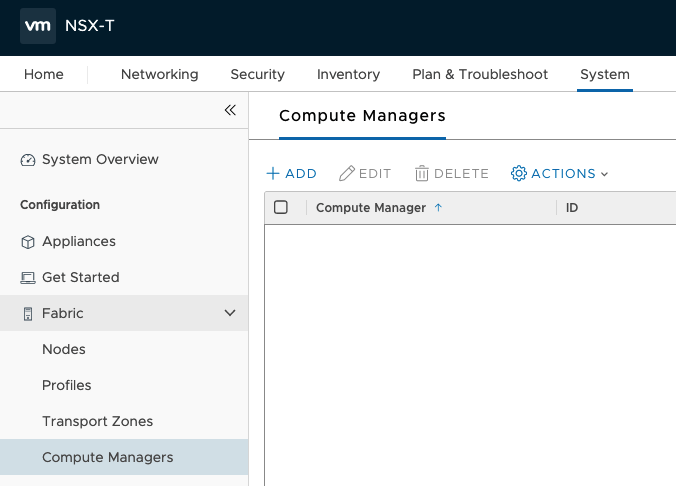
You will also need to apply a license to your NSX Manager. Navigate to the System tab and click +ADD to add a license and fill out the details. vSphere 7 with Kubernetes requires a NSX-T Data Center Advanced or higher license to be applied.



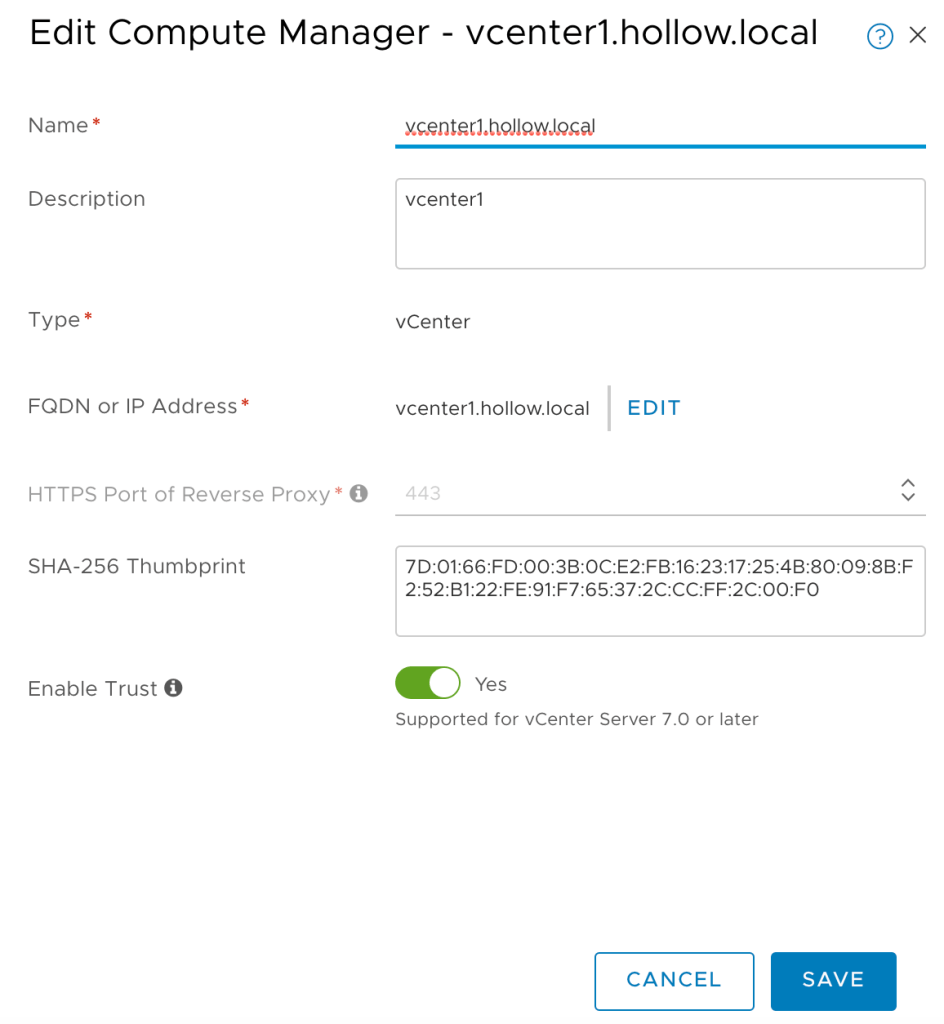
The next step in setting up our lab involves connecting a compute manager. This is a fancy name for vCenter in our case. NSX-T will use this compute manager connection to query objects and create objects as necessary. To setup the computer manager, you’ll need a service account for NSX Manager to talk to vCenter. In my case, I’m using an administrative role (Remember this is a lab), but if you want to be specific about your permissions, and of course you should, you can apply the following privileges to a service account.

|  |
| --- |
| **Extension.Register extension** |
| **Extension.Unregister extension** |
| **Extension.Update extension** |
| **Sessions.Message** |
| **Sessions.Validate session** |
| **Sessions.View and stop sessions** |
| **Host.Configuration.Maintenance** |
| **Host.Configuration.NetworkConfiguration** |
| **Host.Local Operations.Create virtual machine** |
| **Host.Local Operations.Delete virtual machine** |
| **Host.Local Operations.Reconfigure virtual machine** |
| **Tasks** |
| **Scheduled task** |
| **Global.Cancel task** |
| **Permissions.Reassign role permissions** |
| **Resource.Assign vApp to resource pool** |
| **Resource.Assign virtual machine to resource pool** |
| **Virtual Machine.Configuration** |
| **Virtual Machine.Guest Operations** |
| **Virtual Machine.Provisioning** |
| **Virtual Machine.Inventory** |
| **Network.Assign network** |
| **vApp** |

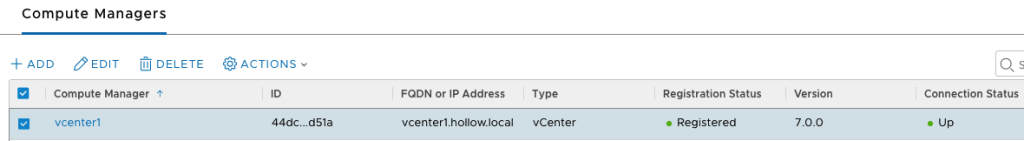
From within the NSX Manager console, go to System –> Fabric –> Compute Managers and click +ADD.



In the next screen, enter your vCenter information and login credentials. Then click the Add button. When you do this for the first time you’ll be presented with a SHA-256 thumbprint and you’ll need to accept that its valid before continuing. Lastly, click the “Enable Trust” button so that it’s in the Yes position. This last step is important as it allows NSX to trust vCenter for authentication.



You will be asked to add a thumbprint. Click Add. When you’re done you’ll have a vCenter configured and registered as a compute manager.



## Summary

In this post we deployed the NSX Manager which is the brains of the NSX-T product and have configured licenses and connected it to our vCenter server. In the [next post](https://theithollow.com/2020/07/14/nsx-pools-zones-and-nodes-setup/) we’ll start configuring NSX-T so that we can start routing some traffic to some virtual machines.

# NSX Pools, Zones, and Nodes Setup

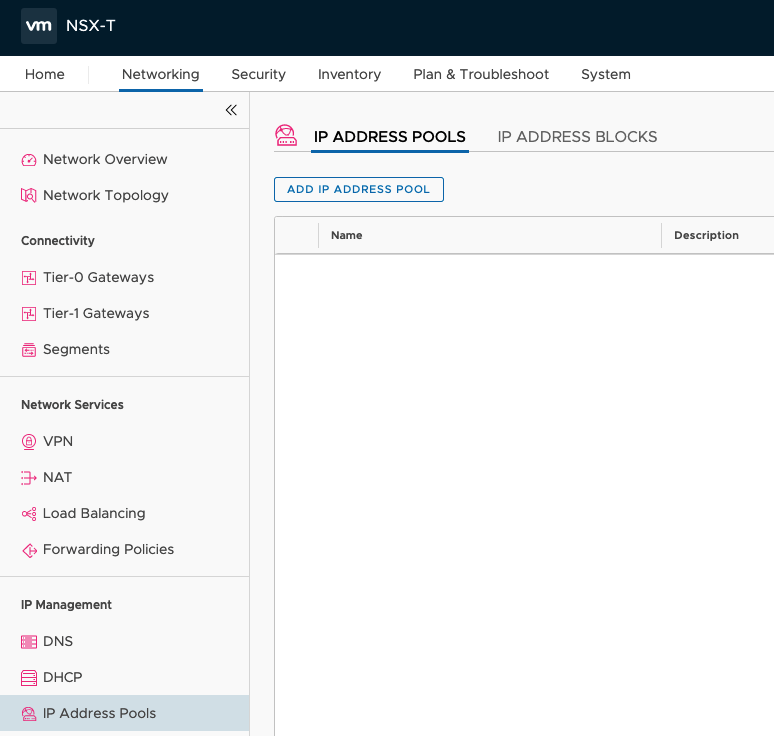
*July 14, 2020*[*2*](https://theithollow.com/2020/07/14/nsx-pools-zones-and-nodes-setup/#comments)*By* [ERIC SHANKS](https://theithollow.com/author/eshanks/)

In the [previous post](https://theithollow.com/2020/07/14/nsx-pools-zones-and-nodes-setup/) we deployed an NSX Manager. Now it’s time to start configuring NSX so that we can build cool routes, firewall zones, segments, and all the other NSX goodies. And even if we don’t want to build some of these things, we’ll need this setup for vSphere 7 with Kubernetes.

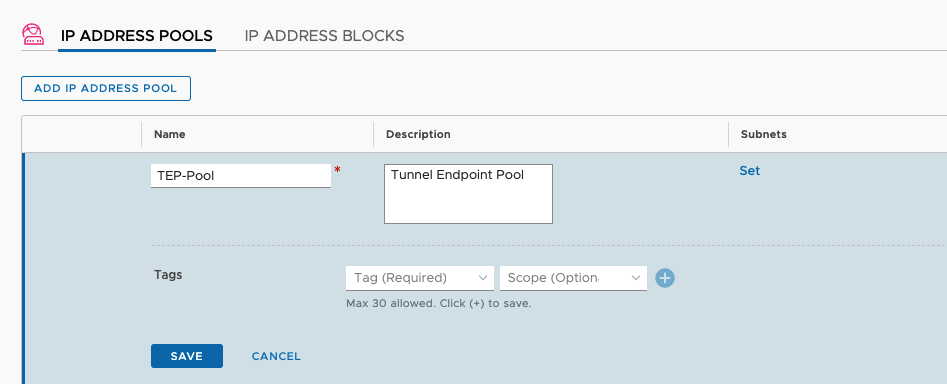
## Add an IP Pool

The first thing we’ll setup is an IP Pool. As you might guess, an IP Pool is just a group of IP Addresses that we can use for things. Specifically, we’ll use these IP Addresses to assign Tunnel Endpoints (Called TEPs previously called VTEPs in NSX-V parlance) to each of our ESXi hosts that are participating in the NSX Overlay networks. The TEP becomes the point in which encapsulation and decapsulation takes place on each of the ESXi hosts. Think of it this way, when encapsulated traffic needs to be routed to a VM on a host, what IP Address do we need to send the traffic to, so that it can reach that VM. This is the TEP. We need to setup a TEP on each host, and the IP Addresses for these TEPs come from an IP Pool. Since I have three hosts, and expect to deploy 1 edge nodes, I’ll need a TEP Pool with at least 4 IP Addresses. Size your environment appropriately.

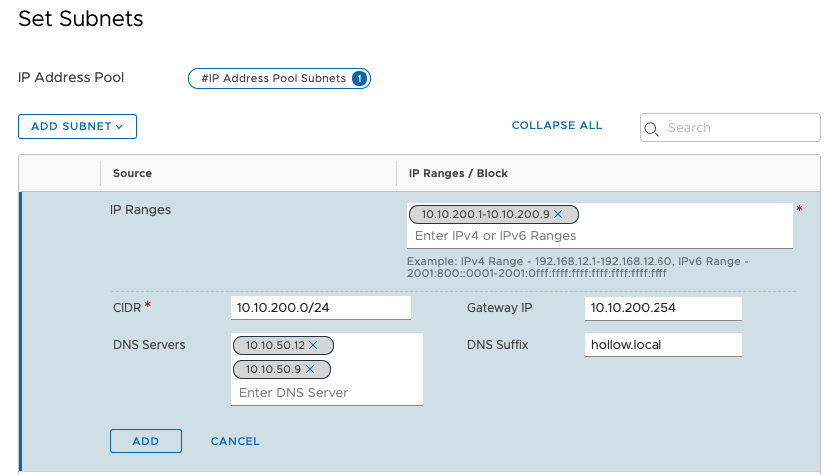
These IP Pools can be setup as DHCP or Static. Since this is a small lab we’ll walk through using static IP Addresses. To begin, navigate to the Networking tab and click IP Address Pools in the NSX Manager portal.



Click the ADD IP ADDRESS POOL button and then give the new pool a name and a description. Then click the Set hyperlink under subnets.



On the subnets setup screen you can add your pool IPs. For this I’ll use a range of IP Addresses in my 10.10.200.0/24 network which is my VLAN 200 NSX\_TEP network. Add your range and click Apply.



## Create a Transport Zone

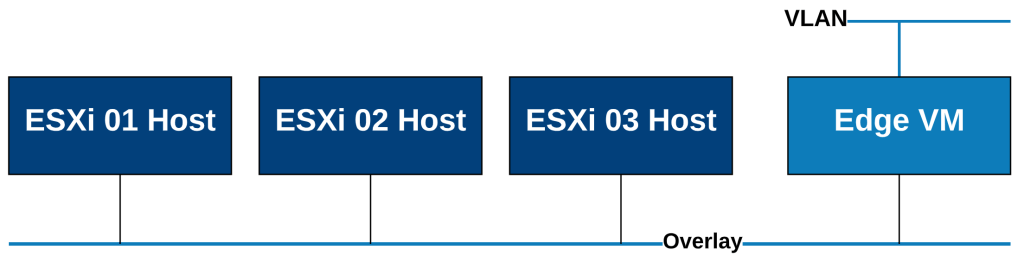
Now it’s time to setup a Transport Zone. A Transport zone is a network that will, you guessed it, transport packets between nodes. And guess where those packets will land? Yep, on the Tunnel Endpoints. The way I think about a Transport zone is that it’s a grouping of hosts that are participating in NSX networks.

We have two types of Transport Zones. VLAN and Overlay.

**Overlay –** These are the networks created by NSX and carry the encapsulated geneve tunnels. When you create a new NSX-T segment, the encapsulated packets are passed via this overlay transport zone.

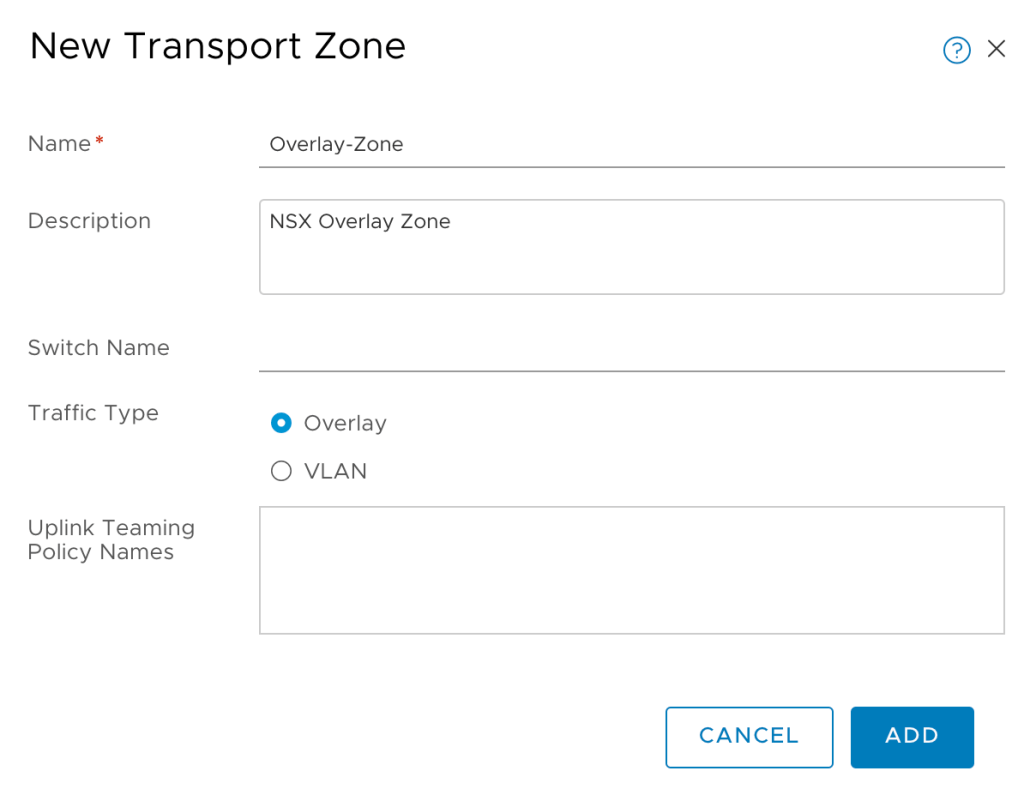
**VLAN –** These are networks backed by a VLAN used for communicating North/South with the physical network. These are commonly deployed on edge nodes so that the Overlay networks can route out to the physical network via an edge.

To give a graphical example of what we’re doing, see below.

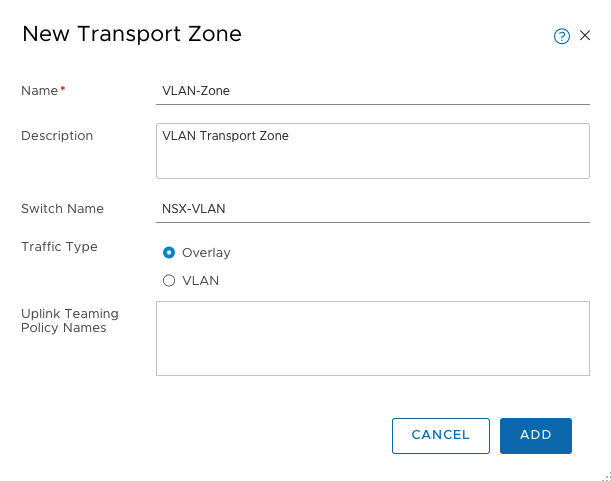


To create our Transport Zones go to System –> Fabric –> Transport Zones in the NSX Manager UI. Click the + button to add a new transport zone. NOTE: there may be default zones created already. I’m ignoring those in my setup and creating my own.

First I’ll create the Overlay Zone. Give it a name, description and a switch name. Then select the type of zone, in my case Overlay. Click Add.



Create another zone, but this time make it a VLAN zone.



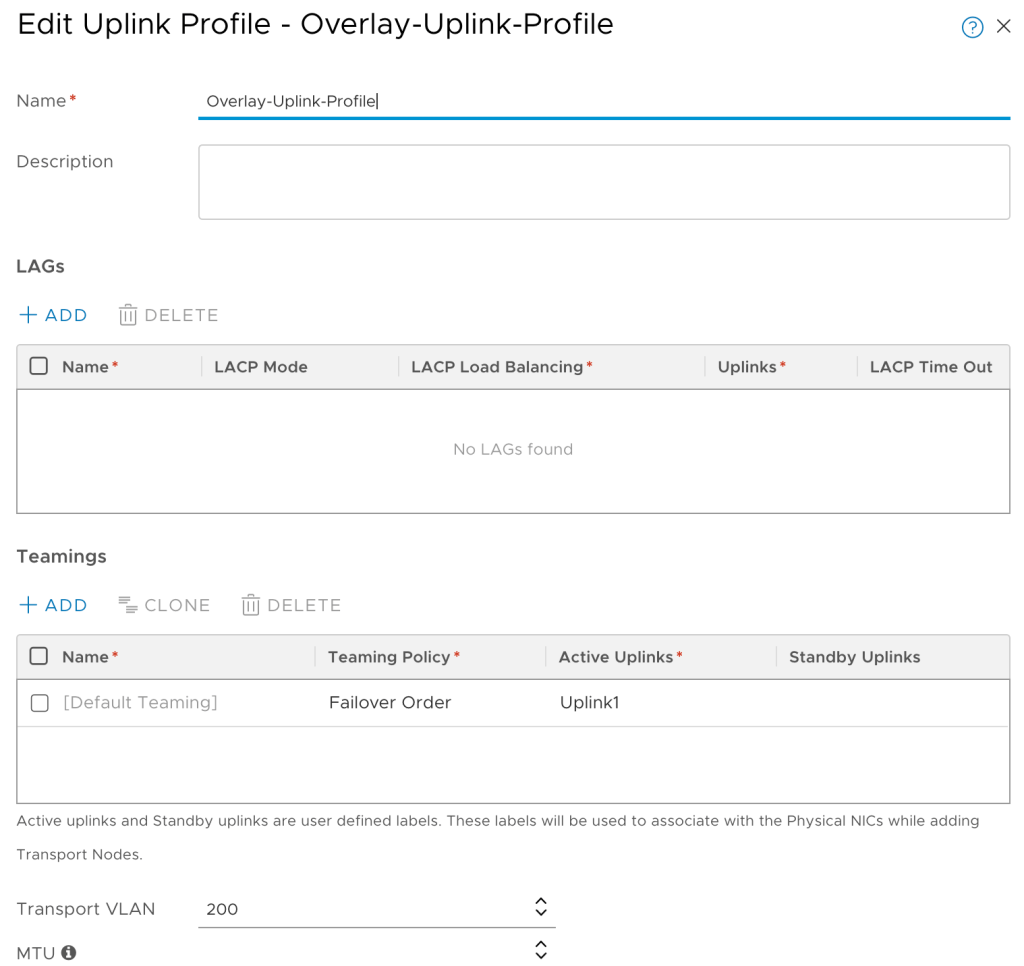
At this point we have enough transport zones to continue. Lets build some Transport Node profiles.

## Uplink Profiles

Uplink Profiles give you a way to set your teaming policies, and uplinks for any of the transport nodes you’ll be creating. Since this is a lab, the default uplink profiles might not be the best fit. I’m using a single NIC which you should not do for a production environment so I’ll create a custom uplink profile.

In the NSX Manager go to System –> Fabric –> Profiles and then click the + under the Uplink Profiles page.

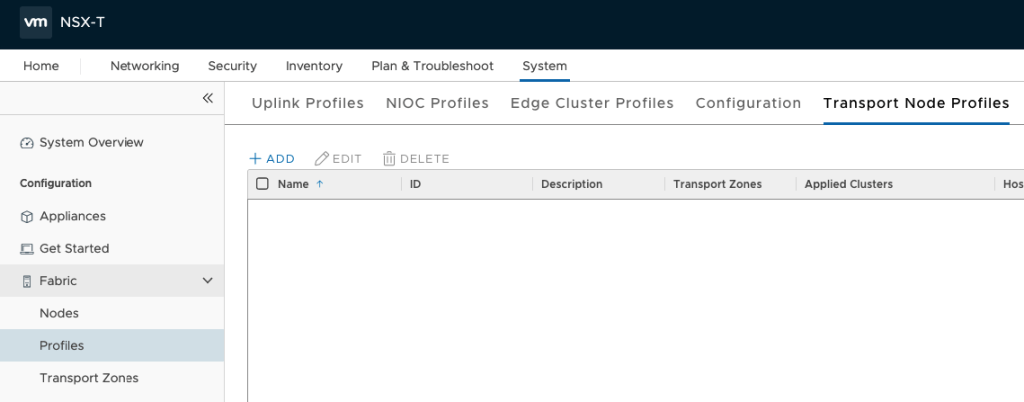
You can see from my screenshot below, that I’ve given it a name and added a single nic to the active uplinks. That nic is named vmnic1 which is the vmnic on my distributed switch. My overlay network is on VLAN 200 so the Transport VLAN field needs to be set to 200. Save your configuration.



## Transport Node Profile

The transport node profile is used to provide configuration for each of the ESXi nodes. The profile specifies which NICs on the nodes need to be configured for the VDS switch. It also specifies the IP Addresses assigned for the TEP on this switch.

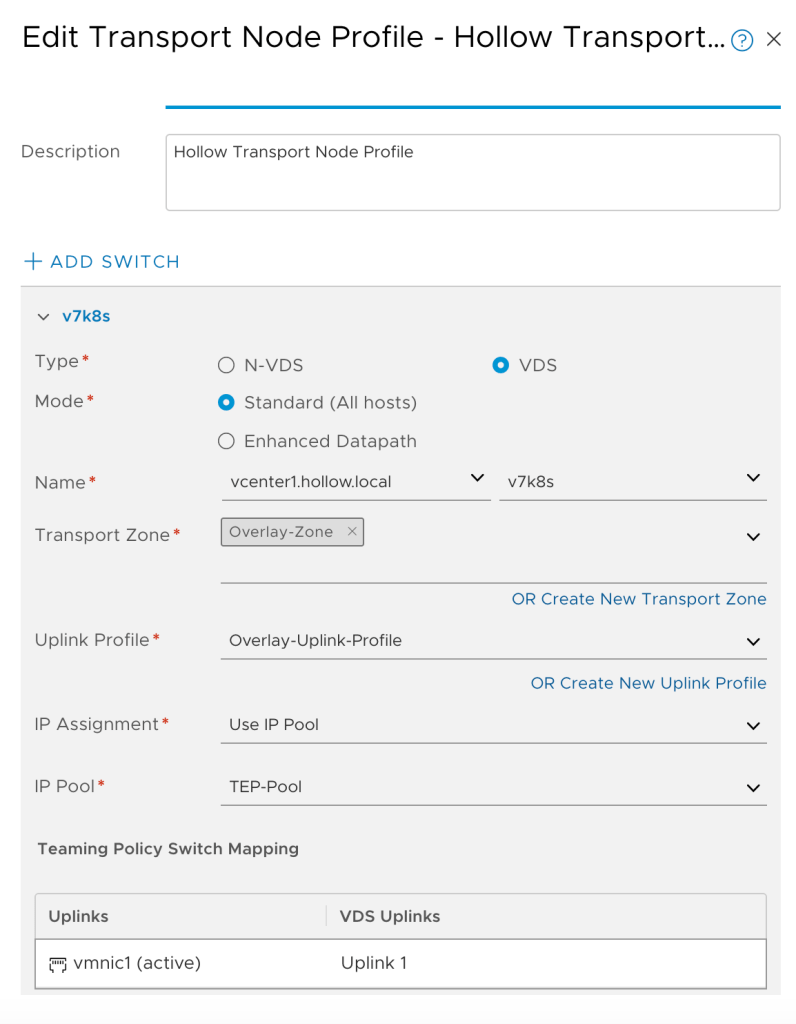
Navigate to the System –> Fabric –> Profiles page. Then click +ADD.



Give the profile a name, and description. Then click VDS as the switch type. You’ll then select your computer manager and the name of the VDS. Then select an uplink profile from the list of defaults. I’ve chosen the Overlay profile from above. Under IP Assignment select Use IP Pool and then under IP Pool, select the TEP Pool we created earlier.

Lastly, in the uplinks, you must specify an ESXi Physical NIC that the VDS switch will use as a physical uplink. In my lab I have Uplink1 for my uplink NIC on the VDS.

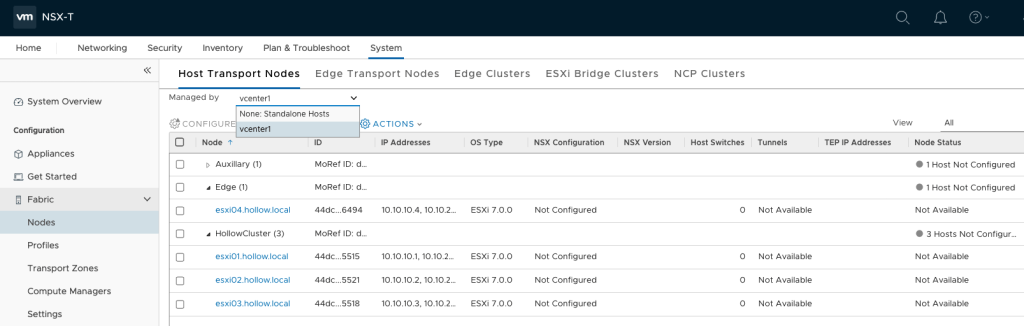
>NOTE: Each ESXi host could be configured differently if they are non-uniform. You would need to configure each node individually instead of as a full cluster. Transport Node Profiles make this a snap as long as you’ve got uniformed infrastructure (meaning the same vmnic is used on each ESXi host in your Transport Zone).



We’re prepared to configure our nodes now. Lets push the configurations down to the nodes to prep them for use.

## Configure Transport Nodes

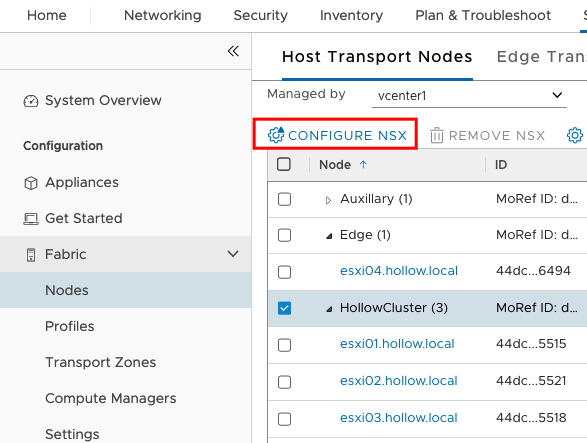
Now that we’ve got a profile, we’ll go to System –> Fabric –> Nodes. Under the Managed by drop down, select your Compute Resource (vCenter). Click +ADD.



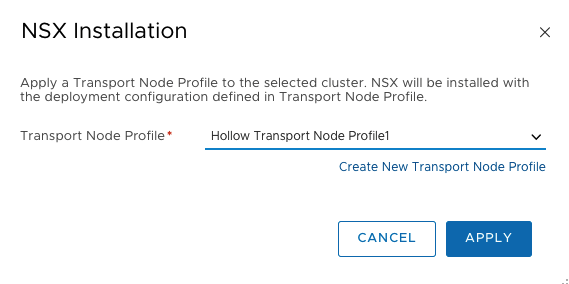
You should see your list of clusters. I’ve expanded HollowCluster cluster and you can see the nodes are not configured. Select the cluster that will be configured with your transport node profile created above.

NOTE: You do not need to configure the edge ESXi host, just the workload nodes.

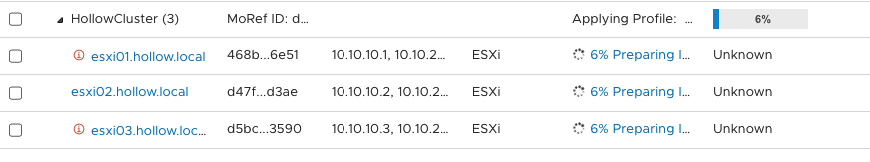
Click the CONFIGURE NSX button.



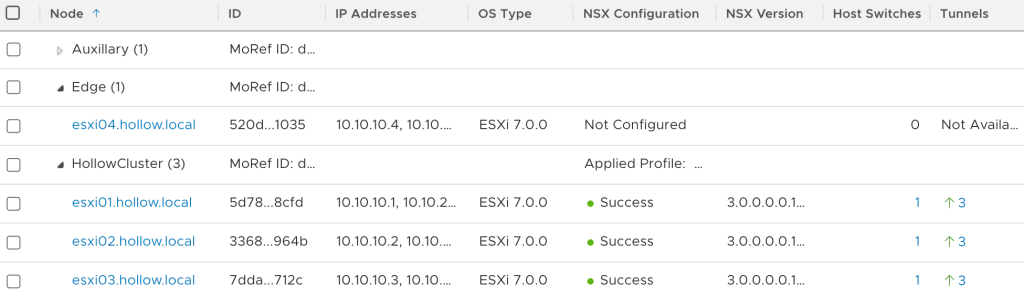
Select the Transport Node Profile we created earlier and then click APPLY.



NSX will soon be configuring things on your ESXi hosts.



When complete you should see success listed next to each of the nodes in the cluster.



## Summary

In this post we configured our nodes with our Transport zone and configured our profiles to configure the virtual switches and NICs on the ESXi nodes. In the [next post](https://theithollow.com/2020/07/14/deploy-nsx-t-edge-nodes/) we’ll move to our Edge Nodes.

# Deploy NSX-T Edge Nodes

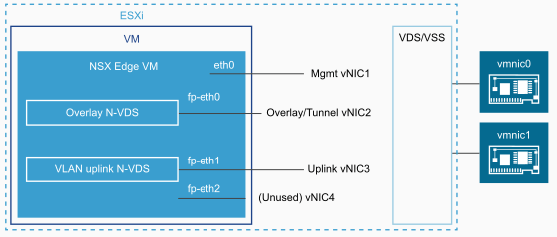
*July 14, 2020*[*7*](https://theithollow.com/2020/07/14/deploy-nsx-t-edge-nodes/#comments)*By* [ERIC SHANKS](https://theithollow.com/author/eshanks/)

NSX-T Edge nodes are used for security and gateway services that can’t be run on the distributed routers in use by NSX-T. These edge nodes do things like North/South routing, load balancing, DHCP, VPN, NAT, etc. If you want to use Tier0 or Tier1 routers, you will need to have at least 1 edge node deployed. These edge nodes provide a place to run services like the Tier0 routes. When you first deploy an edge, its like an empty shell of a VM until these services are needed.

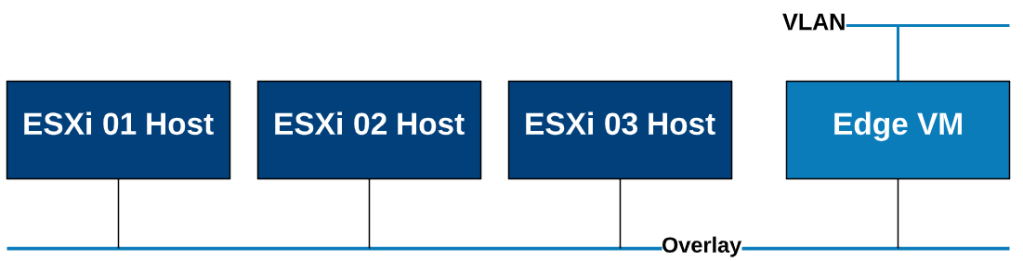
In my lab, I’m deploying the edge nodes to their own cluster. This is not a requirement for a lab, but a good recommendation for a production setup since traffic is usually funneled through these instances and they can become a network hot spot.

Before we do the deploy, let’s revisit this logical diagram for the edge node we’ll be deploying. I’ll be honest, this edge networking caused me fits until I realized that we need to extend the overlay networks from the ESXi hosts in our workload cluster, to the Edge Virtual Machine. You must add the Edge VM to the TEP network to participate in the overlay. Then, you will have a second VM interface that connects to a VLAN transport zone which will be the portgroup created on my Edge ESXi host virtual switch.

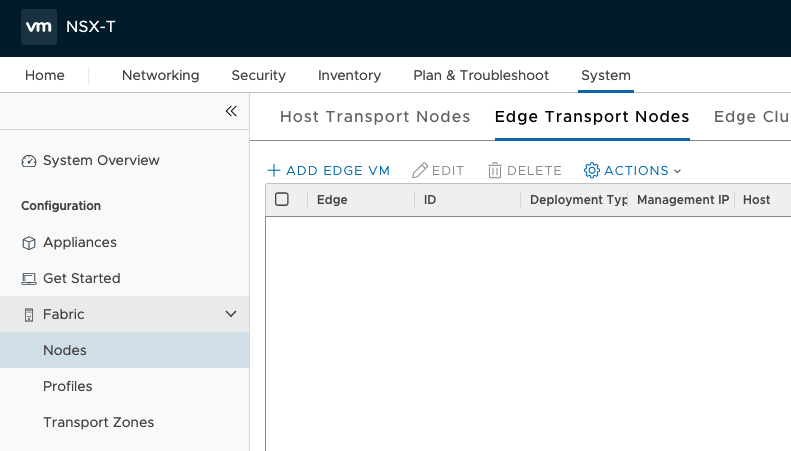
To wrap your head around the Edge VM networking, take a look at this page, and specifically the diagram found below. The Edge VM has a virtual switch inside it, and we’ll connect the edge vm uplinks to the Distributed virtual switch uplinks. The Edge VM will have three or more interfaces. 1 for Management, 1 for Overlay, and 1 for VLAN traffic to the physical network.



In the end, our overlay network will extend from the ESXi hosts to the Edge virtual machine. The edge virtual machine will have a path to the physical network.

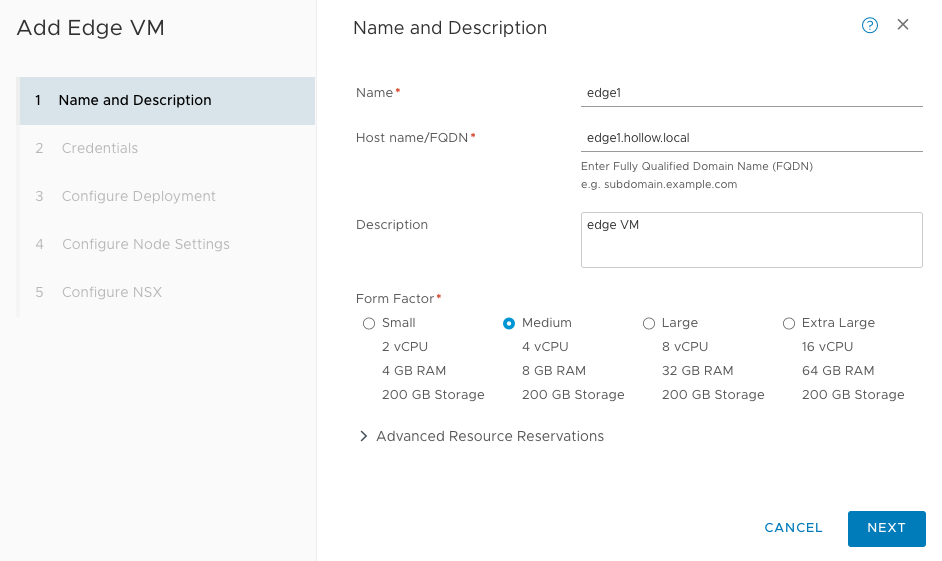


To deploy the first edge node, go to the NSX Manager under System –> Fabric –> Nodes -> Edge Transport Nodes. Click the + ADD EDGE VM button.

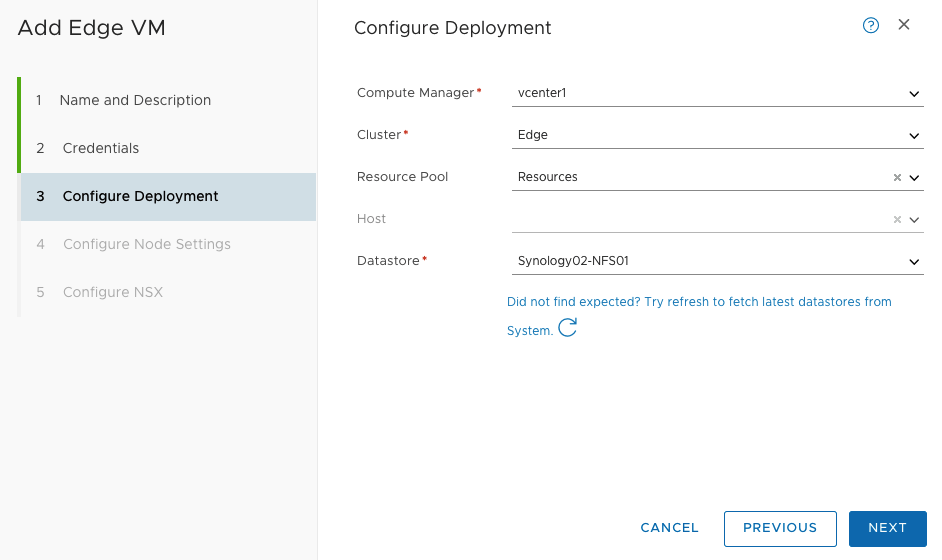


Give the edge vm a name, FQDN, and description before selecting a size. Sizing is critically important for a production environment. It has impacts on the number of load balancers that can be provisioned among other things.

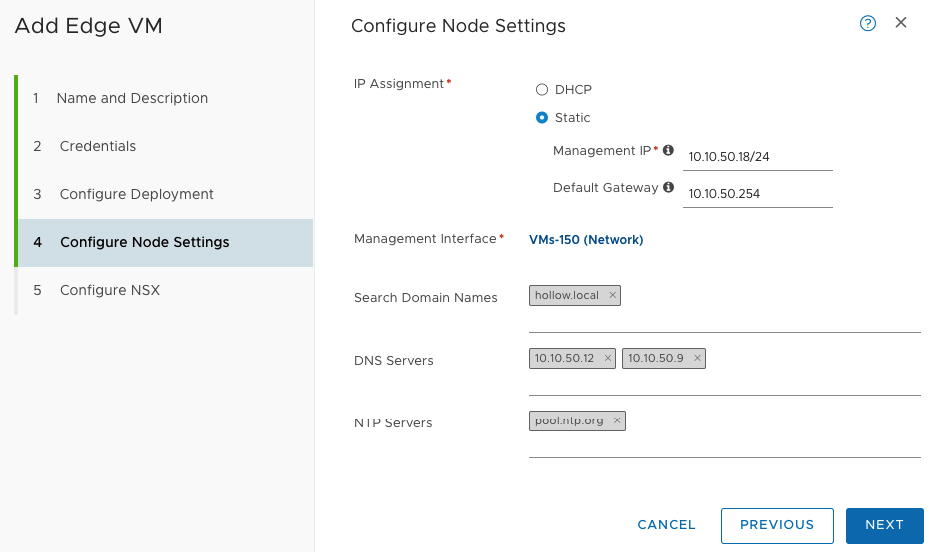
NOTE: The [documentation](https://docs.vmware.com/en/VMware-vSphere/7.0/vmware-vsphere-with-kubernetes/GUID-B1388E77-2EEC-41E2-8681-5AE549D50C77.html) states that you need the Large form factor for the Tanzu components. I was able to get it to come up with a Medium form factor, but could not deploy TKG clusters until I upgraded to Large.



Next, select the compute manager, cluster, resource pool, and datastore for the edge node to be deployed on.



In the configure node settings box, give the VM an IP address on a management network. This is NOT in the data plane, but rather a way to communicate with the edge node. I’ve placed this VM on one of my existing management portgroups on my edge cluster. (VLAN 50 – Management)

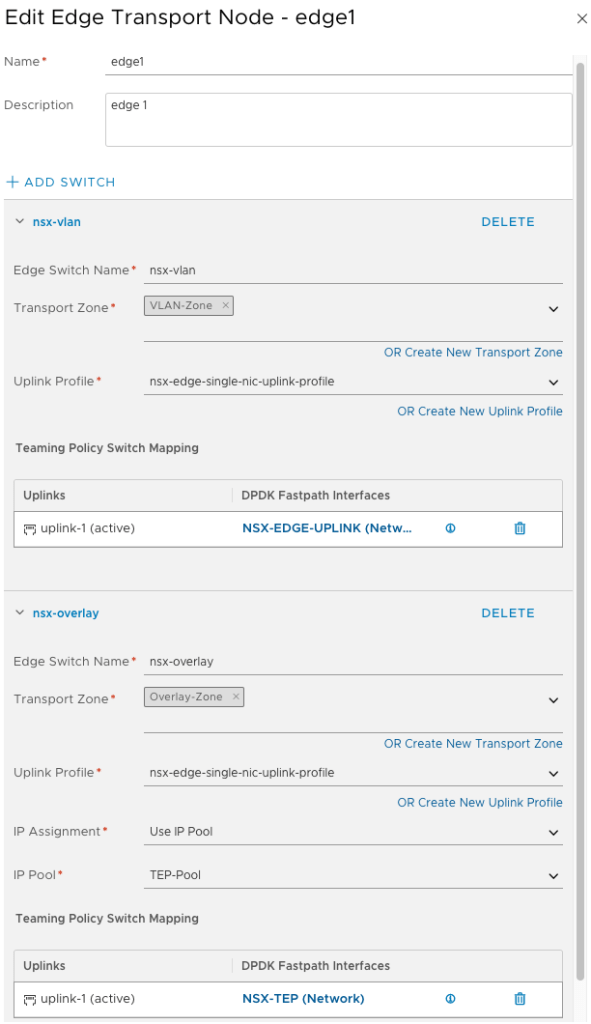


The last screen is where the configuration really happens. This is where we create virtual switches in the Edge VM and connect them (through uplinks) to a physical Nic. We need to create two switches in this screen, these switches will not be visible in the ESXi hosts view, because they exist within the Edge virtual machine.

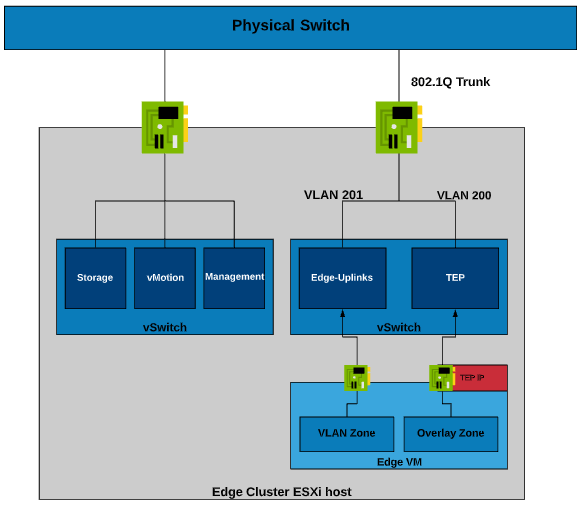
We need two switches created. One for the Overlay network which belongs on the TEP network along with our ESXi hosts in our workload cluster. And another for the VLAN backed network which is how the VM communicates on the physical network.

On the Configure NSX screen, click the +ADD SWITCH link twice so we can setup each switch. The first I’ve named nsx-vlan to represent the northbound physical network. I selected the VLAN-Zone transport zone (VLAN 201 – Edge Network) and the single nic profile which is an out-of-the-box profile. Under the teaming policy, I’ve selected my Edge Uplink portgroup that was already created on my ESXi DVS. This link IS within the data path.

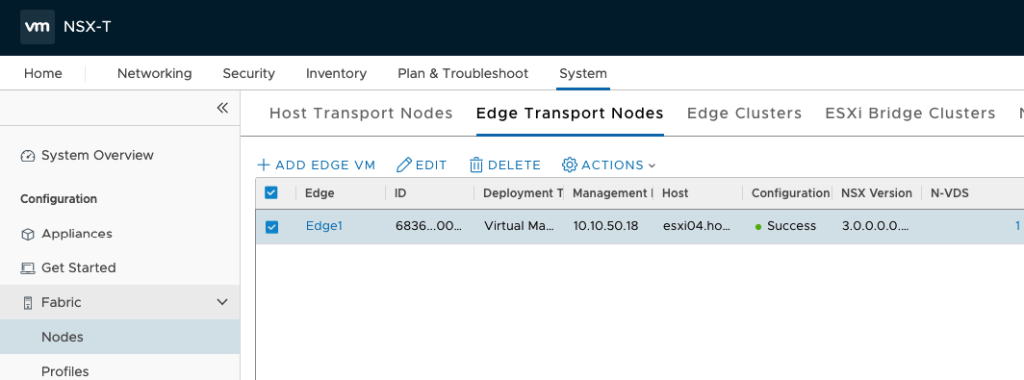
On the second switch in the configuration I’ve added the Overlay-Zone transport zone, again with the single nic profile. Under Address assignment, select Use IP Pool and select the TEP pool that we also used on our workload ESXi hosts to add them to the overlay. Then select the uplink for the NSX-TEP network (VLAN 200 – NSX TEP).



I know this piece was confusing to me, so if you’re stuck, take a look at the diagram below. The edge VM will be created with a pair of switches. The uplinks on those switches will be portgroups on the DVS. My lab layout is shown below for the edge VM.



After finishing the configuration, an edge VM should be listed under your Edge Transport Nodes.

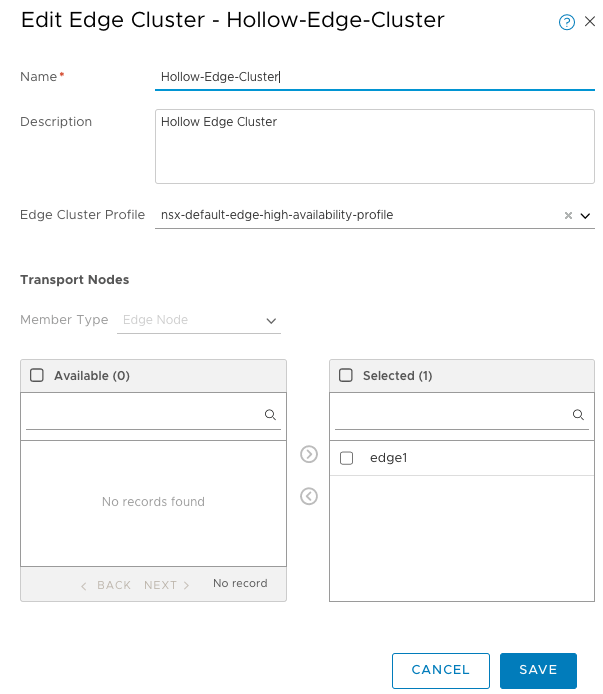


## Create Edge Cluster

Edge Nodes can (and in production environments should) be deployed in pairs. Groups of Edge nodes can then be pooled together into an Edge cluster. Thus far, we haven’t focused enough on High Availability because its a lab short of resources, but these routines should be modified slightly to provide this HA capability. This might include adding a second VLAN transport zone for a second physical switch etc. Edge clusters, although we won’t be using more than one in our example, are required.

Add your new Edge node to an edge cluster by navigating to System –> Fabric –> Nodes –> Edge Transport Zones. Click the + ADD button to create a new cluster.

Give the cluster a name and description. Then make sure your edge VM has been selected and moved to the right column.



## Summary

I found that understanding the edge node routing was the most difficult piece to setting up NSX in the lab. Remember that we’re extending the Overlay transport zone from the workload ESXi host cluster to the Edge node VM. The Edge VM then has a second VLAN transport zone where traffic can be routed to the physical network. Stay tuned for the next post were we create some actual overlay networks that our VMs can use.

# Tier-1 Gateway and NSX Segments

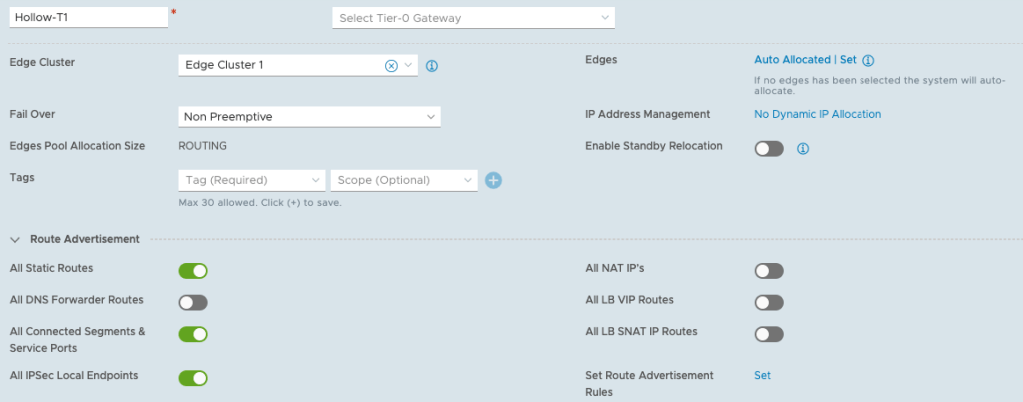
*July 14, 2020*[*1*](https://theithollow.com/2020/07/14/tier-1-gateway-and-nsx-segments/#comments)*By* [ERIC SHANKS](https://theithollow.com/author/eshanks/)

This post will focus on deploying our first NSX Gateway/Router and setting up our overlay segments. Before you can start these steps, the Edge nodes should be up and running so that they can support the Tier-1 gateways.

NSX uses two types of routers/gateways. We’ll start by using a Tier-1 (T1) router. These routers are usually used to pass traffic between NSX overlay segments. We could create NSX segments without any routers, but it would require a router to pass traffic between these segments so we will create a T1 router first.

## Tier-1 Deployment

To setup your first Tier-1 router go to the Networking –> Tier-1 Gateways page and click ADD TIER-1 GATEWAY BUTTON. Give the router a name and select the edge cluster from the drop-down. Under route advertisement, enable All Static Routes, and All Connected Segments and Service Ports. For now, this is really all that needs to be done. We’ll revisit this at a later time.



**Create NSX Overlay Segments**

Segments are layer 2 broadcast domains where we can run our virtual machines. When you create an NSX segment, a portgroup will be created on our VDS virtual switch and then be available for use within the vCenter environment for workloads.

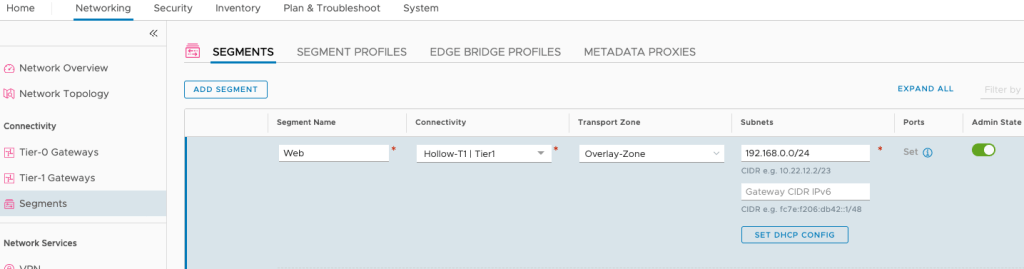
NOTE: creating the segments won’t immediately create portgroups in your vCenter. If nothing is attached to the segment (like a VM) then the portgroup won’t show up.

We’ll create three segments for our workloads.

NOTE: You don’t need VLANs created on the physical network for these networks. These are on the overlay networks created and managed by NSX.

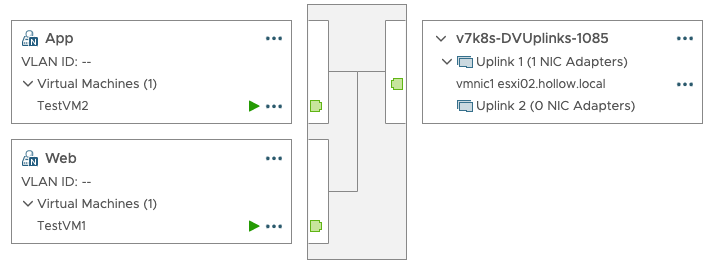
* Web – 192.168.0.1/24
* App – 192.168.1.1/24
* Database – 192.168.2.1/24

Go to Networking –> Segments and then click the ADD SEGMENT button to create a new segment. Fill out the name, and select the T1 router created earlier. Then select the Overlay dropdown and enter the Subnet CIDR for this segment. Then click Save.



Thats all there is to creating a new segment. Continue filling out the segments for each of the networks you’d like to create.

After deploying these segments, I connected a virtual machine to Web and App segments to test connectivity between them. The result was a successful ping between VMs on different segments and in vCenter, you can see the portgroups for those new segments.

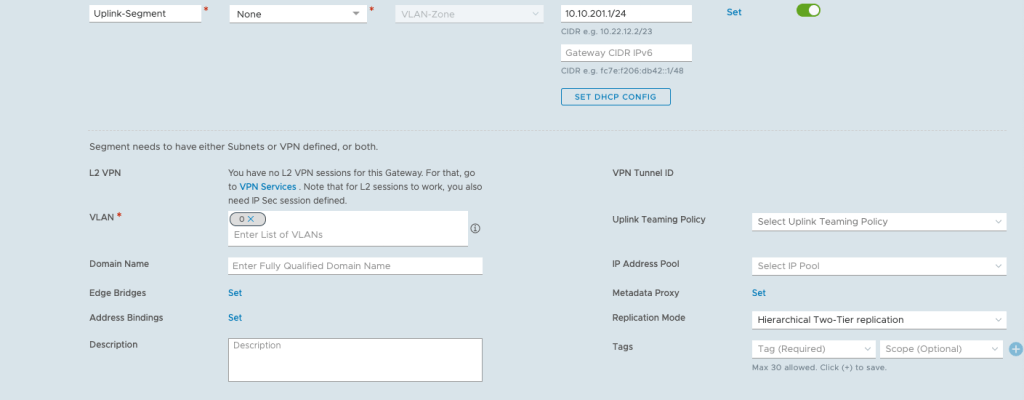


## NSX VLAN Segments

We want to create one more segment for our future Tier-0 router to use to connect to our physical network. I’m naming my segment Uplink-Segment and it belongs to the VLAN-Zone transport zone.

Pay no attention to the connectivity drop down. It shows as required, but it isn’t because you haven’t deployed a T0 router yet to connect it to.

For the subnets, I’ve put in an address on my Edge-Uplinks portgroup. This was my 201 VLAN from the previous examples. The other important thing to note is the VLAN ID. You need to set this, but it should be set to 0 since tagging is done at the vSwitch level.



## Summary

After setting up our Tier-1 router and the overlay segments, you should be able to deploy some virtual machines to those portgroups and have them communicate with each other. There is no North/South routing configured yet for your physical network to access the overlays. This will be covered more during the deployment of the Tier-0 router in the [next post](https://theithollow.com/2020/07/14/tier-1-gateway-and-nsx-segments/).

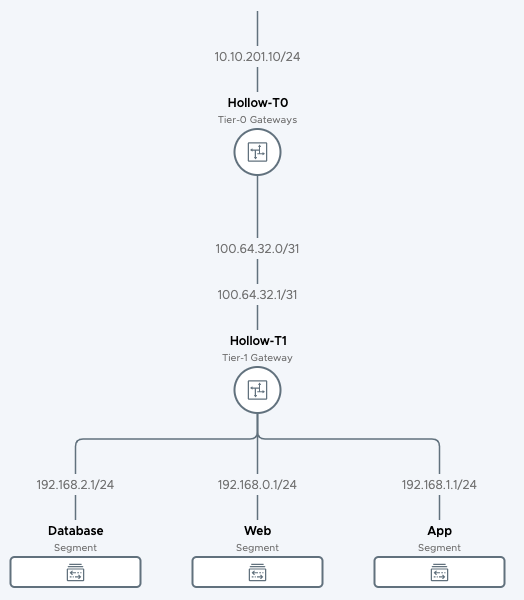
# Tier-0 Gateway

*July 14, 2020*[*3*](https://theithollow.com/2020/07/14/tier-0-gateway/#comments)*By* [ERIC SHANKS](https://theithollow.com/author/eshanks/)

This post will review the deployment and configuration of a Tier-0 gateway to provide north/south routing into the NSX-T overlay networks.

The Tier-0 (T0) gateway is where we’ll finally connect our new NSX-T backed overlay segments to the physical network through an NSX-T Edge which was previously deployed.

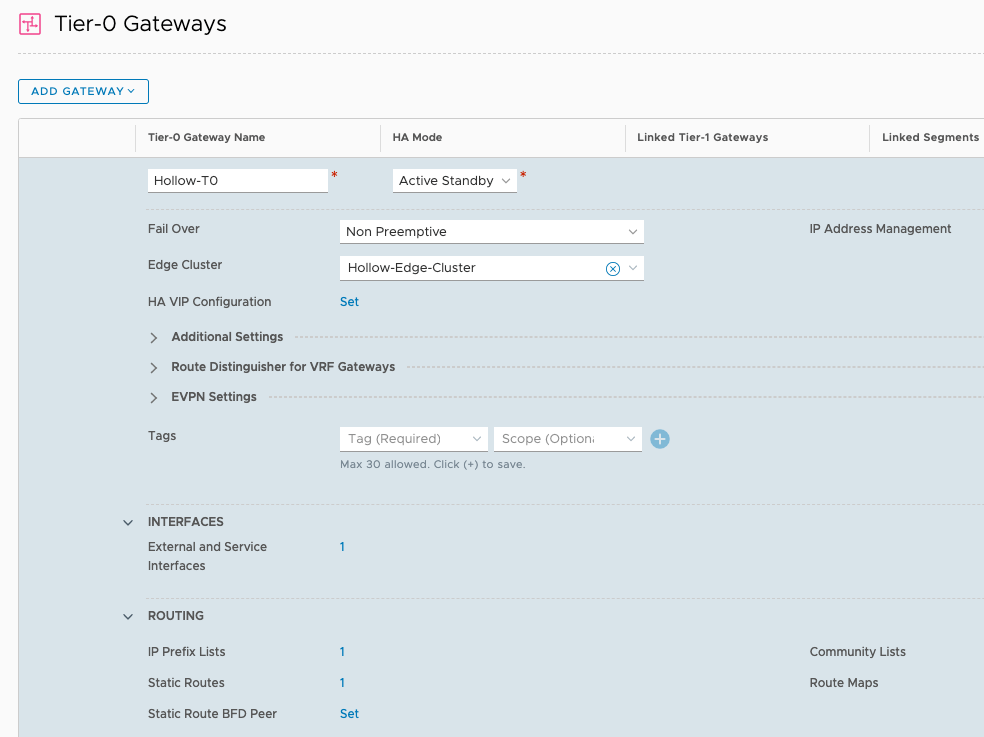
The Tier-0 gateway will connect directly to a physical VLAN and on the other side to our T1 router deployed in the previous post. From there, we should have all the plumbing we need to route to our hosts and begin using NSX-T to do some cooler stuff. In the end, the network topology will look something like this:



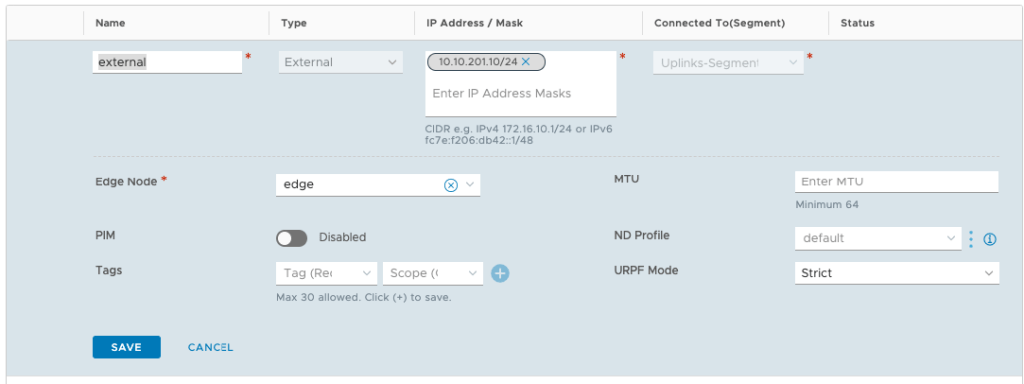
## Deploy the Tier-0 Gateway

Within the NSX-T Manager navigate to Networking –> Tier-0 Gateways. From there click the ADD GATEWAY button.

Give the gateway a name and pick an HA Mode. In our case the HA mode doesn’t really matter because we only have a single edge deployed. In a production setting, this becomes an important consideration. Next, scroll down until you get to the Interfaces section. Click the link next to interfaces to assign an interface to the router.

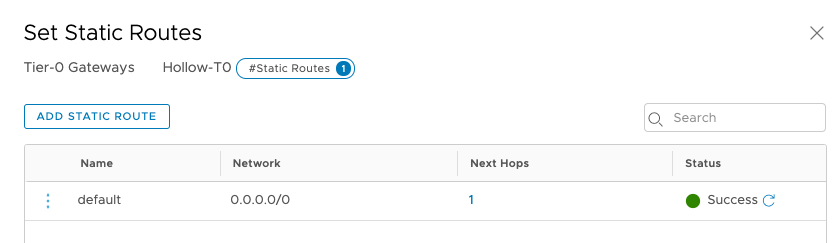


Name the interface, and select the type of external. Then enter an IP Address / subnet for the IP Address that will reside on the external interface side of the router. This should be a routable IP Address on your physical network (VLAN 201 from previous posts).

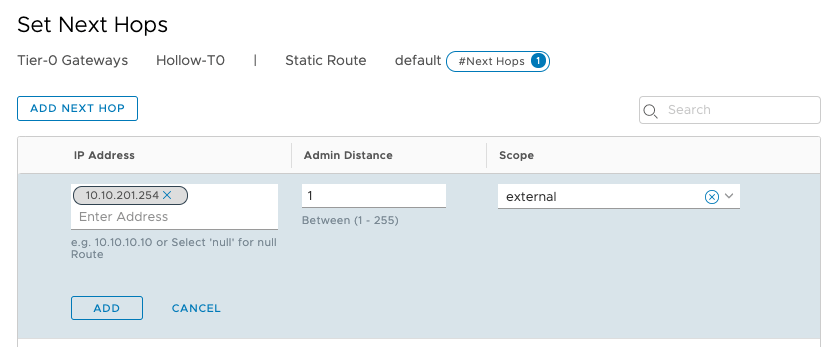


In the Connected to(Segment) box select the Uplink-Segment that was created during the segments post. Then finally select the edge node that will house the resources for this T0 gateway. Click Save to save the interface configuration and go back to the T0 router setup.

Under Static Routes, I added a default routing rule that sends any traffic to 0.0.0.0/0 through my physical switch.

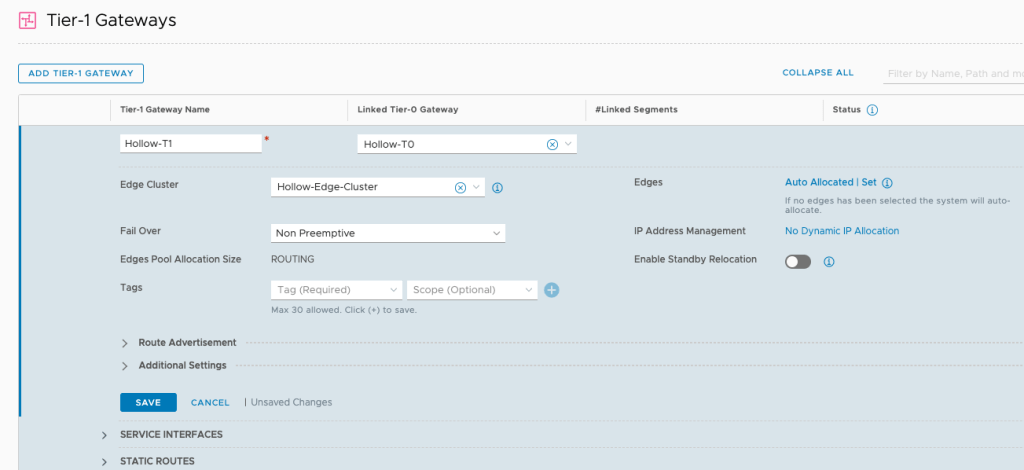


My next hop address is the Physical Switch gateway address on the VLAN 201 network. In my case its 10.10.201.254.



## Connect the Routers

Your Tier-0 Router is now ready to go, lets connect a couple of things together to finish this up. Go to your Tier-1 router created in the previous post and update the Linked Tier-0 Gateway drop down to reflect your new Tier-0 router. Save the configuration and you’ve now connected the T0 down to the Tier-1 and subsequent NSX segments.



## Configure Routing Rules

The last step I can’t help with too much. We need to send traffic from our physical network down to the NSX-T overlay segments through routing rules. When setting up the segments in a previous post, I used the networks below as NSX segments.

NOTE: These networks are just test networks to demonstrate how NSX-T can be used with VMs. These segments are not necessary for vSphere 7 with Kubernetes, but a good way to validate that NSX is working. I am using these networks alongside of the vSphere 7 on Kubernetes deployment.

|  |  |
| --- | --- |
| Segment Name | Segment CIDR |
| Web | 192.168.0.1/24 |
| App | 192.168.1.1/24 |
| Database | 192.168.2.1/24 |

These networks cannot be accessed from outside the overlay networks until you configure routing rules or a dynamic routing protocol. In my case, I updated my Layer 3 switch so that a route for 192.168.0.0/16 points to my Tier-0 Uplink IP Address which was 10.10.201.10.

In your case, you can add static routes, or setup a routing protocol to automatically add these routes. You’ll have to decide for yourself the best method in your lab, with your equipment.

## Summary

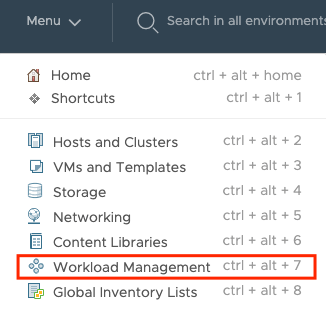
You’ve now deployed the Tier-0 router and connected your NSX-T backed Overlay segments to your physical network. You can begin using NSX-T for vSphere 7 on Kubernetes by following the [next post](https://theithollow.com/2020/07/14/enable-workload-management/), or whatever network segmentation/routing/stretched Layer 2 thing you can come up with. Good luck with your NSX-T labbing!

# Enable Workload Management

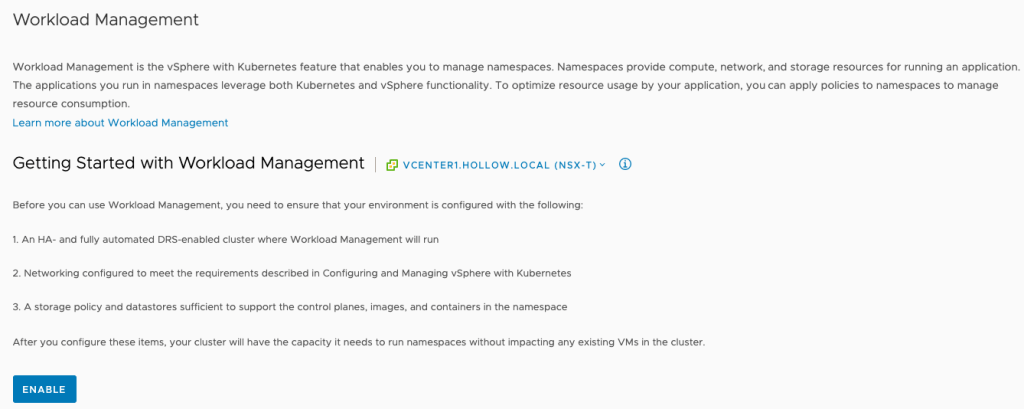
*July 14, 2020*[*4*](https://theithollow.com/2020/07/14/enable-workload-management/#comments)*By* [ERIC SHANKS](https://theithollow.com/author/eshanks/)

This post focuses on enabling the workload management components for vSphere 7 with Kubernetes. It is assumed that the vSphere environment is already in place and the NSX-T configuration has been deployed.

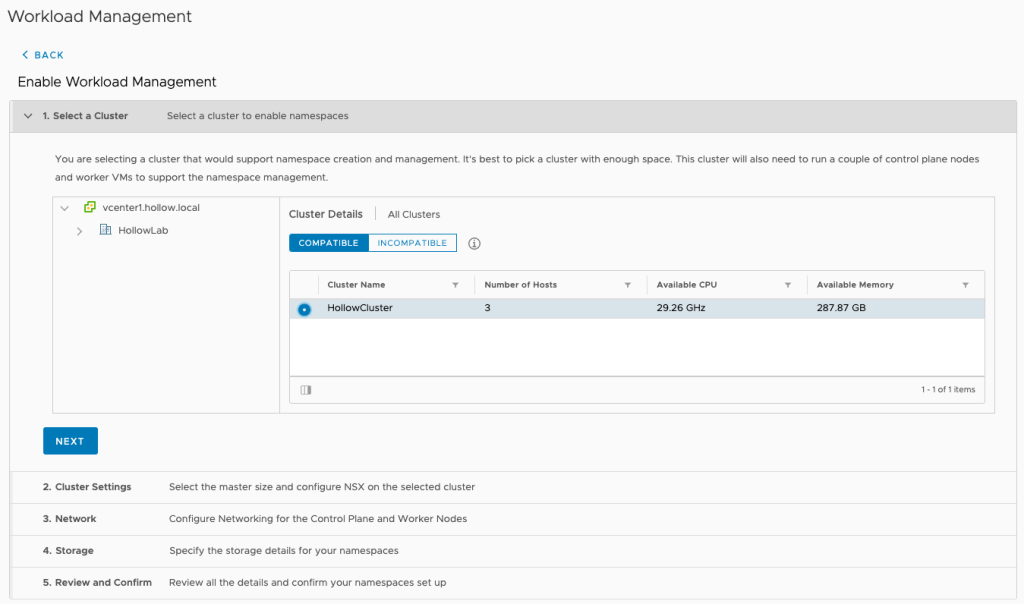
To enable workload management, login to your vCenter as the administrator@vsphere.local account. Then in the Menu, select Work



Within the Workload Management screen, click the ENABLE button.



The first screen in the wizard, will list your compatible vSphere clusters. These clusters must have HA and DRS enabled in fully automated mode. If you are missing clusters, make sure you have ESXi hosts on version 7 with HA and DRS enabled. You’ll also need a Distributed switch on version 7 for these clusters.

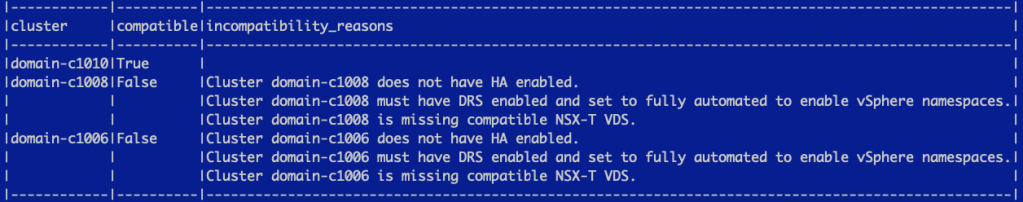


If you’re having trouble finding information about why your cluster isn’t listed as compatible, you can run the command below to list why your cluster is incompatible.

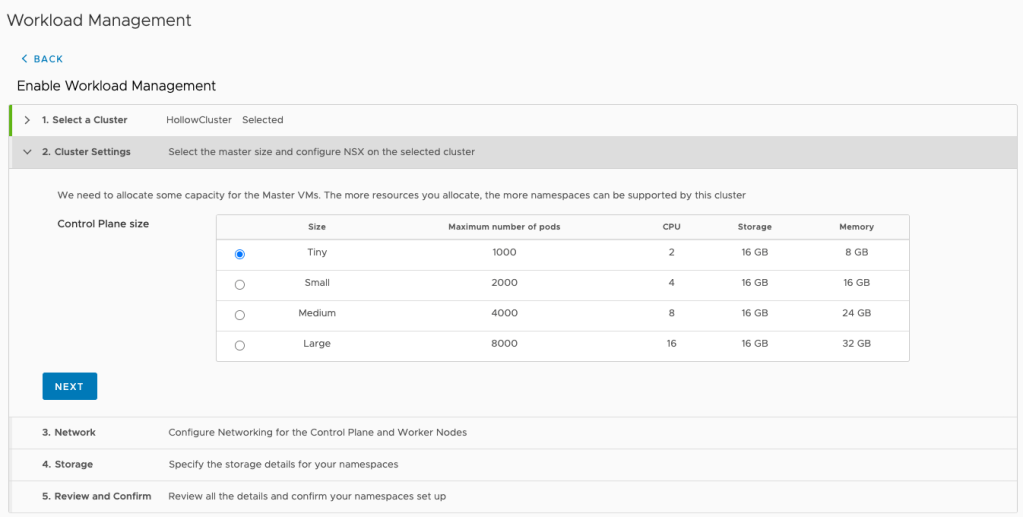
dcli com vmware vcenter namespacemanagement clustercompatibility list

Code language: PHP (php)

You can see why two of my vSphere clusters are incompatible from running the command above. If you have more trouble with enabling “workload management” I recommend reading [this post from William Lam](https://www.virtuallyghetto.com/2020/05/troubleshooting-tips-for-configuring-vsphere-with-kubernetes.html).

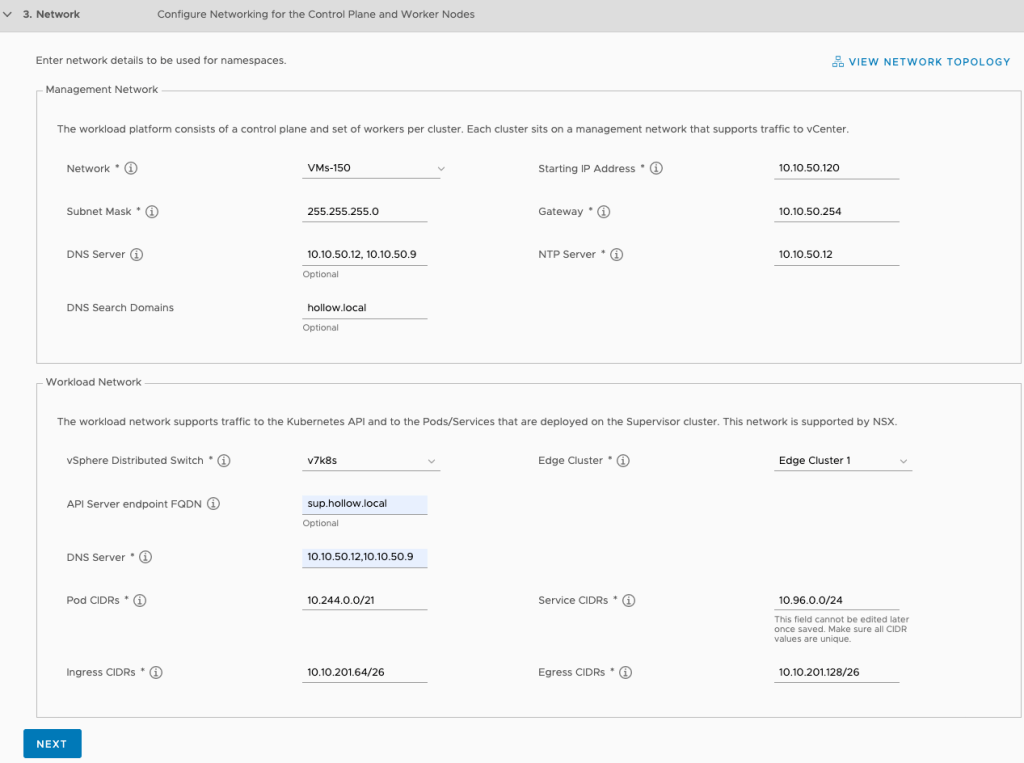


Next, you must select a control plane size. This defines the VM size of the control plane nodes for your Kubernetes clusters. Since I have limited resources in my lab, I’ve chosen the Tiny size.



The next screen requires you to fill out networking information. First, we’ll discuss the management network. Each of the control plane nodes that will be deployed will have a network connection on the management network. (VLAN 150 if you’ve been following the series). Select the management portgroup for your network, and then the starting IP Address to be used for new nodes. They will increment from this IP Address so be sure to have at least five IP Addresses available. Next, set the subnet mask and the gateway, DNS info and NTP configs.

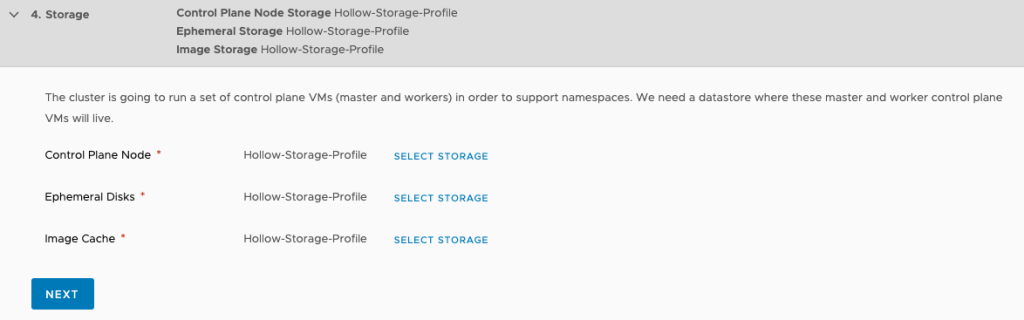
Once you’re through with the management network, its time to configure the workload network. Select the Distributed switch that will be used, and the Edge cluster. Next enter an API Server endpoint DNS name. This will be associated with the first “Starting IP Address” IP created in the management network (So 10.10.50.120 in this example). You will want to add a DNS entry for this FQDN. The Pod CIDRs and Service CIDRs should be fine, but you can change this if you’d like. Lastly, you need to enter Ingress and Egress CIDRs. This IP Address range should come from your external network. In my case this is VLAN 201. I’ve carved two /26s aside for ingress/egress access.

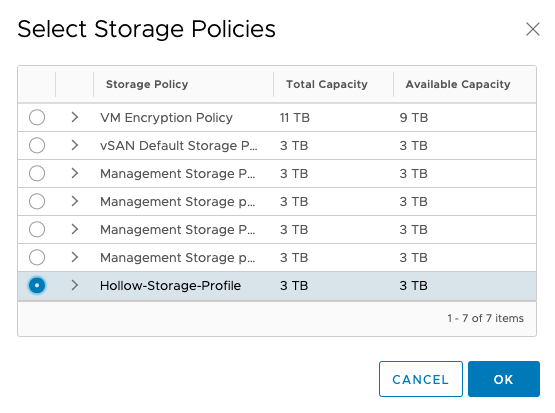


Next up, its time to setup the storage. You’ll need to store three different types of objects on a datastore.

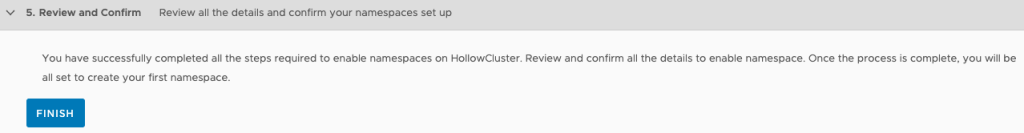
* Control Plane Node – virtual disks for control plane nodes
* Ephemeral Disks – vSphere pod disks
* Image Cache – container image cache

For each of these objects you’ll need to select a storage policy that defines what datastores are compatible. I created a Hollow-Storage-Profile policy as a pre-requisite that selects my vsanDatastore. Select the storage profile configured for each of these components.

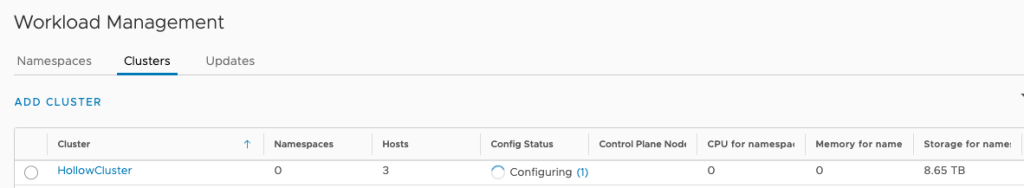




Once you’re done, click Finish and go get some coffee. No, I mean it, go drive to Starbucks or start a fresh pot of coffee and wait for it to be ready. Then drink it, and then come back. This process took about an hour in my cluster to complete.



As the configuration is running, you can view some minimal status information in the clusters screen. You can see here it’s configuring.



As I set this up in my lab, I had a couple of challenges and needed to find details about what was happening. If you need to find log details, login to the vCenter appliance shell and cat or tail the following two log files to give you information about whats happening.

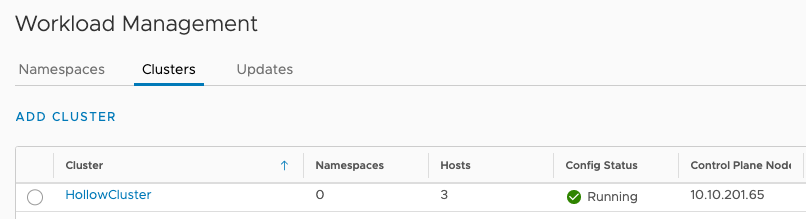
tail -f /var/log/vmware/wcp/wcpsvc.log

tail -f /var/log/vmware/wcp/nsxd.log

Code language: JavaScript (javascript)

NOTE: there are some items which might fail, or give you a 404 error. These seem to be normal operations that will be retried via a control loop. So getting an error here and there might not be anything to worry about.

When complete, you should see your cluster has a “Config Status” of “Running”. You’ll also see the control plane node IP Address which comes form the Ingress CIDR created previously.



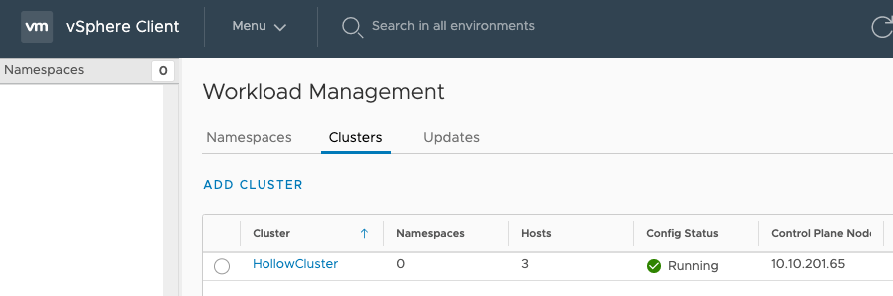
## Summary

Enabling the Workload Management components aren’t too labor intensive once you have the prerequisites done, but it does take a while to enable. You should have a supervisor cluster created and ready to be used at this point. Stay tuned and we’ll cover what to do with that cluster now that you’ve setup vSphere 7 with Kubernetes!

# Creating Supervisor Namespaces

*August 17, 2020*[*2*](https://theithollow.com/2020/08/17/creating-supervisor-namespaces/#comments)*By* [ERIC SHANKS](https://theithollow.com/author/eshanks/)

Congratulations, you’ve deployed the Workload Management components for your vSphere 7 cluster. If you’ve been following along with the series so far, you’ll have left off with a workload management cluster created and ready to being configuring your cluster for use with Kubernetes.



The next step in the process is to create a namespace. Before we do that, it’s probably useful to recap what a namespace is used for.

## Namespaces the Theory

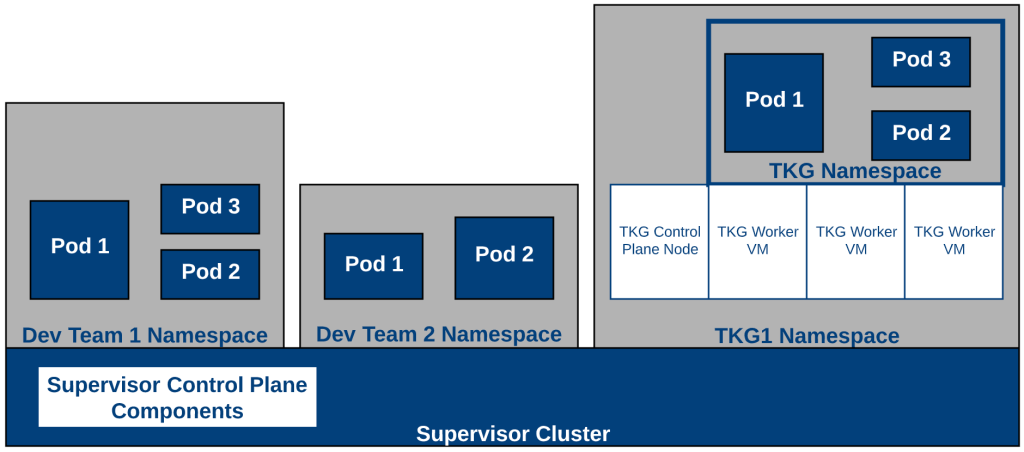
Depending on your past experiences, a namespace will likely seem familiar to you in some fashion. If you have a kubernetes background, you’ll be familiar with namespaces as a way to set permissions for a group of users (or a project, etc) and for assigning resources. Alternatively, if you have a vSphere background, you’re used to using things like Resource Pools to set resource allocation.

A Supervisor Cluster namespace is a combination of resource allocations and permissions set within the Supervisor Cluster. When you create a Supervisor Namespace, you’ll assign who has access to use it, and how many of the ESXi cluster’s resources you can use (much like a resource pool).

When you enabled the Workload Management components, you created a special Kubernetes cluster called the “Supervisor Cluster”. You can continue to deploy virtual machines in this cluster, and you can also deploy kubernetes pods as a “pod vm” which is basically a container with some special wrapping so they are better isolated, like a virtual machine is.

To better illustrate things, the diagram below demonstrates that you can carve up the Supervisor Cluster to suit your needs. The diagram below has two namespaces for two different Development teams (you could carve these up by project, app, or whatever you’d like, dev teams is just an example). Those two namespaces would have different permissions so one development team couldn’t see the pods/resources in the other namespaces. They are also sized differently.

The third namespace on the far right side, is a namespace similar to the first two, but instead of it running PodVMs (think of them as containers for now) it’s running … another Kubernetes cluster within that namespace called a Tanzu Kubernetes Grid (TKG) cluster. That cluster will have resources allocated by the Supervisor Namespace, but then pods can run on those VMs. **NOTE:** I can almost hear you asking why I’d build another cluster within a cluster, but that will need to wait for another post for now.

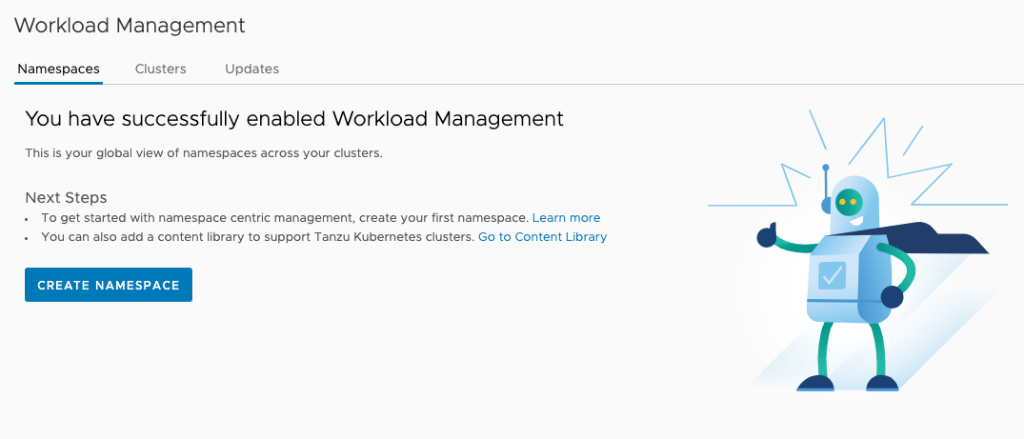


Hopefully you’ve gotten the purpose behind Supervisor Namespaces now, and are ready to configure your cluster.

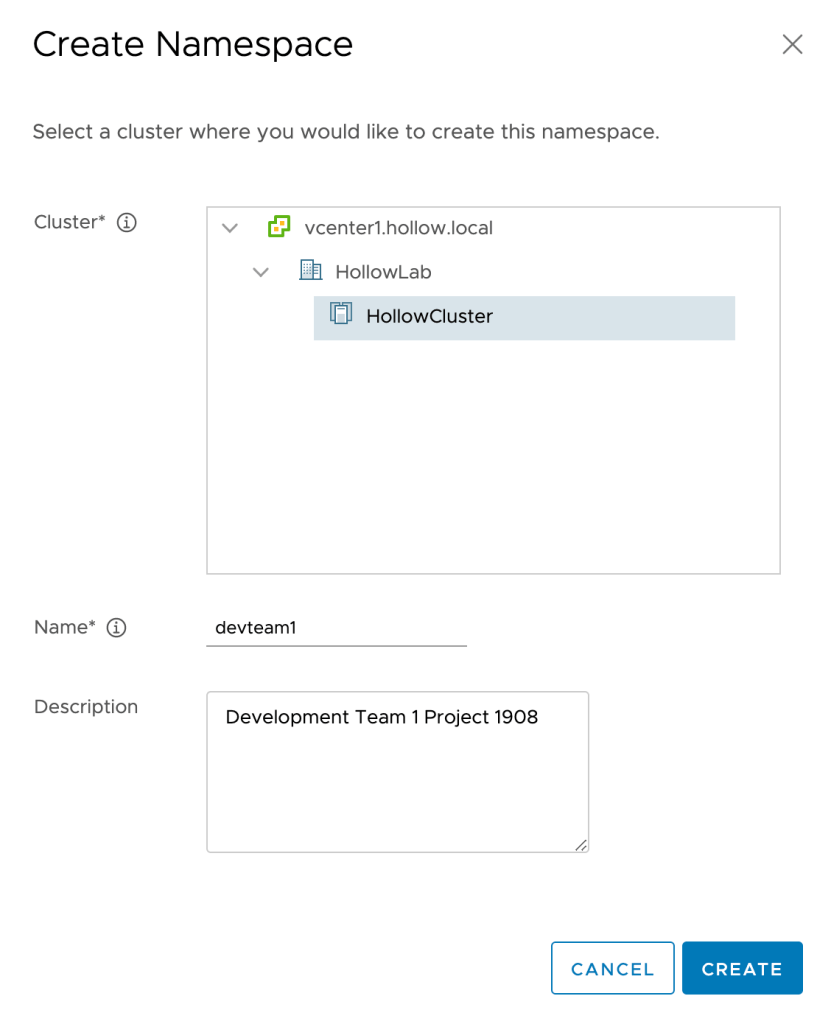
## Create a Namespace

Within the Workload Management menu, select the Namespaces tab. If this is your first namespace, you’ll be greeted with a fancy splash page with the robot thingy on the right side. Sorry, I don’t know it’s name, but it really should have one..)

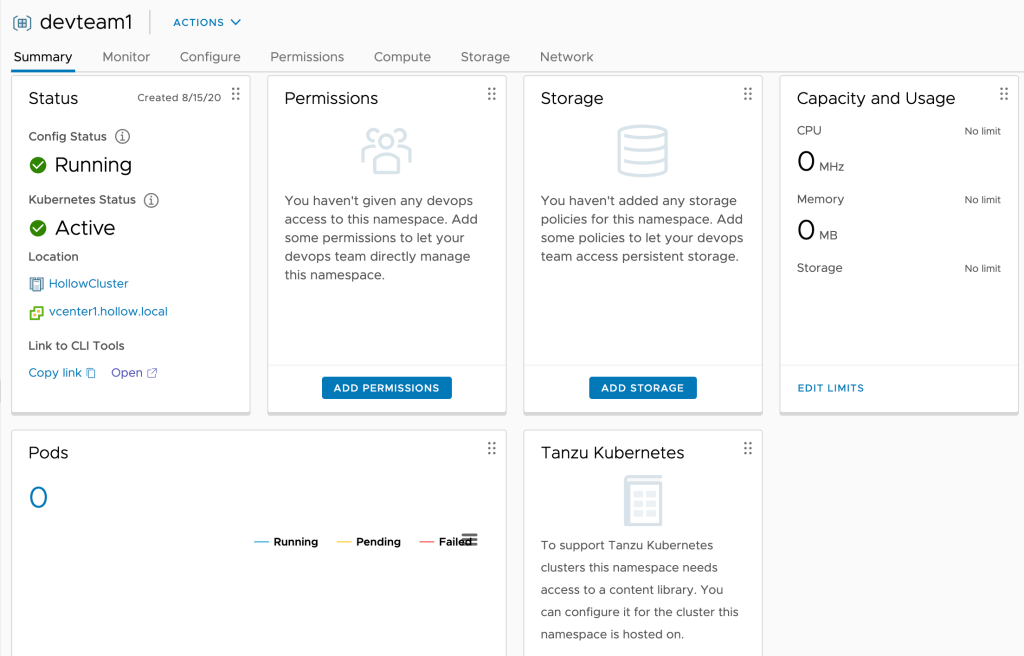
Click Create Namespace.



On the first screen select which Supervisor Cluster to create the namespace, then give it a name and a description. Then click Create.

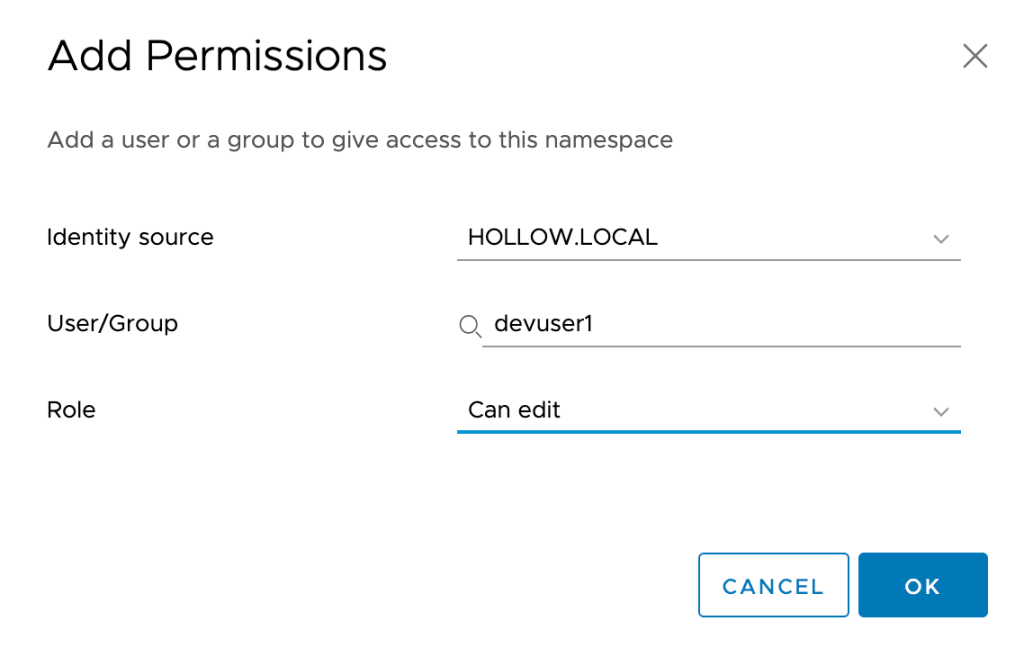


Your Namespace is now created. You should probably do some additional configurations though now. Your screen should look something like this now.

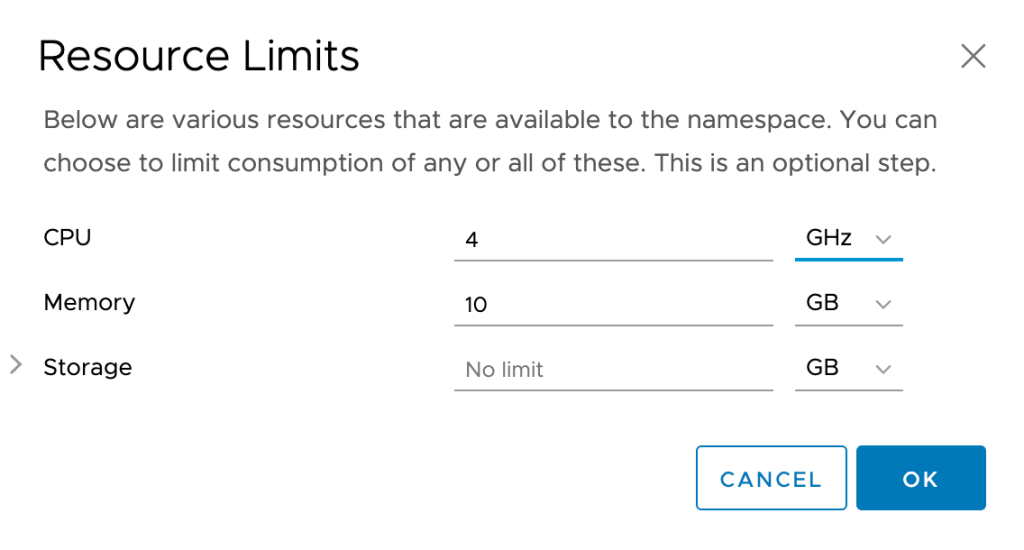


You can see a couple of buttons on that overview screen. Let’s set some permissions so that some of our users can use this namespace soon. Click the App Permissions button within the “Permissions” tab. Select your identity source, a user or a group from that source, and either view or edit permissions. I’ve used edit permissions so we can use this user in later posts. Click OK.

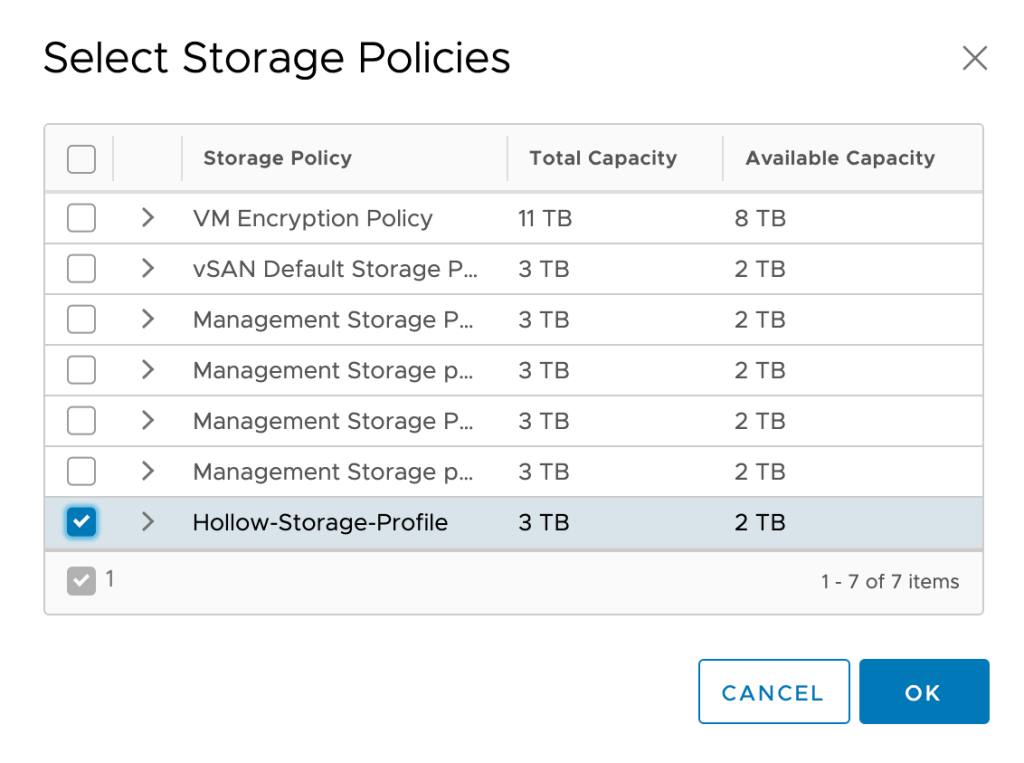
NOTE: View permissions = Get,list,watch permissions while Edit permissions also include Update,Delete,Patch, and Update.



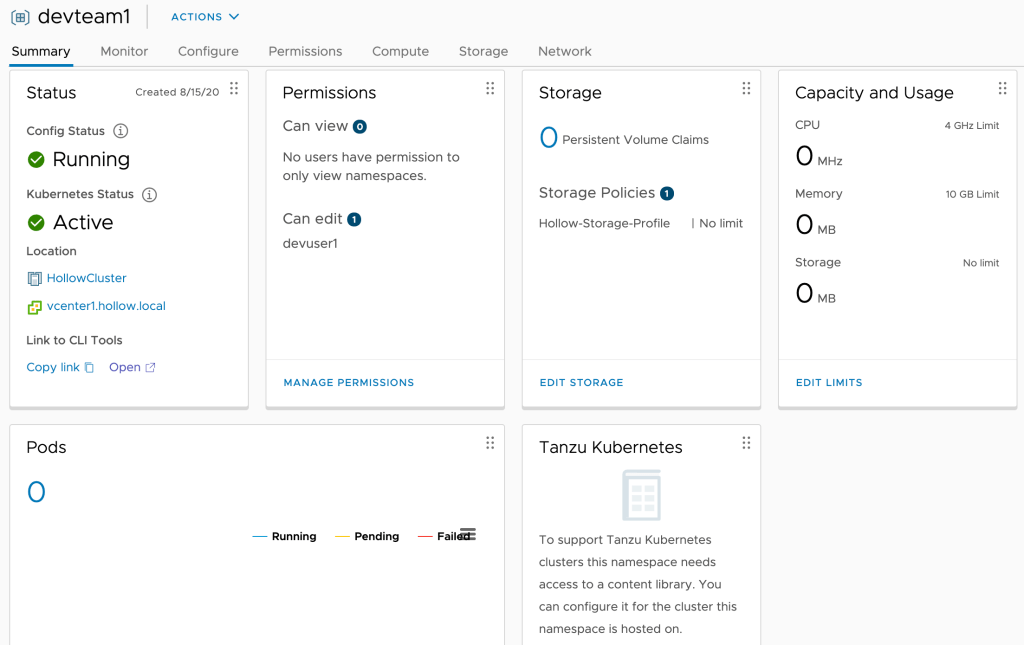
Next, in the Capacity and Usage tab, click the Edit Limits link to set some resource limits on the namespace. Enter limits for CPU, Memory and Storage for this namespace and then click OK.



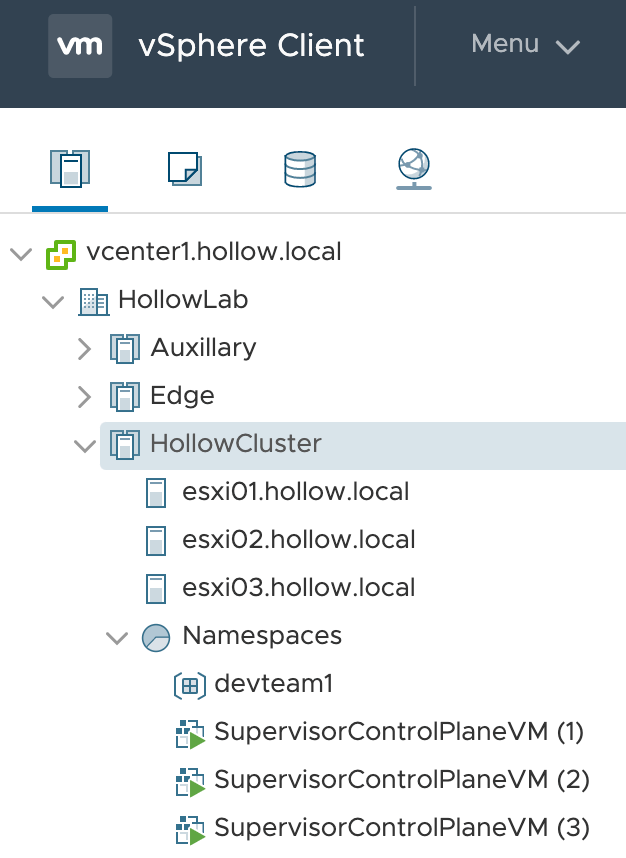
The last setting we’ll do in this post is the Add Storage button in the Storage tab. Here you’ll select a storage policy that can be used with this namespace. These are standard storage policies that can be used on vsphere datastores to select the correct ones.



In the end, your dashboard will look something like the one below.



And you’ll notice the namespaces are also listed under the “Hosts and Clusters” view if you’re logged in as administrator@vsphere.local, very similar to a resource pool would look.



## Summary

Supervisor Namespaces are a way to isolate resources, assign role based access controls, and allocate physical resources for your users. Stay tuned for the next post when we login to the Supervisor Cluster within this namespace.

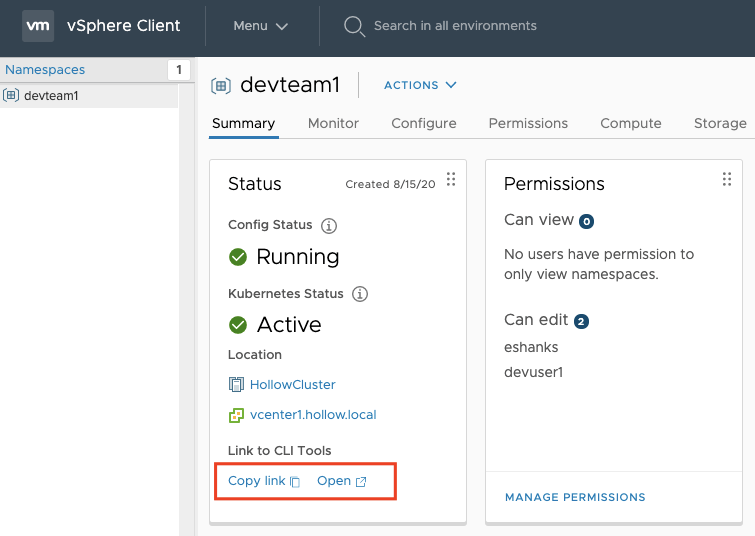
# Connecting to a Supervisor Namespace

*August 24, 2020*[*2*](https://theithollow.com/2020/08/24/connecting-to-a-supervisor-namespace/#comments)*By* [ERIC SHANKS](https://theithollow.com/author/eshanks/)

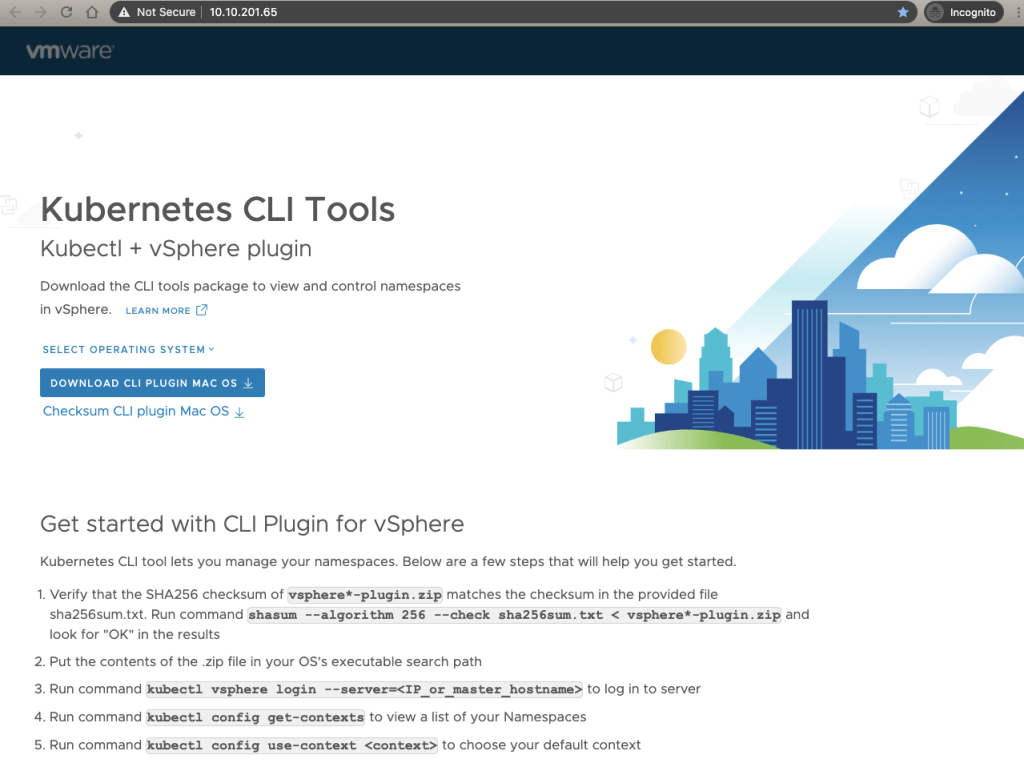
In this post we’ll finally connect to our Supervisor Cluster Namespace through the Kubernetes cli and run some commands for the first time.

In the [last post](https://theithollow.com/2020/08/17/creating-supervisor-namespaces/) we created a namespace within the Supervisor Cluster and assigned some resource allocations and permissions for our example development user. Now it’s time to access that namespace so that real work can be done using the platform.

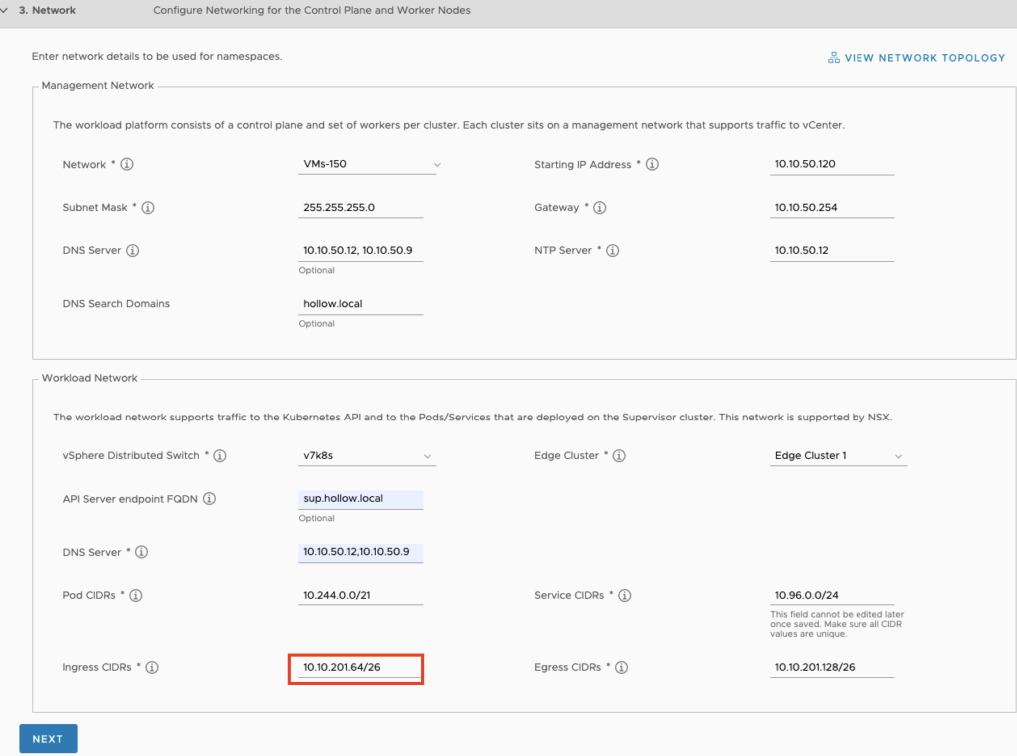
First, login to vCenter again with the administrator@vsphere.local account and navigate to the namespace that was previously created. You should see a similar screen where we configured our permissions. In the Status tile, click one of the links to either open in a browser or copy the URL to then open in a browser.



When you open the link, you should see a webpage like this one. This is the Kubernetes API endpoint.



Notice the IP Address in the URL. It should seem pretty familiar to you because you entered the CIDR range during the [workload control plane setup](https://theithollow.com/2020/07/14/enable-workload-management/). Remember setting up the Ingress CIDRs? This is where the IP Address came from.

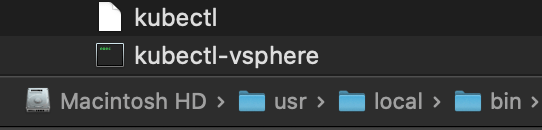


Now, that you’ve got the Kubernetes API endpoint page displayed, you can follow the directions listed on the page to setup your client. This is nice since once you setup the namespace, you can give this web page to anyone using the Kubernetes clusters, to then use the namespace, get the clients, and describe the process. I’ll walk through those steps below.

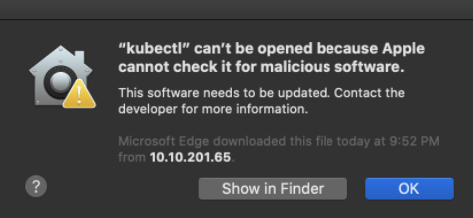
## Download CLI Plugin

There will be a blue link on the Kubernetes API endpoint page that is a download link for the two CLIs you’ll need. These will come zipped and you’ll need to uncompress them after downloading.

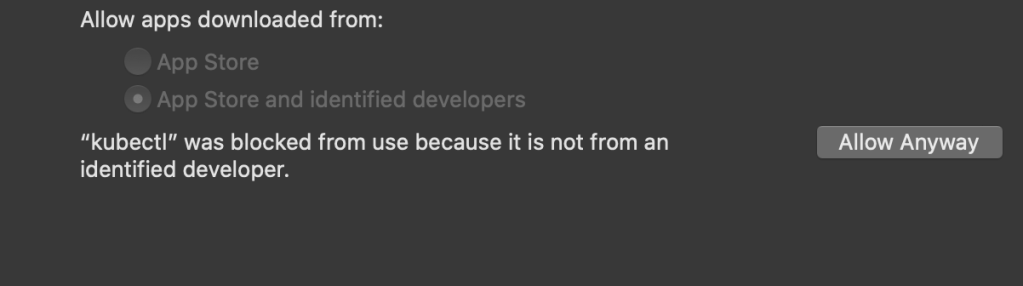
Once you’ve downloaded the CLI tools, you need to put them into your Operating System Path. On my Mac, I’ve moved these to /usr/local/bin and set permissions so that I can execute these executables.



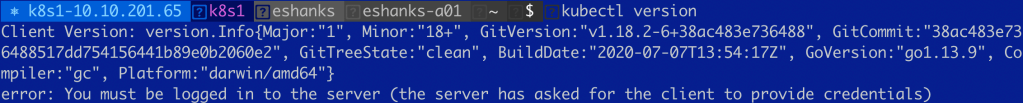
To verify that the CLIs are working, open a shell session and run the command: kubectl version just to see if the CLI will respond with the version. In my case, running it gave me a security error that I needed to fix because Apple couldn’t check it for malicious software.



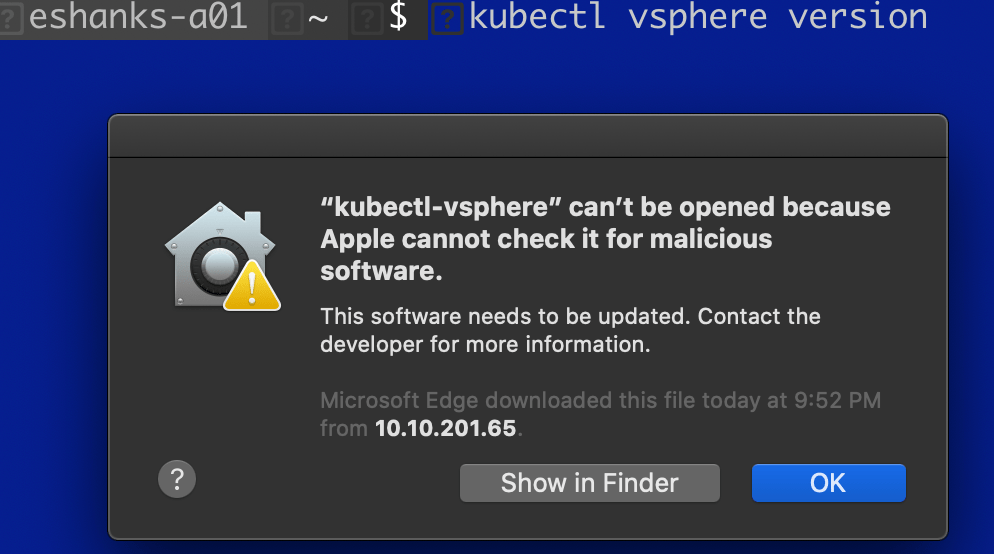
To fix this on a Mac, go into Security & Privacy –> General Tab. Here you’ll see the CLI tool that was just executed and a button to Allow the software to be executed.



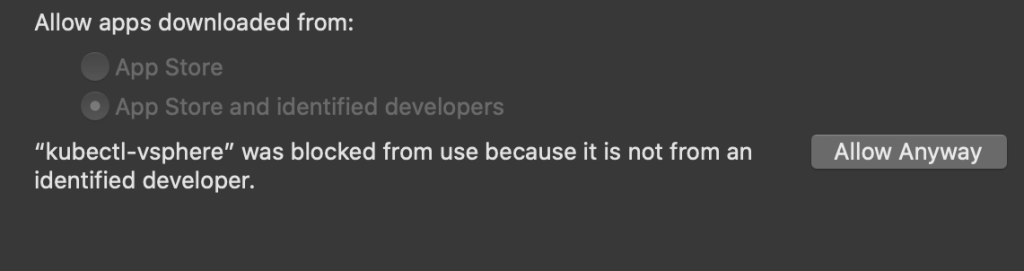
Running the kubectl version command again, gives me the version info as expected.



Next, we need to repeat the process with the kubectl vsphere CLI.



Again, ensure the software can be executed.



Once you can see the versions for both CLI components we can move on with accessing our cluster.

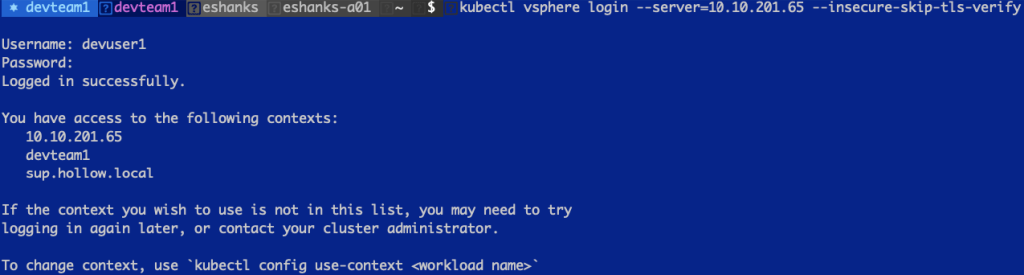


## Connect to the Kubernetes Namespace

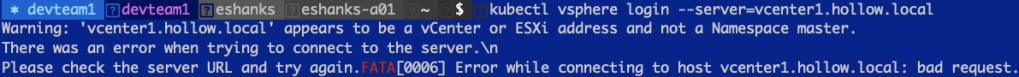
Now that the CLI tools are working, we need to first authenticate with the vSphere API. You can do this by running:

kubectl vsphere login --server=[ip\_or\_fqdn\_of\_Kubernetes\_API\_Endpoint]

If you are not using trusted certificates, you’ll need to append the --insecure-skip-tls-verify switch as seen in the screenshot below.



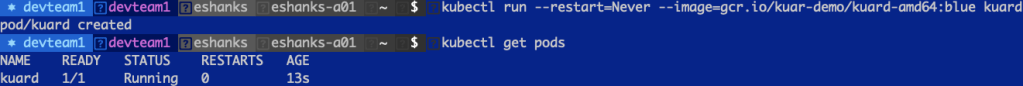
**NOTE:** I found the instructions a tad confusing. The Kubernetes API Endpoint webpage shows --server which I initially mistook for the vCenter server. It’s really asking for the Kubernetes API endpoint. In the screenshot below, you’ll see what happens if you use the wrong address, which was a helpful error message.



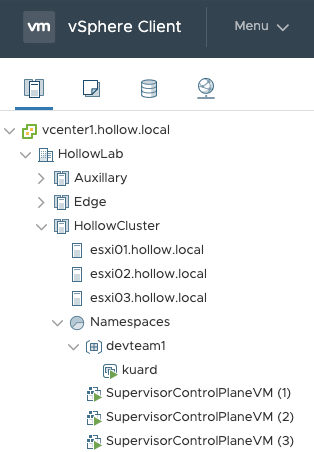
Once you’ve authenticated the CLI will display your Kubernetes Contexts which will now include the context for this Namespace we’ve created. You can change your context using:

kubectl config use-context [context name]

And after that you can start running Kubernetes commands. In the below screenshot, I’ve deployed the “Kubernetes Up and Running” container which you can see is running.



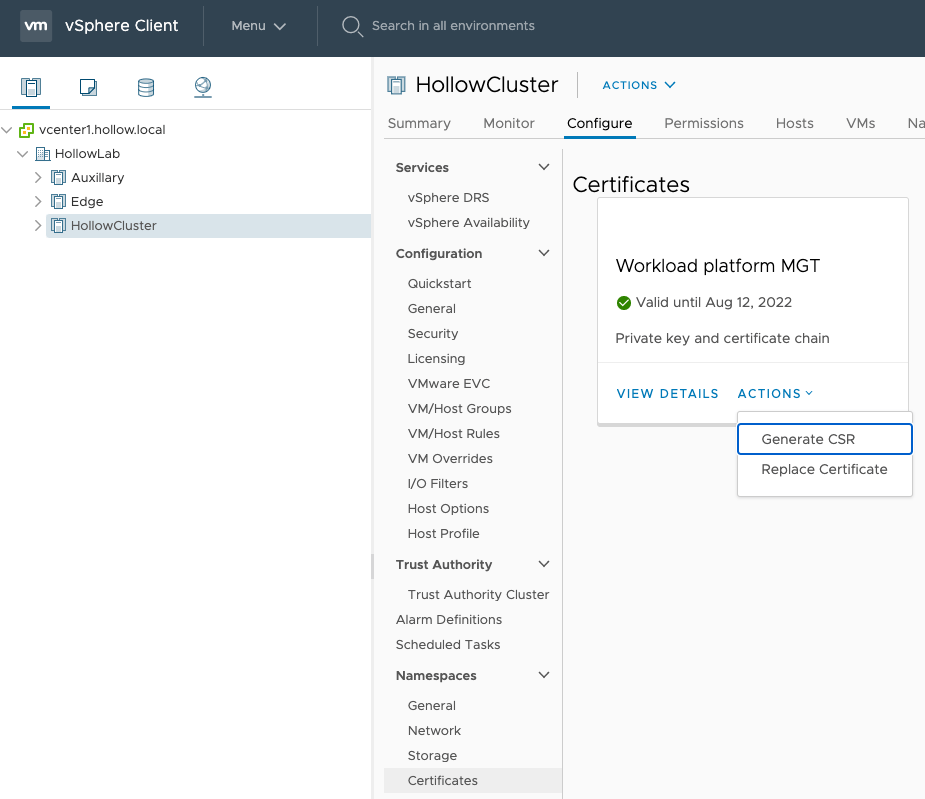
If you login to vCenter as the adminsitrator@vsphere.local user, you will even see the pod running in vCenter, under our namespace in the Hosts and Clusters view.



## Summary

You’ve now got a namespace where you can deploy PodVMs within your Supervisor cluster. We downloaded the CLI tools from the Kubernetes API Endpoint, authenticated to vCenter, and executed some kubectl commands against our cluster.

One last note. At this point you’re authenticating to the Kubernetes API Endpoint using a non-trusted certificate because we specified the --insecure-skip-tls-verify switch at login. We can use an existing CA to sign new certificates and use them. We may cover this in another post, but the location to generate a cert request is found in Supervisor Cluster –> Configure –> Namespaces –> Certificates –> Workload platform MGT.



# Replace vSphere 7 with Tanzu Certificates

*August 31, 2020*[*2*](https://theithollow.com/2020/08/31/replace-vsphere-7-with-tanzu-certificates/#comments)*By* [ERIC SHANKS](https://theithollow.com/author/eshanks/)

When setting up your vSphere 7 with Tanzu environment, its a good idea to update the default certificate shipped from VMware with your own certificate. This is a good security practice to ensure that your credentials are protected during logins, and nobody likes to see those pesky certificate warnings in their browsers anyway, am I right?

## Create and Trust Certificate Authority

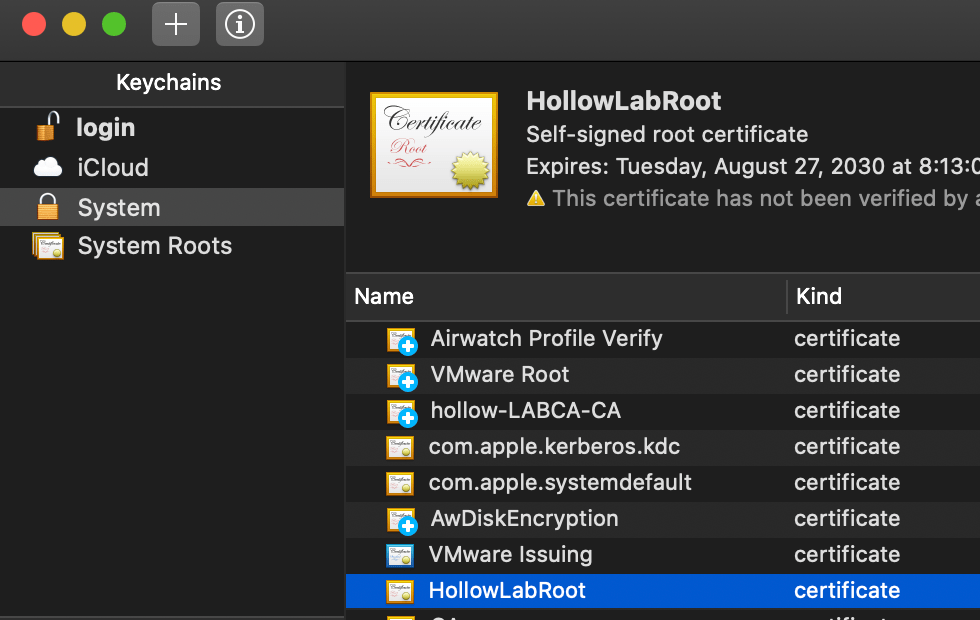
This section of the blog post is to create a root certificate. In many situations, you won’t need to do this since your organization probably already has a certificate authority that can be used to sign certificates as needed. Since I’m doing this in a lab, I’m going to create a root certificate and make sure my workstation trusts this cert first. After this, we can use the root certificate to sign our vSphere 7 certificates.

To create the CA certificate and Private key, download and install openssl and then run the command below, replacing the CN with your own Root Certificate Name.

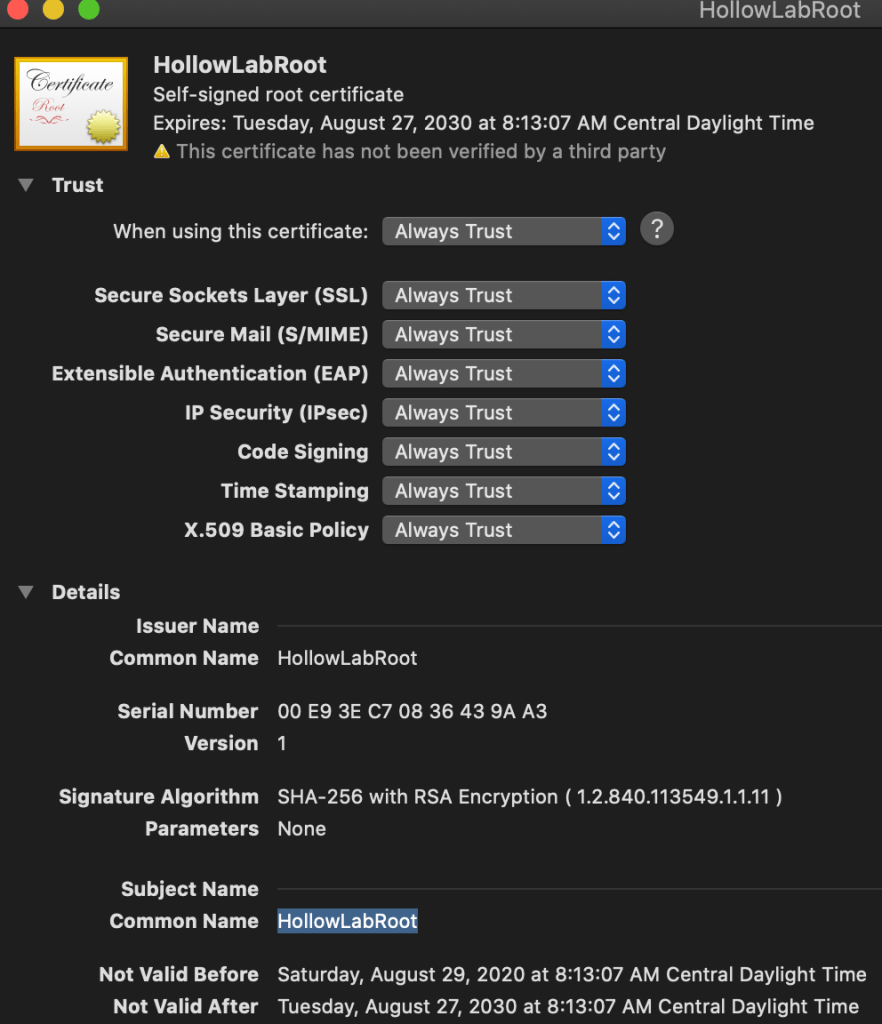
openssl req -nodes -new -x509 -days 3650 -keyout ca.key -out ca.crt -subj "/CN=HollowLabRoot”

After running the command you should have a ca.crt and ca.key file created in your working directory.

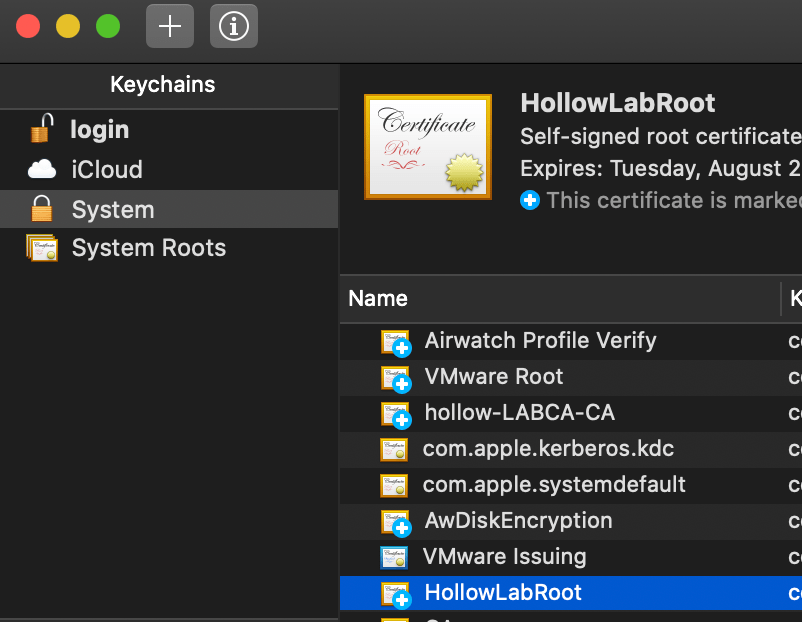
After the root certificate has been created, you need to distribute this certificate to any machine that will access your vSphere 7 with Tanzu API endpoints. On my mac, I’ve open the CA.crt file which opened the Keychain Access program.



Double click the certificate to open the settings and change the When using this certificate setting to Always Trust.

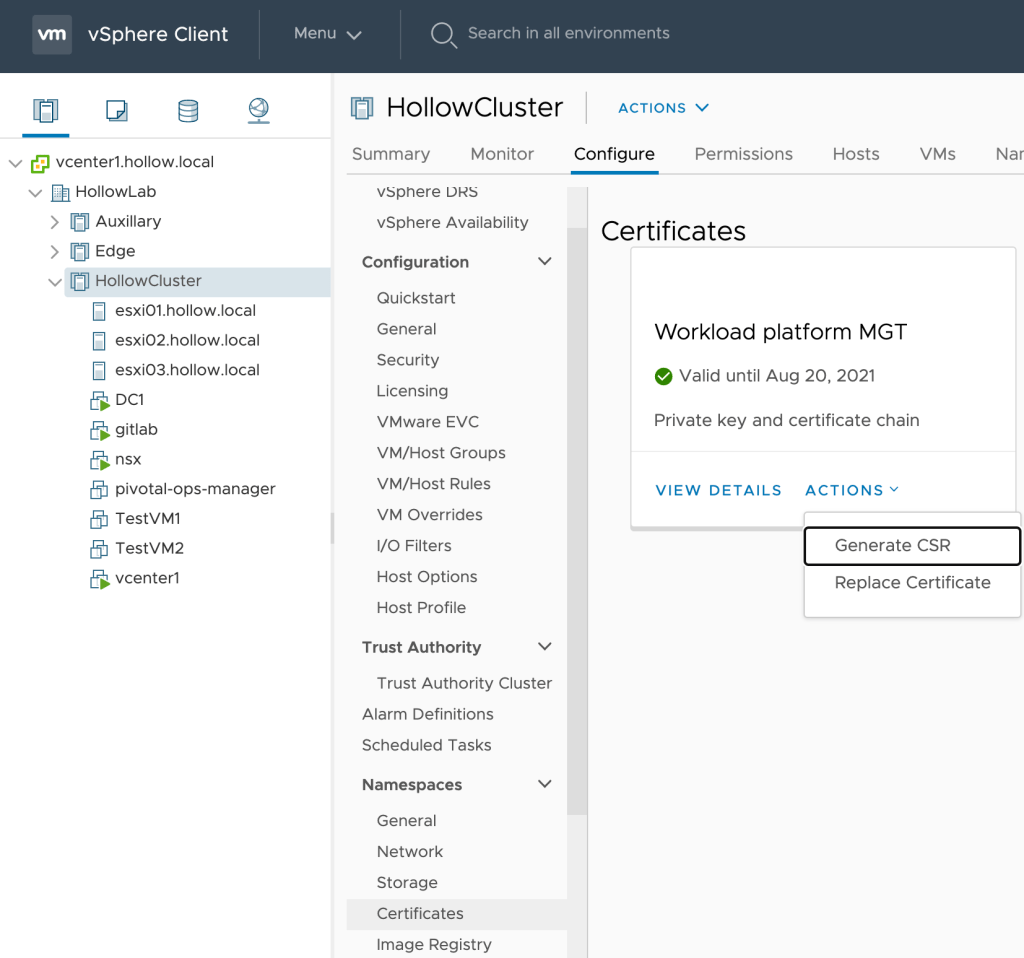


When you finish, there should be a little blue checkmark next to the certificate demonstrating that it’s trusted. Now, any certificates signed by this root certificate, will be automatically trusted by this workstation.

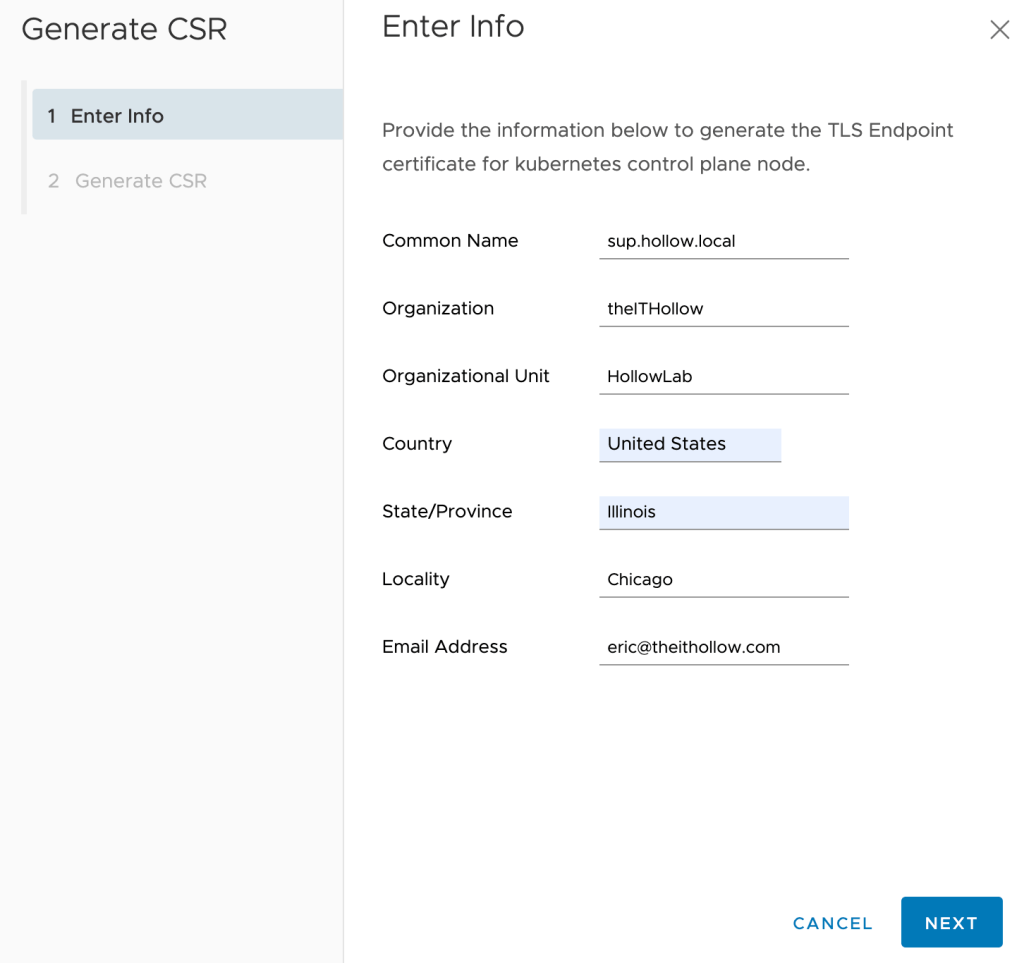


## Generate Certificate Signing Request

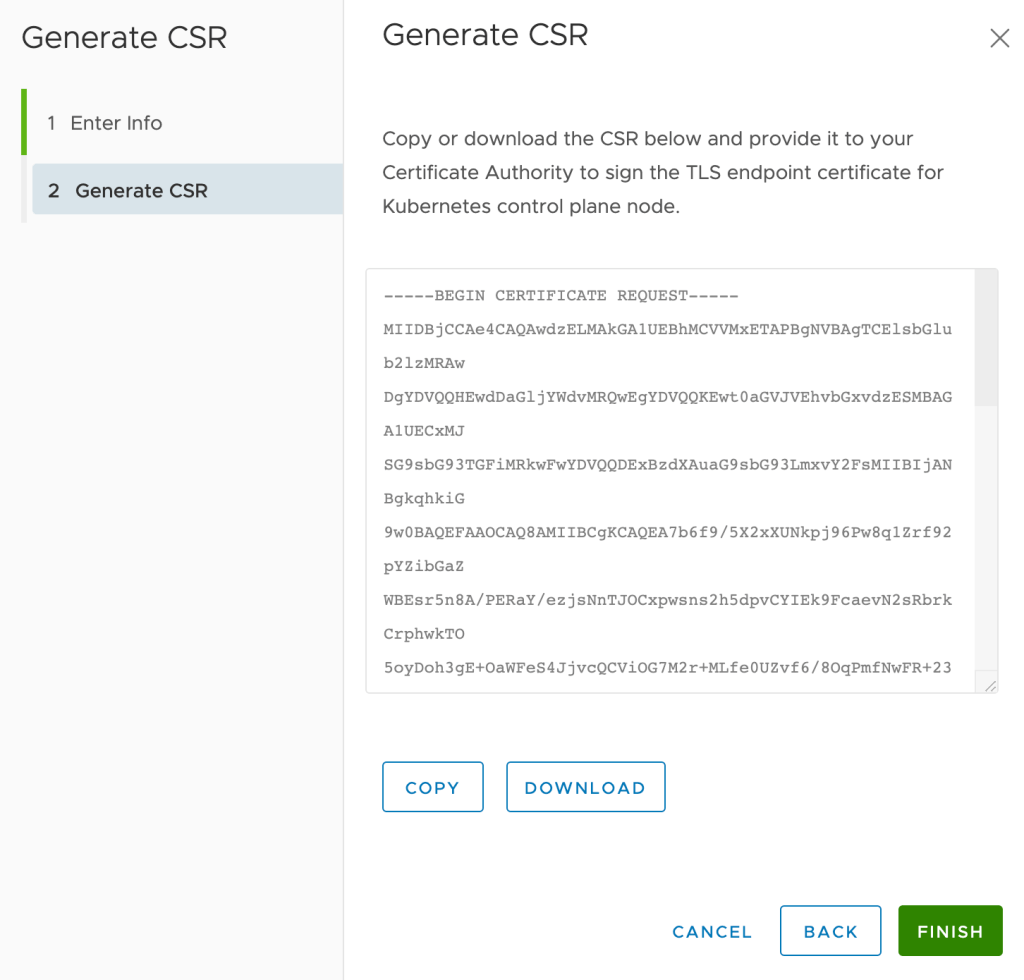
In the vCenter UI, navigate to the Supervisor Cluster that you’ve created and click the Configure tab and then the Certificates menu item. You should see a Workload platform MGT tile. Under Actions click Generate CSR.



Fill out the information relating to your own organization and then click Next.



On the last screen, click the DOWNLOAD button to download the certificate signing request.



I downloaded my CSR and named it v7wt.csr and placed it in my directory with my CA cert and key.

Lastly, it is probably a good idea to check the certificate signing request to make sure it looks ok. We can also check the Subject Alternate Name (SAN) properties on the cert at this time. You can view the csr by running:

openssl req -text -noout -verify -in v7wt.csr



## Sign the Certificate Request

Create an openssl config file. I’ve named mine ext.cnf. An important part of this config file is to have the [alt\_names] section updated so that it matches the SAN properties from the signing request. If you leave these off, you may strip off this SAN information during the signing and the cert will not be trusted by today’s browsers.

Replace the DNS.1 and IP.1 values to match your own CSR.

keyUsage = critical, digitalSignature, keyEncipherment

extendedKeyUsage = serverAuth

basicConstraints = CA:FALSE

nsCertType = server

subjectKeyIdentifier = hash

authorityKeyIdentifier = keyid,issuer:always

subjectAltName = @alt\_names

[alt\_names]

DNS.1 = sup.hollow.local

IP.1 = 10.10.201.65

Code language: PHP (php)

Once your config file is created, you can use the CA certificate and CA key to sign the CSR.

openssl x509 -req -in v7wt.csr -CA ca.crt -CAkey ca.key -out v7wt.crt -CAcreateserial -days 365 -sha256 -extfile ext.cnf

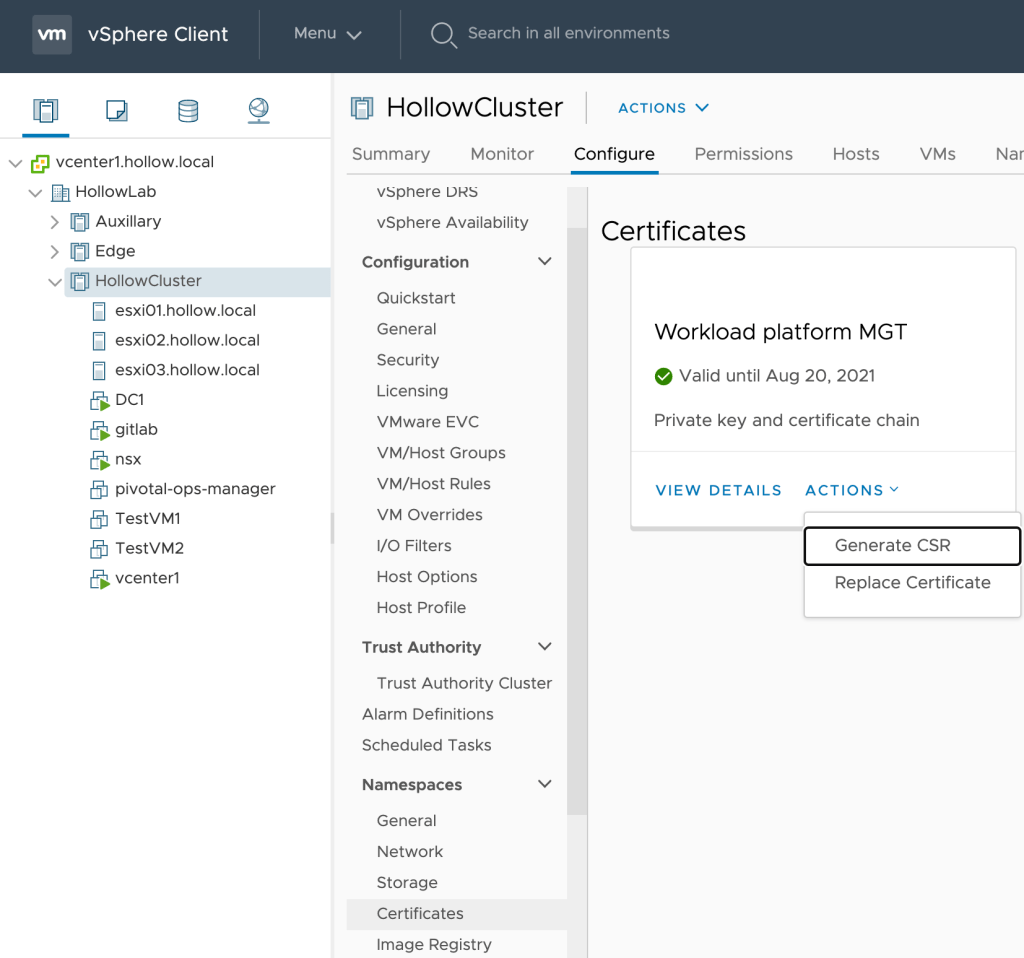
After running the above command you should now have a v7wt.crt and you can inspect it to make sure it still includes your SAN properties.

openssl x509 -in v7wt.crt -text -noout

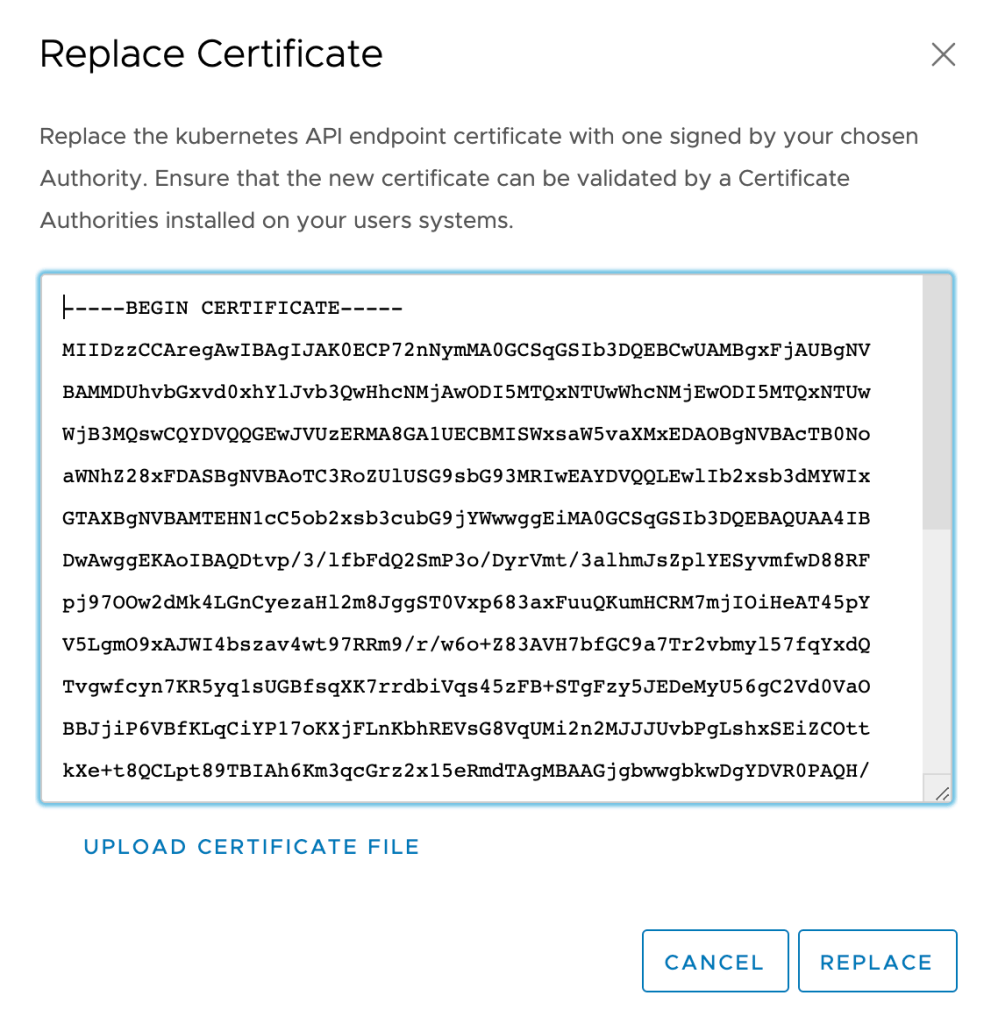


## Replace the Certificate

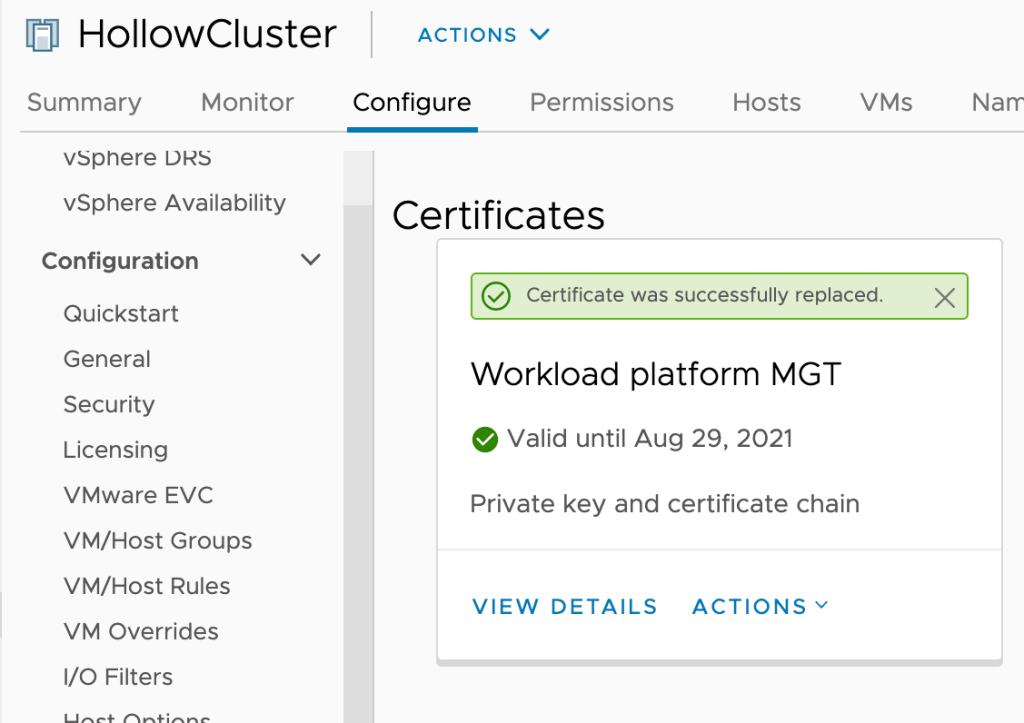
You now have a valid certificate and are ready to replace the existing VMware certificate with your own. Go to the Supervisor cluster in vCenter again, and this time select Replace Certificate from the same location where you generated the CSR.



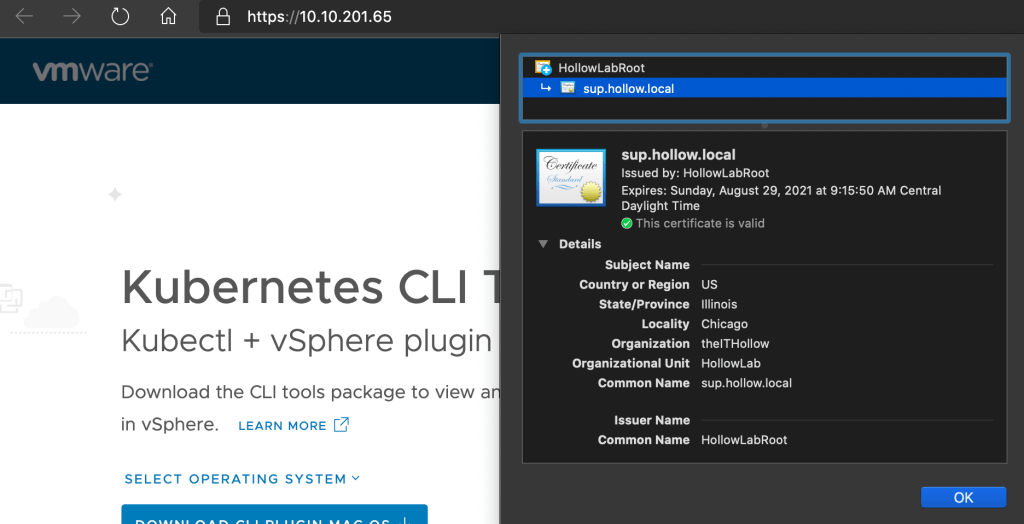
Copy the certificate, or upload from your workstation’s file system. Then click the Replace button.



When you’re done, you should see that the certificate was replaced.



To verify that everything is working the way you want it to navigate to the Kubernetes API Endpoint webpage for one of your namespaces. When opening the page, you should not get a warning about the untrusted certificate. Inspecting the certificate should show the correct chain and that it’s valid.



If you’d like to check your cli access now as well, try running a kubectl vsphere login command to see if you still need the --insecure-skip-tls-verify flag set. You shouldn’t need to use this anymore since your endpoint has a trusted certificate now.



## Summary

In this post you created your own root certificate, generated a certificate signing request from vCenter, and then signed that certificate request with your root CA. Lastly you replaced the VMware certificate with your own and tested to validate you can access the Kubernetes API Endpoint without using the insecure-skip-tls-verify option. It feels good to get rid of those certificate warnings doesn’t it?

# Create a Content Library for vSphere 7 with Tanzu

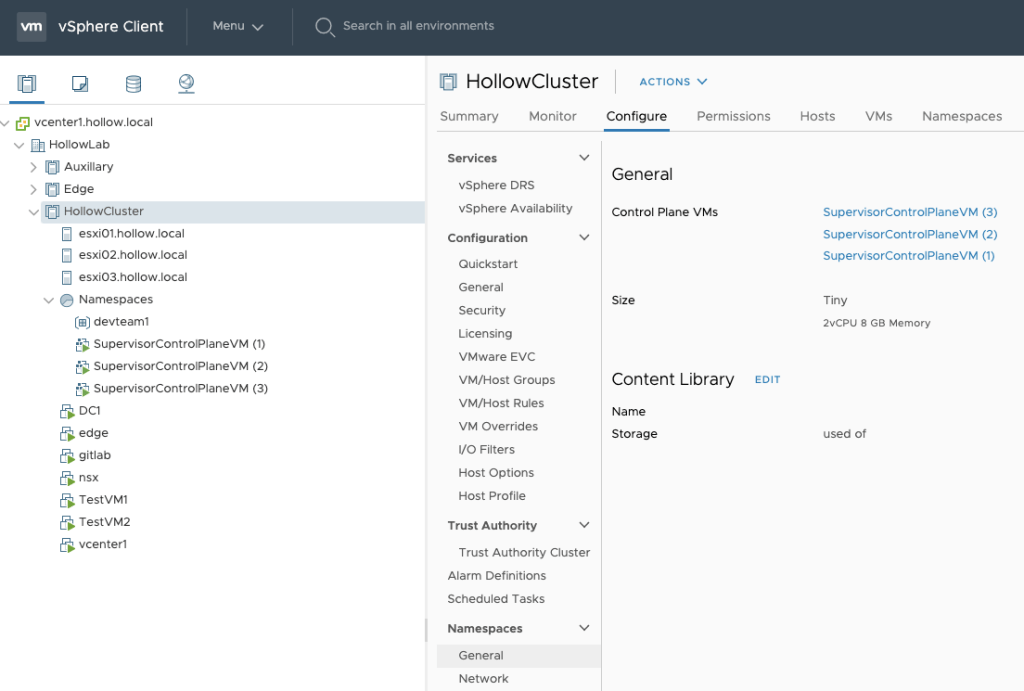
*September 8, 2020*[*1*](https://theithollow.com/2020/09/08/create-a-content-library-for-vsphere-7-with-tanzu/#comments)*By* [ERIC SHANKS](https://theithollow.com/author/eshanks/)

In this post we’ll setup a vSphere Content Library so that we can use it with our Tanzu Kubernetes Grid guest clusters. If you’re not familiar with Content libraries, you can think of them as a container registry, only for virtual machines.

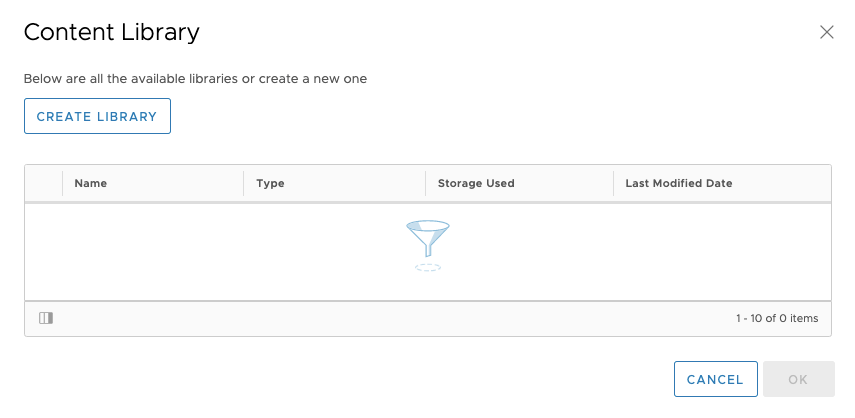
Why do we need a content library? Well, the content library be used to store the virtual machine templates that will become Kubernetes nodes when you deploy a TKG guest cluster.

## Create a Content Library

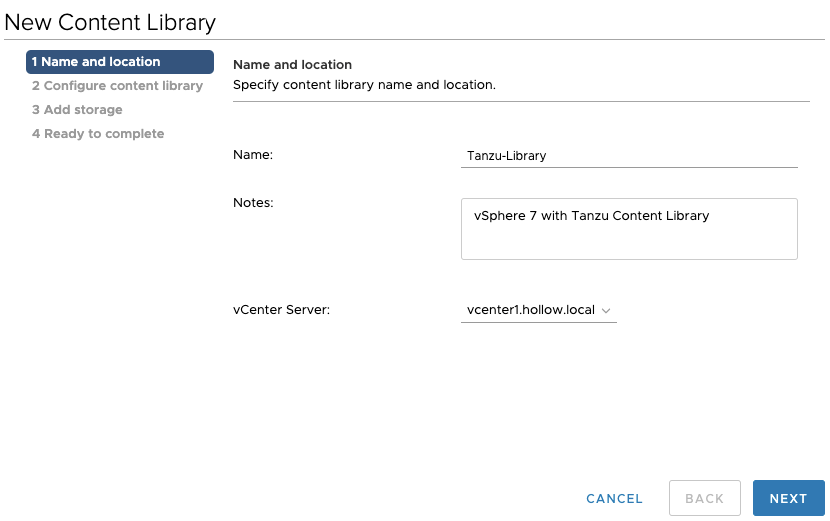
You can create content libraries by navigating to Menu –> Content Libraries or you can select your Supervisor Cluster and in the Configure menu, click General. You’ll see there, whether a content library has already been assigned to your cluster or not. Click the EDIT hyperlink to take you to the content libraries.



The screen that opens allows you to pick a content library if one has already been created. In our case, we don’t have any content libraries available so we need to create one. Luckily we can click the CREATE LIBRARY button.



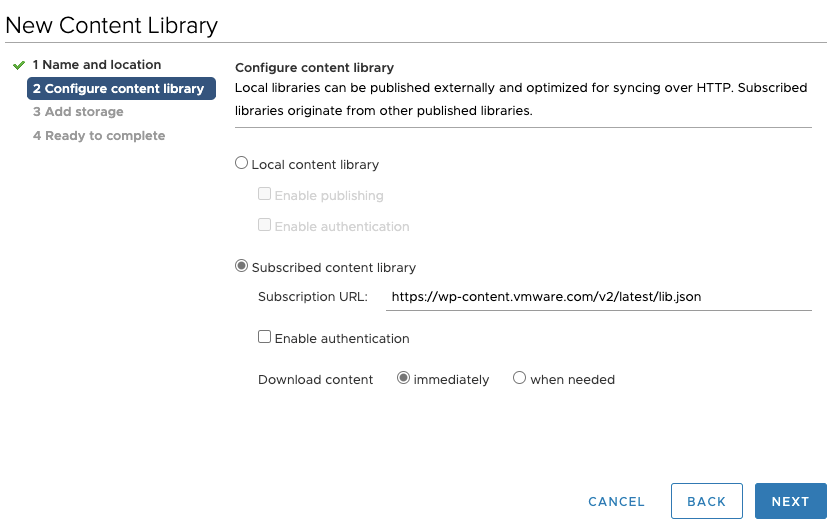
Here you’ll begin the process of creating a new Content Library through the wizard. Give the Content Library a name and any notes that you may have. Select the vCenter and then click the NEXT button.



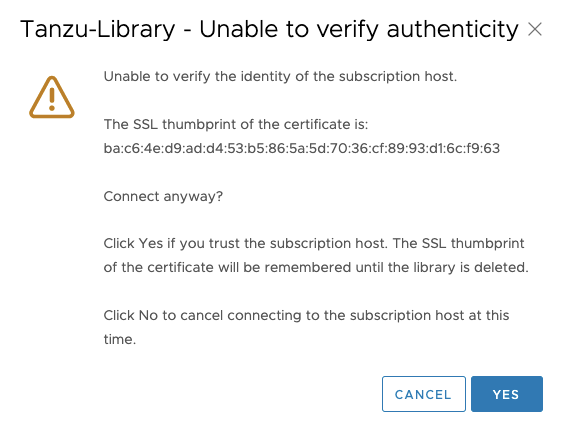
The critical screen is next. Select Subscribed content library and then enter the following subscription URL: https://wp-content.vmware.com/v2/latest/lib.json

This URL is a publicly accessible repository which has the virtual machines templates which are configured for Tanzu Kubernetes Grid. You can use other subscription URLs with content libraries, but for Tanzu you should use this URL to get the appropriate templates.

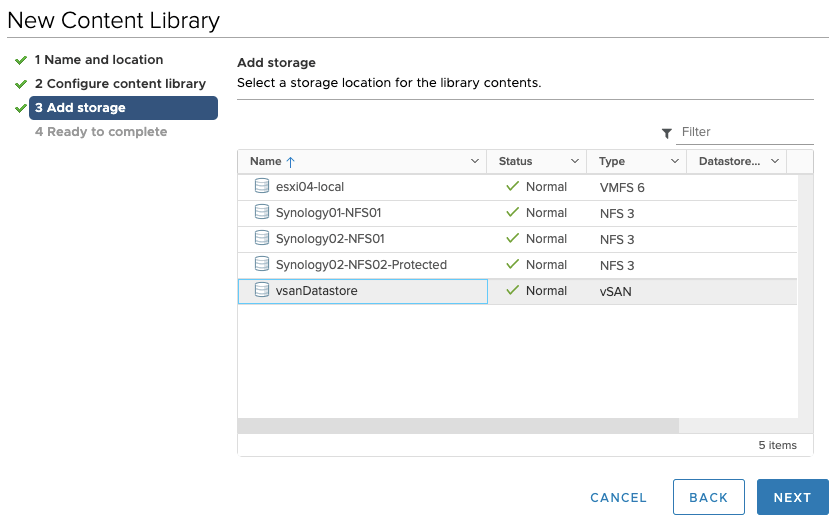
Under the Download content section, pick whether you want to immediately download the templates, or download them when they are requested to save space at the cost of the first deployment taking longer.



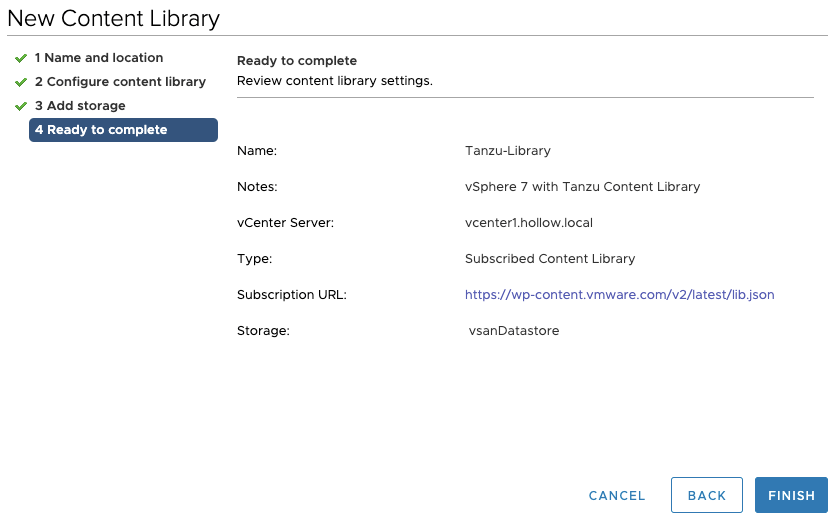
After you click next, you’ll need to accept the SSL thumbprint of the certificate by clicking the YES button.



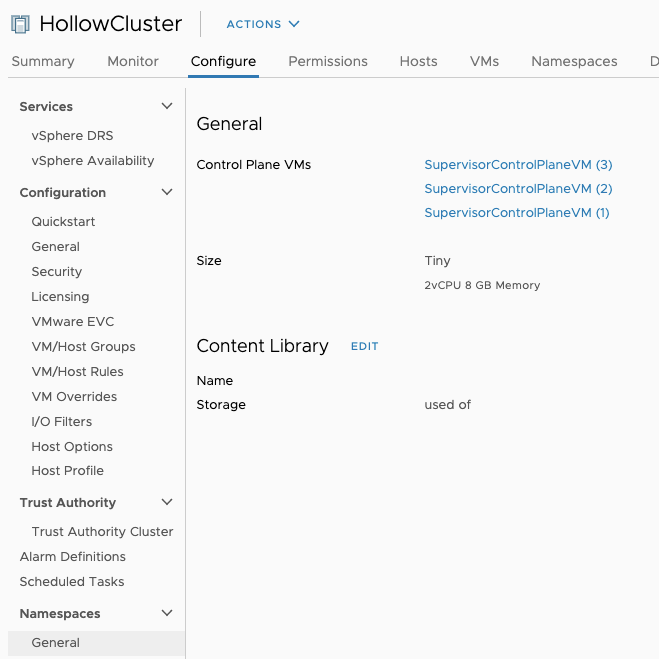
Then you’ll select the vSphere datastore which will store the virtual machine templates. Select your favorite datastore and click Next.



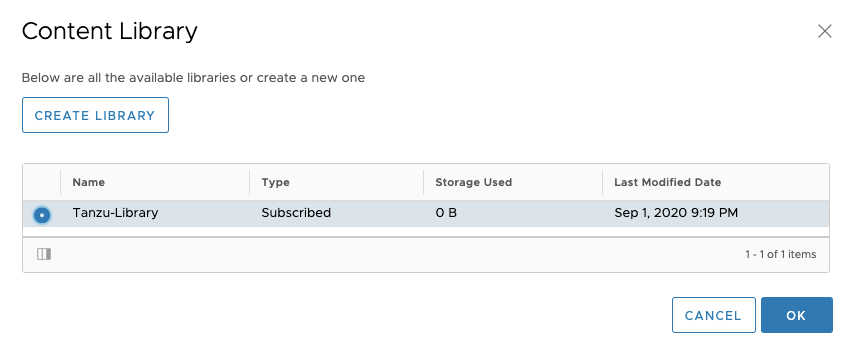
Verify the settings and click the finish button.



Lastly, go back to the supervisor cluster and click the edit button next to content library again.



Now you can select the content library created earlier. Select your content library and click OK.



## Summary

The vSphere content library isn’t new, but gives us some powerful functionality to enable building Tanzu Kubernetes Grid guest clusters. Create a library and assign it to your Supervisor cluster before moving on to the next post about building TKG guest clusters.

# Deploying Tanzu Kubernetes Clusters on vSphere 7

*September 9, 2020*[*1*](https://theithollow.com/2020/09/09/deploying-tanzu-kubernetes-clusters-on-vsphere-7/#comments)*By* [ERIC SHANKS](https://theithollow.com/author/eshanks/)

This post will focus on deploying Tanzu Kubernetes Grid (TKG) clusters in your vSphere 7 with Tanzu environment. These TKG clusters are the individual Kubernetes clusters that can be shared with teams for their development purposes.

I know what you’re thinking. Didn’t we already create a Kubernetes cluster when we setup our Supervisor cluster? The short answer is yes. However the Supervisor cluster is a unique Kubernetes cluster that probably shouldn’t be used for normal workloads. We’ll discuss this in more detail in a follow-up post. For now, let’s focus on how to create them, and later we’ll discuss when to use them vs the Supervisor cluster.

## Gather Deployment Information

These steps assume that you’ve followed [the series](https://theithollow.com/2020/07/14/vsphere-7-with-kubernetes-getting-started-guide/) so far and have configured the prerequisites such as a Supervisor Cluster, a namespace, a content library, and a user with edit permissions.

The steps to deploy a new TKG cluster consists of running a single command from the CLI.

kubectl apply -f tkgcluster.yaml

Yeah, that’s it. If you’re a Kubernetes operator, this command is going to seem very familiar! We’re deploying entire Kubernetes clusters based off of a desired state expressed by a YAML file. This means that building a TKG cluster really consists of us gathering the information we need to layout the desired state.

Let’s look at the contents of a TKG YAML file and then begin to fill in some desired state information.

apiVersion: run.tanzu.vmware.com/v1alpha1

kind: TanzuKubernetesCluster

metadata:

name: [clustername]

namespace: [namespace-name]

spec:

distribution:

version: [v1.16]

topology:

controlPlane:

count: [3]

class: [guaranteed-large]

storageClass: [tkc-storage-policy-yellow]

workers:

count: [5]

class: [guaranteed-xlarge]

storageClass: [tkc-storage-policy-green]

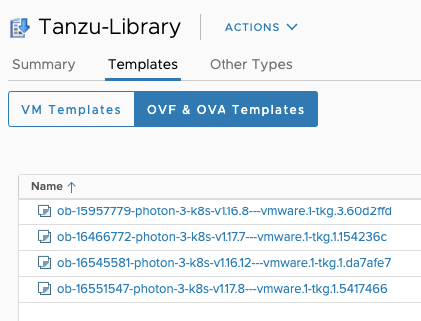
The items in brackets [] are items we need to fill in to create a new cluster. Most of these items seem pretty self explanatory. For example, how many control plane nodes and worker nodes the Kubernetes clusters should have. The clustername is completely up to you, the namespace must match the namespace deployed in the Supervisor Cluster that you have edit permissions on.

Now, lets discuss a few fields that might need a bit more explanation.

NOTE: For full descriptions of ALL fields, consult the official VMware documentation [here](https://docs.vmware.com/en/VMware-vSphere/7.0/vmware-vsphere-with-kubernetes/GUID-4E68C7F2-C948-489A-A909-C7A1F3DC545F.html).

### **Version**

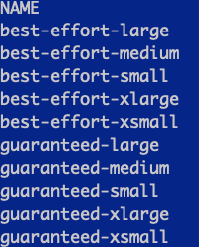
Version is the Kubernetes version that will be deployed. This is a nice feature since you can have multiple versions of Kubernetes clusters all managed from the Supervisor Cluster. A short version such as 1.16 can be used, or you can specify the image name of the kubernetes version. These image names come from the Content Library templates.



### **Class**

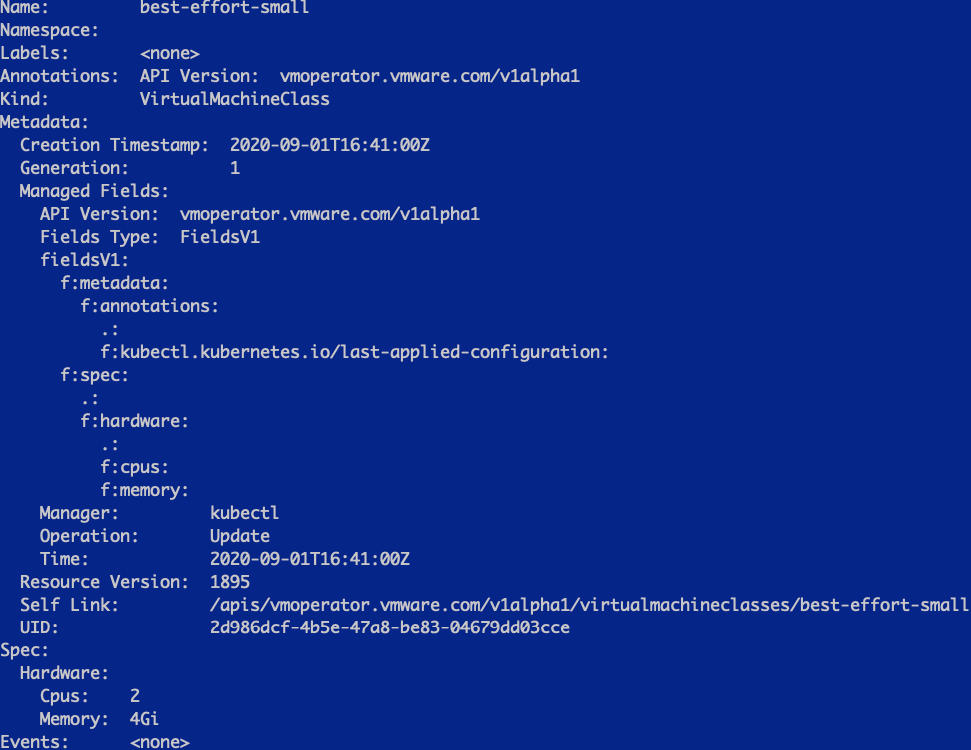
Class refers to the pre-set sizes of the nodes. You can specify different size VMs for the control plane nodes and worker nodes. So how do you know what classes are options for your cluster? Well, you could look at the official documentation [here](https://docs.vmware.com/en/VMware-vSphere/7.0/vmware-vsphere-with-kubernetes/GUID-7351EEFF-4EF0-468F-A19B-6CEA40983D3D.html). Or you could look at the resources in the supervisor cluster.

kubectl get virtualmachineclasses



If you want details about those objects, you can run a describe to find more information, just like you would in a normal Kubernetes environment.

kubectl describe virtualmachineclasses best-effort-small



### StorageClass

Storage Classes are used by Kubernetes to know how to create persistent volumes. In the TKG clusters, we specify the vSphere Storage Policy as a storageclass so that the clusters will understand how to provision persistent volumes on vSphere datastores.

The names of the storage classes can be obtained by running the following command from the Supervisor cluster namespace:

kubectl describe ns



## Create a TKG Cluster

OK, I’ve filled out my entire TKG YAML manifest and I’m ready to create my cluster. Here is the YAML I’m deploying for my cluster.

apiVersion: run.tanzu.vmware.com/v1alpha1

kind: TanzuKubernetesCluster

metadata:

name: tkg-cluster-1

namespace: devteam1

spec:

distribution:

version: v1.16

topology:

controlPlane:

count: 3

class: best-effort-small

storageClass: hollow-storage-profile

workers:

count: 5

class: best-effort-small

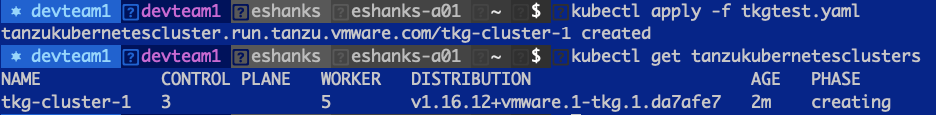
storageClass: hollow-storage-profile

After logging into the Supervisor cluster, I can simply apply the manifest to create the cluster.

kubectl apply -f [filename].yaml

Once applied, you can check the status of your cluster by running

kubectl get tanzukubernetesclusters

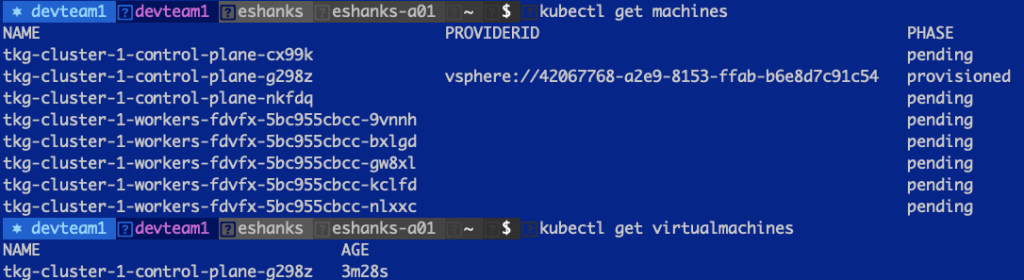


There are also a few other resources that might be checked for troubleshooting purposes. You can also list the machines and virtual machine objects.

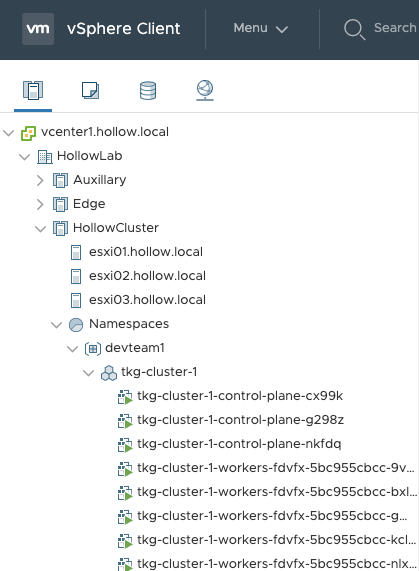
kubectl get machines

kubectl get virtualmachines

These two objects can help identify if the IaaS Provider is creating the resources correctly.



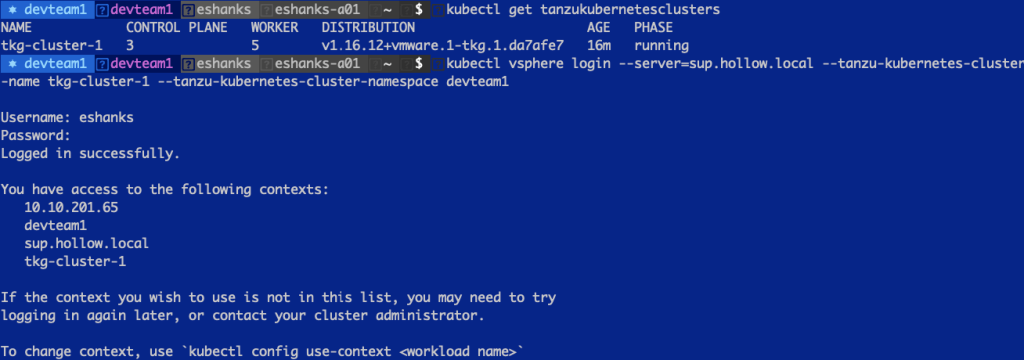
Once provisioning is over, you should see your cluster provisioned in vCenter, the get cluster command should show a provisioned cluster, and you can begin the fun work of building apps for your cluster!



## Connect to the TKG Cluster

OK, you want to know how to access your new cluster, right? Well, here you go. To login to the guest cluster you just created, run:

kubectl vsphere login --server=[SupervisorControlPlane] --tanzu-kubernetes-cluster-name [tkg cluster name] --tanzu-kubernetes-cluster-namespace [Supervisor Namespace]



Now you only need to change your context to start deploying resources on your cluster!

## Summary

In this post we gathered the appropriate information to build a desired state configuration file for our TKG guest clusters. We deployed the cluster and connected to it through the Kubectl cli and can now provision workloads.

Stay tuned for future posts where we discuss how this cluster can be updated, upgrade, modified, and destroyed.

# Enable the Harbor Registry on vSphere 7 with Tanzu

*January 4, 2021*[*0*](https://theithollow.com/2021/01/04/enable-the-harbor-registry-on-vsphere-7-with-tanzu/#comments)*By* [ERIC SHANKS](https://theithollow.com/author/eshanks/)

Your Kubernetes clusters are up and running on vSphere 7 with Tanzu and you can’t wait to get started on your first project. But before you get to that, you might want to enable the Harbor registry so that you can privately store your own container images and use them with your clusters. Luckily, in vSphere 7 with Tanzu, the Harbor project has been integrated into the solution. You just have to turn it on and set it up.

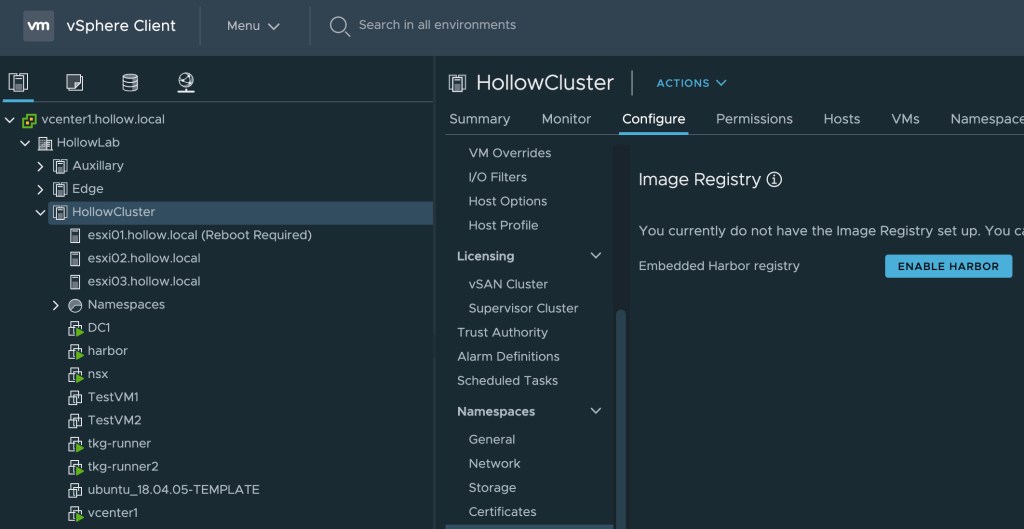
NOTE: This article takes advantage of the updates in vSphere 7 U1c. Prior to the C release, the kubernetes clusters didn’t automatically trust the harbor certificate. It’s still possible to use it, but takes additional configuration.

If you’re not familiar with what a container image registry is, in simple terms, its where you store your container images after you’ve built them. A very common and public registry is [docker hub](https://hub.docker.com/). Its pretty common to grab some public images like ubuntu, nginx, alpine, redis, etc right off of docker hub. You should, however, take care when using container images from a public repo because you aren’t really sure what code has been put into that container. Perhaps it is malicious. This is where a private image registry comes in. Place your containers with your corporate logic in these private registries, hopefully with high bandwidth connections to your clusters for faster downloads.

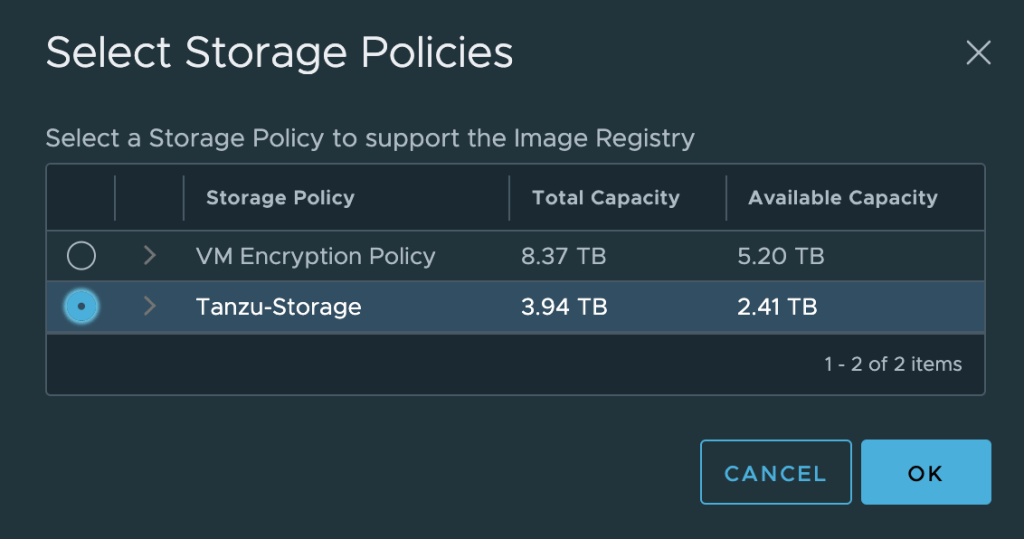
## Enable the Registry

Before you enable the harbor registry, you’ll need to have a vSphere 7 with Tanzu Supervisor cluster deployed. [This series](https://theithollow.com/2020/07/14/vsphere-7-with-kubernetes-getting-started-guide/) will help if you haven’t done this already.

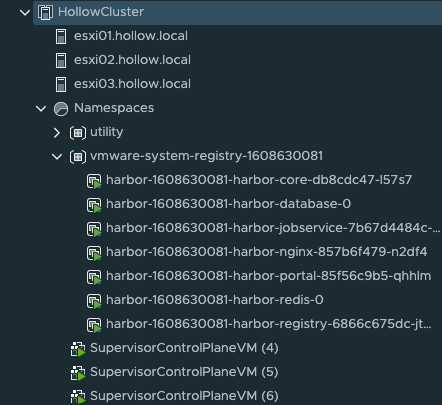
To enable the Harbor Registry, select the vSphere Virtual Machine cluster running your Supervisor cluster. Under the configure tab select Image Registry and click Enable Harbor.



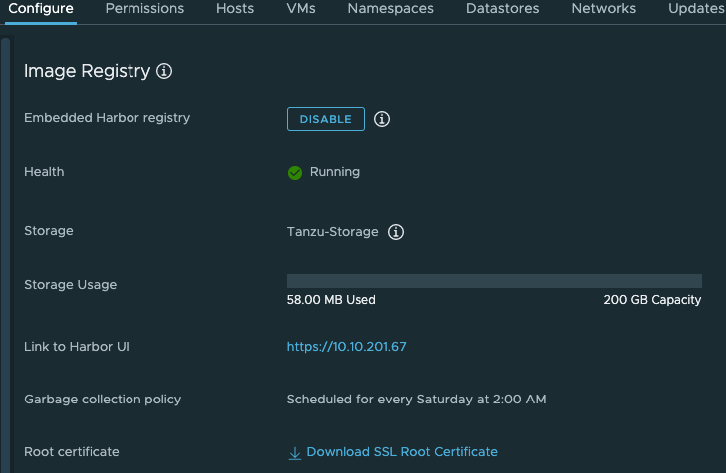
After you do this, you’ll need to specify a storage policy to decide what datastore the Harbor registry service will live within. Select your storage policy and click OK.



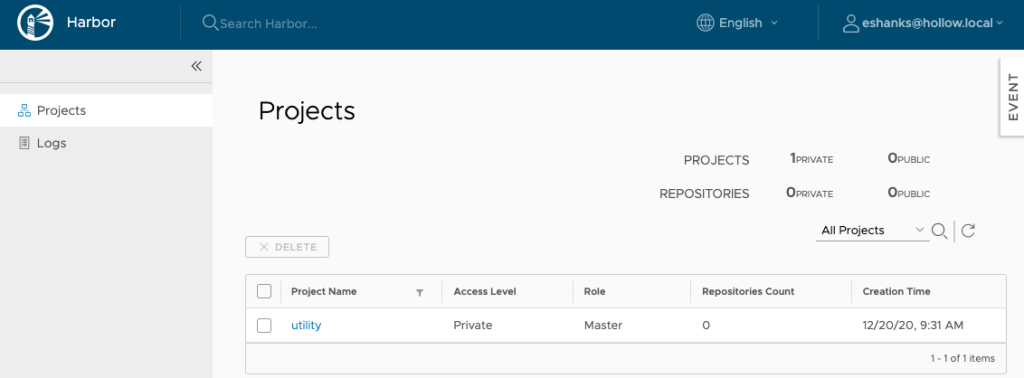
That was pretty easy huh? Well your next step is to wait for the registry to be enabled fully. If you’ve logged in as the administrator@vsphere.local user, you’ll probably see some new virtual machines being deployed in your cluster. Otherwise, these may be hidden from you.



Also, when Harbor has been deployed, you’ll get some information in the configure tab that shows the URL and the storage space used for the harbor registry.



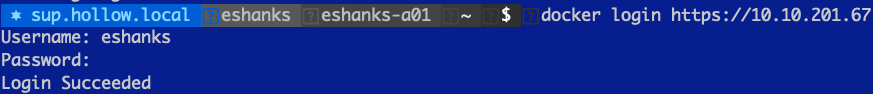
You can click the link to the Harbor UI and login with your vSphere credentials. You’ll notice that in my instance, there is already a project named utility. Thats because I have a namespace in my supervisor cluster called utility. You’ll have one project for each namespace within the supervisor cluster.



## Use Harbor with Supervisor Cluster

At this point we could start using the harbor registry for our supervisor cluster. I’ve already got a container image on my local workstation that I’ve built and plan to push it to our new harbor registry.

Before we can push anything, we need to login to the harbor registry from our workstation with a docker login command. Then login with your vSphere credentials.

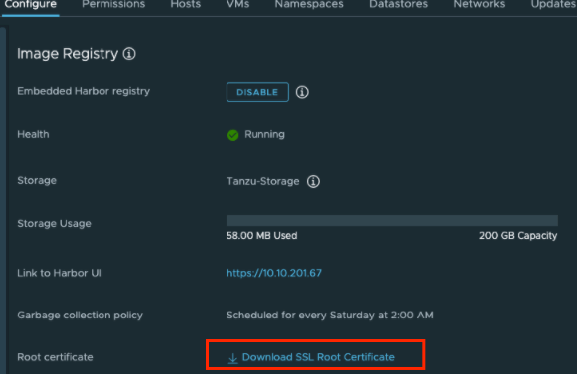


**NOTE:** If you don’t have the root certificate installed, you won’t be able to login with this method. To install the root certificiate, you can download the certificate from the harbor configuration page. You’ll need to place that certificate into your trusted store on the client you’re using. For Mac OS you can run the following commands:

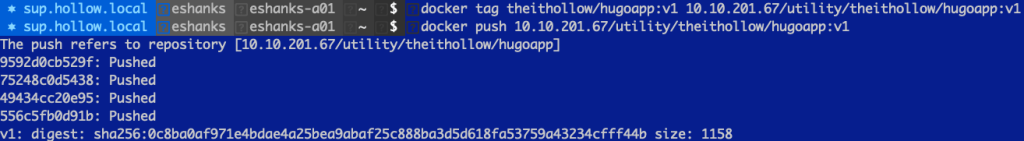
security add-trusted-cert -d -r trustRoot -k ~/Library/Keychains/login.keychain ca.crt

Code language: JavaScript (javascript)

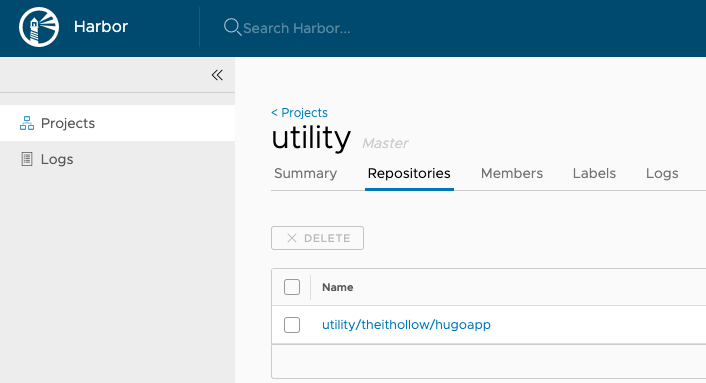
When done, you need to restart the docker service.



Once you’re logged into the registry, you can push your own image. In the screenshot below, I’ve tagged my image and pushed it to the repository.



After its been pushed, you’ll see the repository listed under your namespace in the harbor UI.



And lastly, login to your supervisor cluster within the namespace configured, and try to pull down the image and run it. You can see the image was pulled to create a new PodVM in the Supervisor cluster.



## Use Harbor with TKG Clusters

Our first example was neat, but many customers won’t run pods directly on the Supervisor cluster. If you’re building Tanzu Kubernetes Grid Clusters (referred to as TKCs, guest clusters or child clusters) you will need to take a couple of extra steps. Namely, you’ll need to obtain and deploy an image pull secret for the harbor registry on any of the child clusters.

To obtain the image pull secret, login to the Supervisor namespace.

kubectl get secret -n [vsphere-namespace] [namespace]-default-image-pull-secret -o yaml > registrysecret.yaml

Code language: CSS (css)

Edit that YAML file to remove the namespace entry. You’ll deploy this secret into the TKC namespace that will be pulling images. In my example that is the default namespace so I’ve removed the namespace: utility entry in my registrysecret.yaml file.

Next, login to your TKC cluster that was deployed within the same supervisor namespace and apply the secret.

kubectl apply -f registrysecret.yaml

Code language: CSS (css)

Once your secret has been deployed to the Tanzu Kubernetes Cluster [Child/Workload Cluster] you can add that image pull secret to your YAML manifests to start using the harbor registry for your containers.

...

spec:

containers:

- name: private-reg-container

image:

imagePullSecrets:

- name: [registrysecretname]

...

## Summary

Now that the vSphere 7.0 U1c patch has been released, you can start using the embedded Harbor registry with both your Supervisor Cluster as well as any Tanzu Kubernetes Clusters you may have provisioned. You can securely store your images here which will be very close to your workload clusters so you can expect quick downloads when new images are called for by your Kubernetes pods.

### Share this:

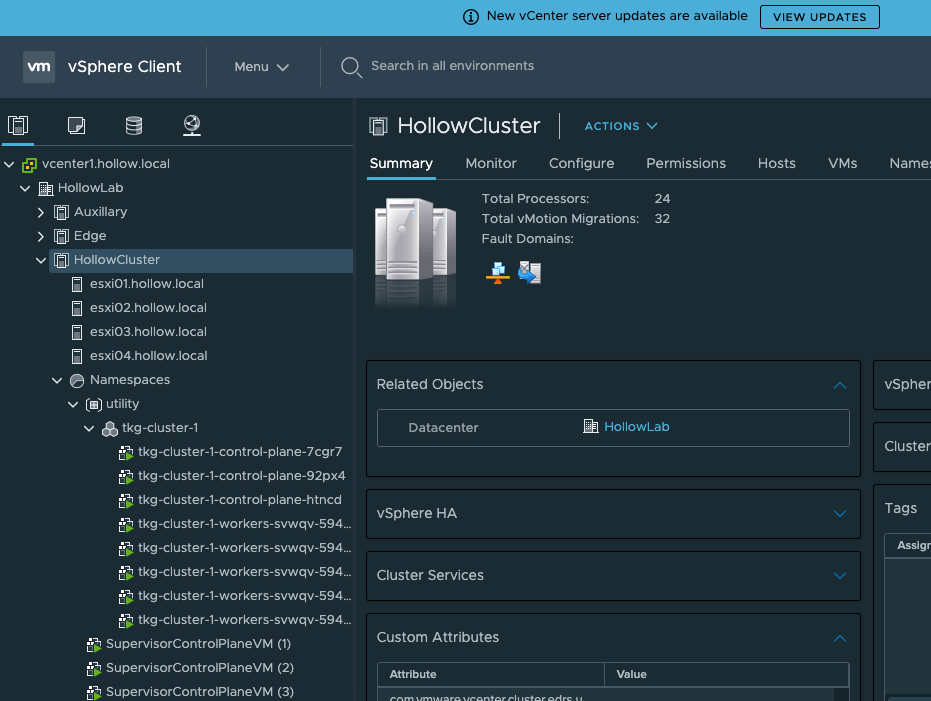
# vSphere 7 with Tanzu Updates

*May 13, 2021*[*0*](https://theithollow.com/2021/05/13/vsphere-7-with-tanzu-updates/#comments)*By* [ERIC SHANKS](https://theithollow.com/author/eshanks/)

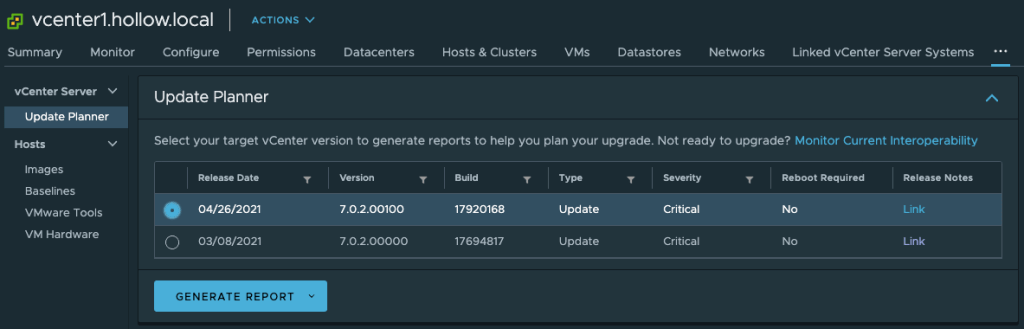
At some point, you’ll be faced with an upgrade request. New Kubernetes features, new security patches, or just to maintain your support. A vSphere 7 with Tanzu deployment has several components that may need to be updated and most of which can be updated independently of one another. In this post we’ll walk through an update to vSphere, then update the Supervisor namespace, and then finally the Tanzu Kubernetes cluster.

## vSphere Update

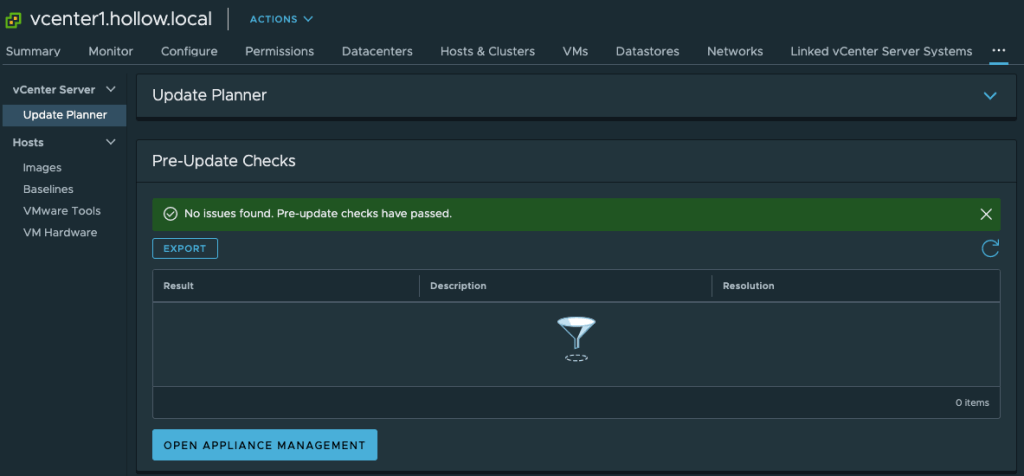
To begin we’ll start with a vSphere update. According to the vSphere update planner from the vCenter UI, I have some patches available for deployment.



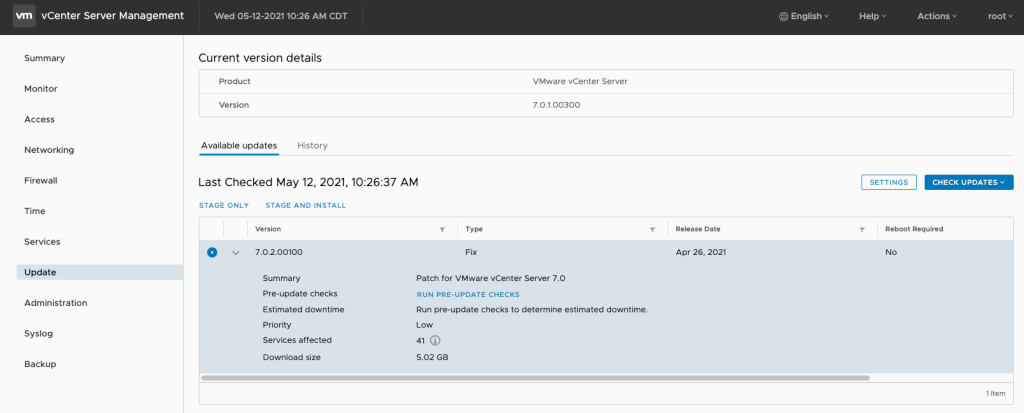
When we go to the Update Planner screen within Center, we’ll select the update that we plan to deploy and click the generate report button for a pre-update check. This makes sure the update should be able to complete successfully before we actually run the update.



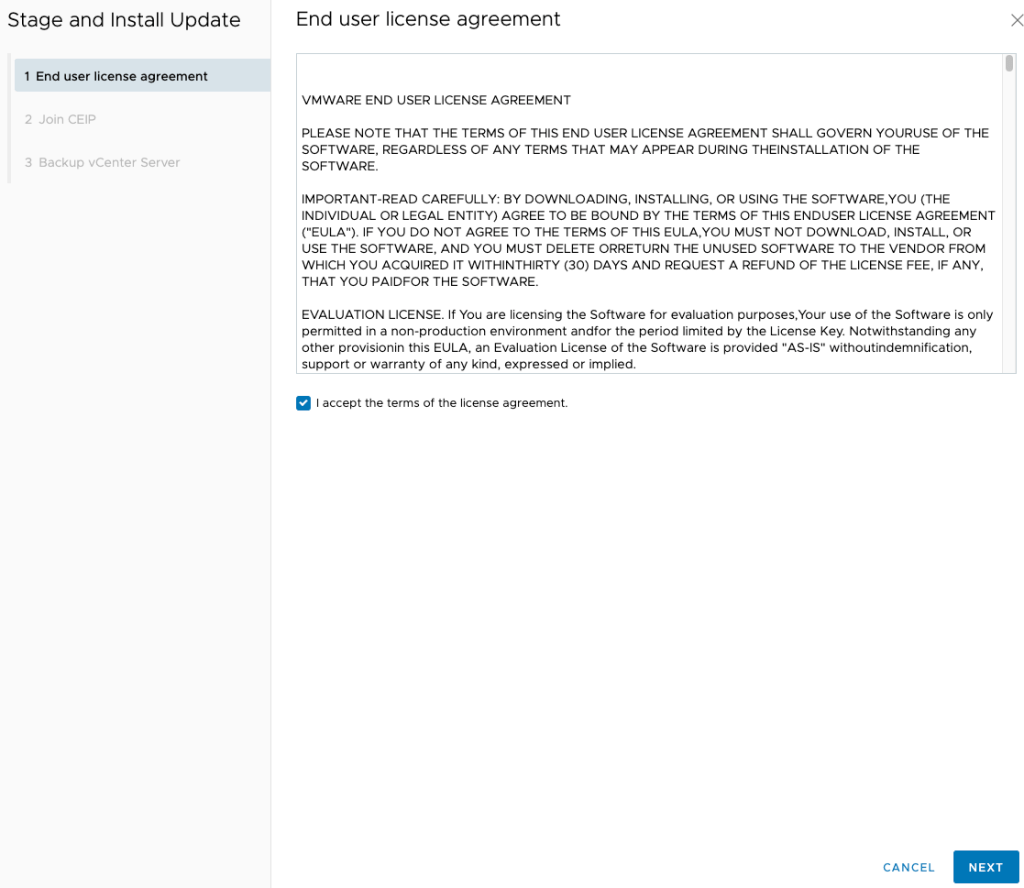
As you can see, the pre-update checks passed, so it should be safe to deploy the update. If you click the Open Appliance Management button, it will send you management UI for vCenter.



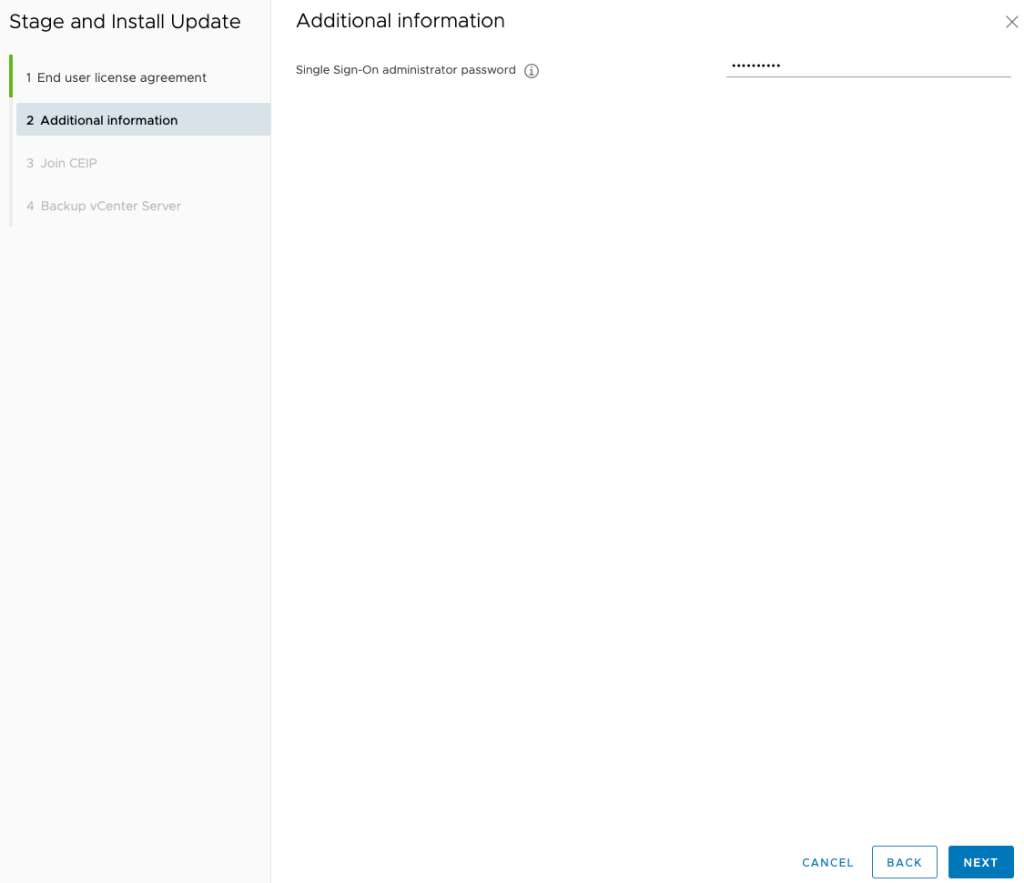
In the management UI, navigate to the Update tab. Here you should be able to check for updates, and or apply them. Here You can see that the 7.0.2 update is available and ready for deployment. I select the update and click the Stage and Install link.



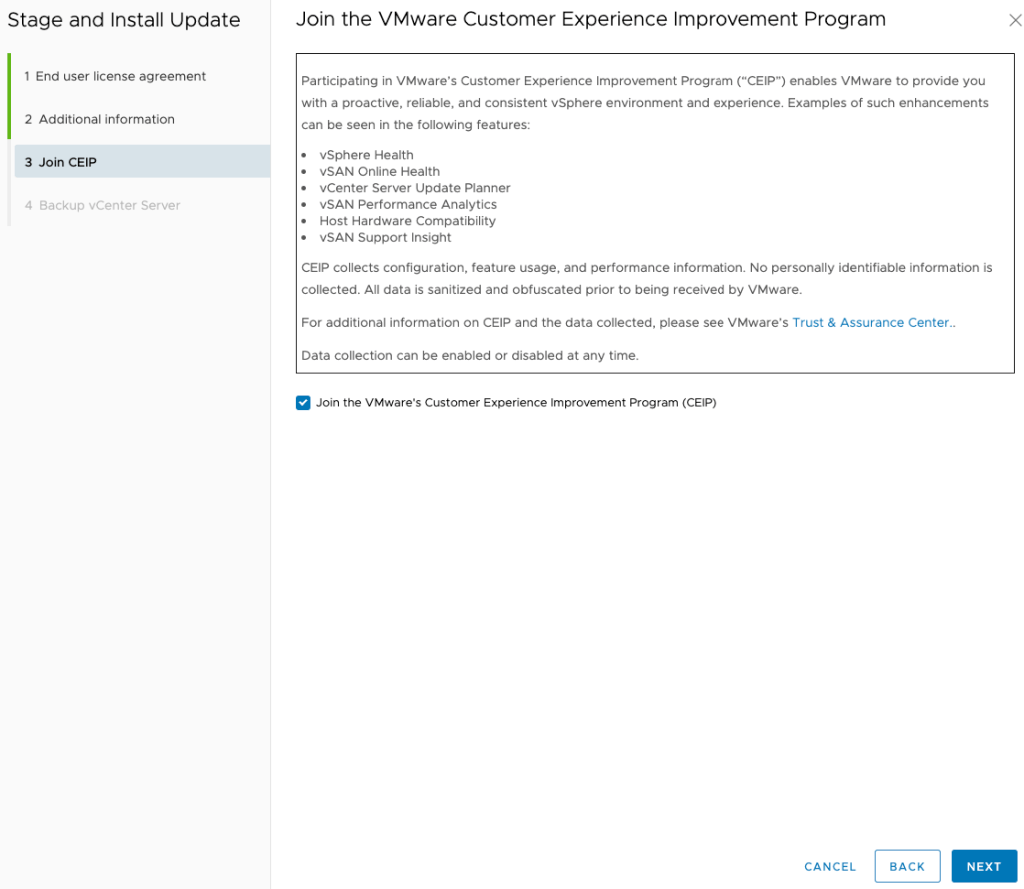
The link walks us through a few extra steps in a wizard format. First, you must accept the user agreement.



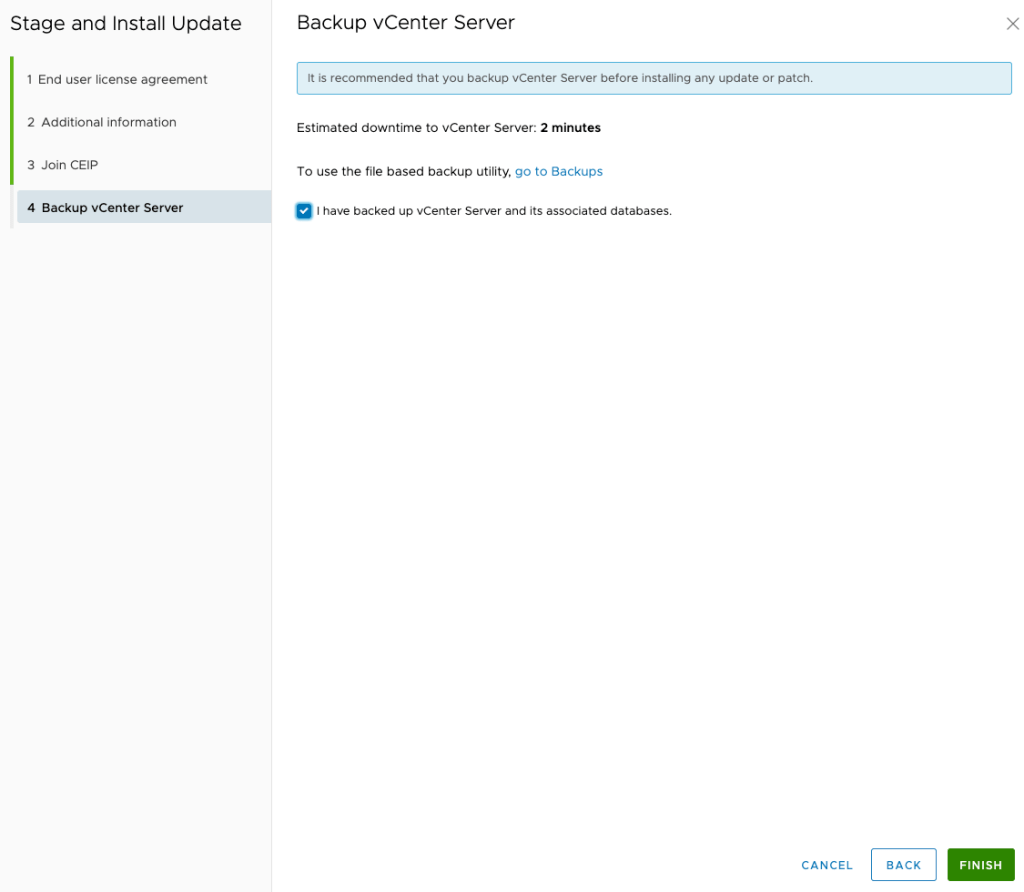
Next, you must provide the SSO password.



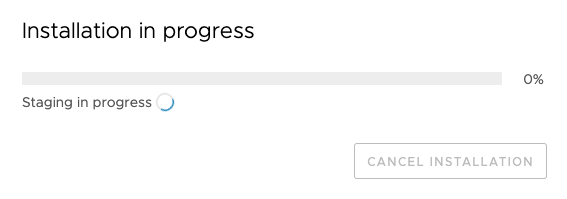
Then you need to decide whether you want to join the VMware Customer Experience Improvement Program (CEIP). This is recommended because it can provide interoperability information to you in the vCenter console, as well as helping to improve the product in future versions.



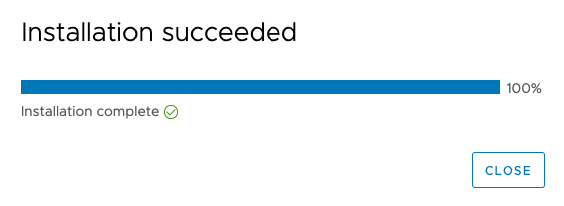
Lastly, you have to verify that you’ve backed up your vCenter. You could always lie about your backup status, but I would recommend actually taking a good backup first before these upgrades.



When done you’ll see a status message about the progress.

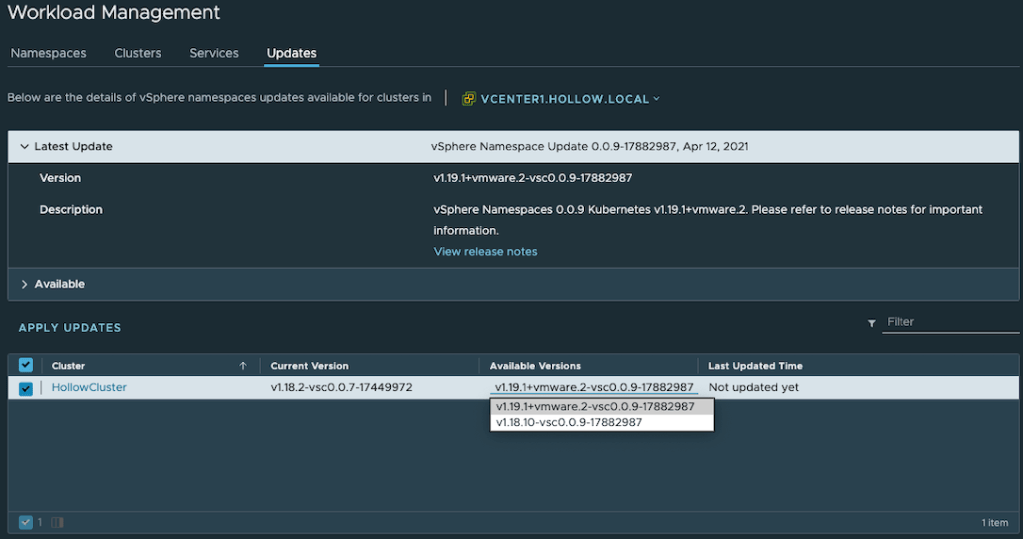


Eventually it should complete.

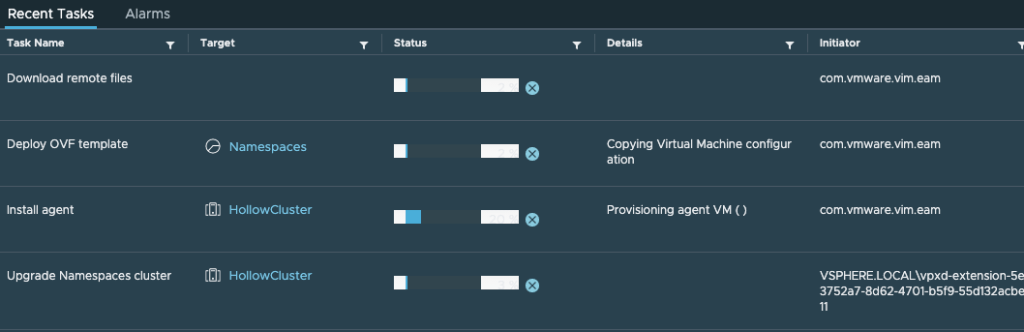


## Update Supervisor Namespaces

Once the vCenter has been updated, you can continue updating your supervisor namespaces. If you navigate to the Workload Management screen, you go to the updates tab. Here you’ll select your supervisor cluster where you’ll see the current version and will be able to select the supported available versions. In my case I’m selecting the most recent update. Click the Apply Updates link.



When you being the update process you’ll see some activity in the recent tasks window in vCenter. The update performs a rolling update to the Supervisor cluster VMs. The Supervisor cluster should remain online during these updates as one node at a time is upgraded and placed back into the cluster.



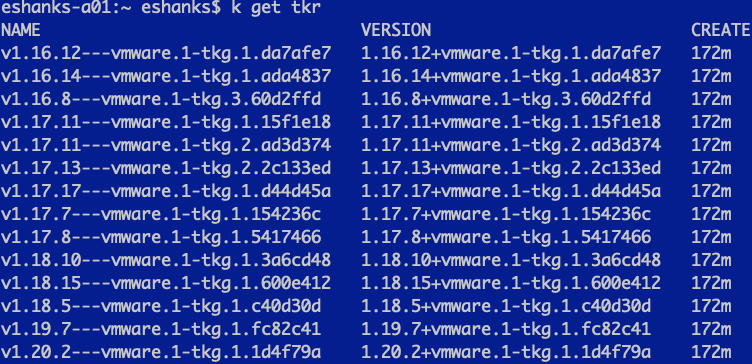
## Tanzu Kubernetes Cluster Updates

After the Supervisor namespaces have been upgraded, you can shift your attention to the Tanzu Kubernetes clusters (TKC). **NOTE:** These are often referred to as child clusters, or workload clusters.

First we login to the Supervisor namespace that contains our workload clusters. This is done through the kubectl vsphere login process covered in [other posts](https://theithollow.com/2020/09/09/deploying-tanzu-kubernetes-clusters-on-vsphere-7/). Once logged in, you can run kubectl get tkc which will list your clusters, and their versions. In the example below my cluster is running version 1.18.15.



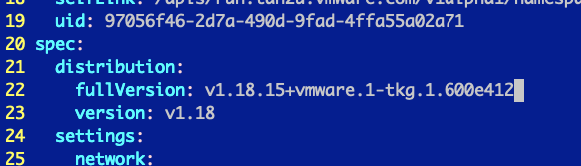
Before we can upgrade the cluster, we need to know what releases are available to us. We can do this by running kubectl get tkr. **NOTE:** that tkr stands for TanzuKubernetesReleases which can also be used in the command line instead of the short form of tkr.



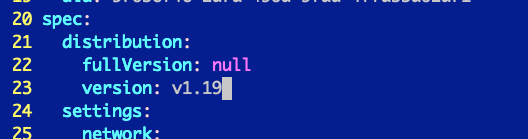
Once you’ve found the version you plan to upgrade to, it’s time to edit your cluster config to use your new version. You should be aware that you can only update minor versions and that you must upgrade them sequentially. In our case we’ll upgrade from 1.18 to 1.19 before upgrading to 1.20.

To update my cluster config we’ll run kubectl edit tkc tkg-cluster-1 and we’ll update both the “fullVersion” and the “version” spec.





For the full version, we’ll replace it with null. For the version, we’ll specify the short form version of our update. You’ll notice that after the update, the fullVersion is fully populated again. Save the config.



Once you set the version and save the config you can check the cluster status with the kubectl get tkc again and you’ll notice that the phase is in an updating state. Just like the Supervisor cluster, the workload cluster will be updated in a rolling update fashion. If you have a “Production” cluster deployed with three control plane nodes, you should see no downtime as the nodes are replaced one by one.



After the deployment is fully complete, you can re-run the process to update to any other versions. You can see that after I did my first update, I have a new column letting me know there is another version i’m eligible to upgrade to if I chose.



## Summary

Kubernetes clusters are not immune to upgrades but with a rolling update methodology, we can limit or prevent downtime to any applications running on them. vSphere with Tanzu allows you to upgrade Tanzu Kubernetes clusters independently from the Supervisor cluster, but you need to be within two minor versions to be supported. Upgrade your vCenters, then update your Supervisor Namespaces, and lastly you can update the Tanzu Kubernetes clusters.