PKS workshop

[Pivotal Ops Manager 1](#_Toc534279436)

[Goal 1](#_Toc534279437)

[Prerequisites 1](#_Toc534279438)

[Download the OVA Template: 1](#_Toc534279439)

[Upload the OVA Template to vSphere: 1](#_Toc534279440)

[Configuring PCF Ops Manager 2](#_Toc534279441)

[PCF Director 2](#_Toc534279442)

[Goal 3](#_Toc534279443)

[Prerequisites 3](#_Toc534279444)

[Enable PCF Director and (vSphere) Cloud Communication 3](#_Toc534279445)

[Complete the PCF Director Installation 5](#_Toc534279446)

[Pivotal Container Service Tile (PKS) 5](#_Toc534279447)

[Goal 5](#_Toc534279448)

[Prerequisites 5](#_Toc534279449)

[Download PKS Tile 6](#_Toc534279450)

[Upload PKS Tile to Pivotal Ops Manager 6](#_Toc534279451)

[Configure the PKS Tile 6](#_Toc534279452)

[Complete the PKS Installation 8](#_Toc534279453)

[Setting up the PKS CLI 8](#_Toc534279454)

[Goal 8](#_Toc534279455)

[Install the PKS CLI 8](#_Toc534279456)

[Show the PKS CLI options available 9](#_Toc534279457)

[Manage Kubernetes Clusters 10](#_Toc534279458)

[Goal 10](#_Toc534279459)

[Prerequisites 10](#_Toc534279460)

[Determine PKS Management API IP 10](#_Toc534279461)

[Access the deployed PKS Management API Service 10](#_Toc534279462)

[Deploy a Kubernetes Cluster 11](#_Toc534279463)

[Load Balancing a Kubernetes Cluster 12](#_Toc534279464)

[Using BOSH to show PKS Clusters 13](#_Toc534279465)

[Goal 13](#_Toc534279466)

[Prerequisites 13](#_Toc534279467)

[Access the BOSH CLI 13](#_Toc534279468)

[View BOSH Deployments 13](#_Toc534279469)

[Interacting With Kubernetes Clusters 15](#_Toc534279470)

[Goal 15](#_Toc534279471)

[Prerequisites 15](#_Toc534279472)

[Install the Kubernetes CLI - kubectl 15](#_Toc534279473)

[Authenticate to the Kubernetes Cluster 16](#_Toc534279474)

[Viewing Kubernetes Internals 17](#_Toc534279475)

[Enjoy your Kubernetes Cluster! 17](#_Toc534279476)

[PKS Pipelines 18](#_Toc534279477)

[Pre-Requisites 18](#_Toc534279478)

[(Basic) Creating PKS Clusters 19](#_Toc534279479)

[Setup the Create-Cluster Pipeline 19](#_Toc534279480)

[(Advanced) Monitor Desired PKS Clusters vs Current PKS Clusters 19](#_Toc534279481)

[Store Desired PKS State 20](#_Toc534279482)

[Setup the Monitor PKS Clusters Pipeline 20](#_Toc534279483)

[Create a New PKS Cluster with our External Configuration 21](#_Toc534279484)

[Scale and Delete Existing PKS Clusters with our External Configuration 22](#_Toc534279485)

[Using the UAA Client (UAAC) with PKS 22](#_Toc534279486)

[Pre-Requisites 22](#_Toc534279487)

[Goal 23](#_Toc534279488)

[Determine UAA Admin Credentials 23](#_Toc534279489)

[Access the UAAC CLI 23](#_Toc534279490)

[Access the deployed UAA Service 23](#_Toc534279491)

[Grant Cluster Access to a User 24](#_Toc534279492)

[Login to PKS With Newly Created User 25](#_Toc534279493)

[Setting User Roles with Kubernetes Role Bindings 25](#_Toc534279494)

[Pre-Requisites 25](#_Toc534279495)

[Goal 25](#_Toc534279496)

[Create Local UAA User - (Not required if UAA is integrated with LDAP) 25](#_Toc534279497)

[Determine UAA Admin Credentials 25](#_Toc534279498)

[Access the UAAC CLI 26](#_Toc534279499)

[Access the deployed UAA Service 26](#_Toc534279500)

[Create New Local User 27](#_Toc534279501)

[Create a Kubernetes Role and Role Binding 27](#_Toc534279502)

[View the default Kubernetes Namespaces, and Roles To Assign Our User 27](#_Toc534279503)

[Bind the Kubernetes Namespace, User, and Role together 33](#_Toc534279504)

[Login to Kubernetes as a developer (The Local User or LDAP User) 33](#_Toc534279505)

[Setup the Kubeconfig 33](#_Toc534279506)

[Use Kubernetes as a Developer! 34](#_Toc534279507)

[PKS Networking with NSX-T 35](#_Toc534279508)

[Pre-Requisites 35](#_Toc534279509)

[Goal 35](#_Toc534279510)

[Create a namespace 35](#_Toc534279511)

[View new objects with NSX-Manager 36](#_Toc534279512)

[View logical router created automatically 36](#_Toc534279513)

[Kubernetes network policy and microsegmentation 36](#_Toc534279514)

[Create a network policy 37](#_Toc534279515)

[View firewall rules created automatically 39](#_Toc534279516)

[Traceflow 41](#_Toc534279517)

[Configure Traceflow source 41](#_Toc534279518)

[Configure Traceflow destination 42](#_Toc534279519)

[Verify packets are dropped 43](#_Toc534279520)

[Remove network policy 43](#_Toc534279521)

[Re-trace your application 44](#_Toc534279522)

[Cleanup deployments 44](#_Toc534279523)

[Harbor (Container Registry) 45](#_Toc534279524)

[Goal 45](#_Toc534279525)

[Prerequisites 45](#_Toc534279526)

[Download Harbor Tile 45](#_Toc534279527)

[Upload PKS Tile to Pivotal Ops Manager 45](#_Toc534279528)

[Configure the Harbor Tile 45](#_Toc534279529)

[Configure Harbor DNS 46](#_Toc534279530)

[Repository Management 46](#_Toc534279531)

[Prerequisites 46](#_Toc534279532)

[Goal 47](#_Toc534279533)

[Login to Harbor UI 47](#_Toc534279534)

[View Projects and Repositories 47](#_Toc534279535)

[Vulnerability Scans 49](#_Toc534279536)

[Configuration Options 49](#_Toc534279537)

**Pivotal Ops Manager**

**Goal**

Bring up the PCF Ops Manager VM within vCenter. This will be the operational console we will use to deploy PKS.

**Prerequisites**

* VMware vCenter
* Ability to upload \*.OVA template to vCenter
* IP to assign to the PCF Ops Manager with DNS hostname.
  + Gateway, DNS, and Netmask corresponding to this IP.
* NTP Server

**Download the OVA Template:**

1. Point your browser to https://network.pivotal.io/
2. Create an Account and Login
3. Scroll down and under the Pivotal Cloud Foundry section click Pivotal Cloud Foundry Operations Manager
4. Under Release Download Files click on the one that reads Pivotal Cloud Foundry Ops Manager for vSphere.
5. Accept the EULA
6. **Ensure** the release is 2.0 or higher!

**Upload the OVA Template to vSphere:**

1. Login into your vSphere Web Client
2. Using the hosts and clusters view right click on your vCenter
3. Click Deploy OVF Template
4. Click Browse... and navigate to where the OVA downloaded earlier is located
5. Select a Datacenter where the VM will live
6. Select a cluster where the VM will live
7. Review the details and click next
8. Select a datastore where the VM will live
9. Select a network where the VM will live
10. Fill in the specific values for PCF Ops Manager as defined below and click next followed by finish!
    * **Admin Password:** This password is used to SSH into the Ops Manager VM we are deploying. The username is always ubuntu
    * **Custom Hostname:** If you would like to set a custom hostname for the PCF Ops Manager do so here, this hostname should resolve to the PCF Ops Manager IP.
    * **DNS:** The DNS server that can resolve hostnames within your datacenter.
    * **Default Gateway:** When we are attempting to communicate with an IP that is out of our local subnet we will go to a router for directions. Put that router IP here.
    * **IP Address:** The IP address for the Ops Manager VM.
    * **NTP Servers:** A valid NTP server to keep time for the Ops Manager VM.
    * **Netmask:** The corresponding netmask for our subnet.
11. Once fully uploaded to vSphere, power on the VM.

**Configuring PCF Ops Manager**

1. Navigate to https://<HOSTNAME-SPECIFIED-EARLIER>. If you do not have DNS setup to resolve the hostname to the Ops Manager VM given earlier when deploying the OVA, directly utilize the IP specified earlier, to access Ops Manager.
2. You might need to accept the SSL cert if it is self-signed.
3. Depending on your needs select the correct type of Authentication System, we will choose Internal Authentication for the reminder of this guide. Under the covers PCF Ops Manager deploys PCF UAA which manages access to the PCF ecosystem. The type of authentication system you choose defines how UAA will retrieve and store users that can access the PCF Ops Manager.

More info below:

* + **Use an Identity Provider:** Links together PCF Ops Manager with an external Identity Provider which maintains the user database to access the PCF Ops Manager. SAML is an example of an IdP type, and Okta is an example of a IdP itself.
  + **Internal Authentication:** PCF UAA will create and maintain a new user database for use with PCF Ops Manager.

1. Set username and password. These will be used to login to the PCF Ops Manager **GUI**.
   * *Remember ssh credentials were specified earlier!*
2. Set the Decryption passphrase. This is used to encrypt the database backing our authentication service PCF UAA.
3. Set Http Proxy, Https Proxy, and No proxy depending on *if* you need to use a proxy to access the internet from your datacenter.
4. Agree to the terms and click finish!

**PCF Director**

**Goal**

Configure and deploy PCF Director Tile within PCF Ops Manager. This will be our base to build PKS on.

**Prerequisites**

* VMware vCenter
  + Admin Username/Password
  + Datacenter Name
  + Cluster Name
  + Datastore Name
  + Network Name
* IP Range of (around) 30 IPs converted to CIDR to assign to PKS. ( [CIDR Calculator](http://www.subnet-calculator.com/cidr.php) )
  + Gateway, DNS, and Netmask corresponding to this range.
* NTP Server
* Deployed Pivotal Ops Manager

**Enable PCF Director and (vSphere) Cloud Communication**

1. Navigate to the Ops Manager FQDN (hostname).
2. Login with the Ops Manager credentials setup earlier.
3. Click on Director Tile (vSphere). PCF deploys and manages disks, networks, and vms. In order to do this, it needs access to our IaaS. This is where we will configure how it manages the IaaS.
4. On the vCenter Config page fill in the following information:
   * **vCenter Host:** The hostname of the vCenter that manages the ESXi/vSphere cluster.
   * **vCenter Username:** A vCenter username with create and delete privileges for virtual machines (VMs), disks, and folders.
   * **vCenter Password:** The password for the vCenter user specified above.
   * **Datacenter Name:** The name of the datacenter as it appears in vCenter.
   * **Virtual Disk Type:** The Virtual Disk Type to provision for all VMs.
   * **Ephemeral Datastore Names (comma delimited):** The names of the datastores that store ephemeral VM disks deployed by Ops Manager.
   * **Persistent Datastore Names (comma delimited):** The names of the datastores that store persistent VM disks deployed by Ops Manager.
   * **Networking Type:** We will use Standard vCenter Networking for this guide. NSX can also be used.
   * **VM Folder:** The vSphere datacenter folder (default: pcf\_vms) where Ops Manager places VMs.
   * **Template Folder:** The vSphere datacenter folder (default: pcf\_templates) where Ops Manager places VM templates.
   * **Disk path Folder:** The vSphere datastore folder (default: pcf\_disk) where Ops Manager creates and stores disk images. You must not nest this folder.
   * *Note: After your initial deployment, you will not be able to edit the VM Folder, Template Folder, and Disk path Folder names.*
5. On the Director Config page fill in the following:
   * **NTP Servers:** A valid NTP server(s) to keep time for the PCF Director.
   * ***Ensure*** **Enable Post Deploy Scripts** is checked.
6. On the Create Availability Zones page we create multiple Availability Zones which allow us to provide high-availability and load balancing to our containers deployed to PKS. At least three availability zones are recommended for a highly available installation of PKS.

To add **Availability Zones** run the following:

* + Click Add.
  + Enter a unique Name for the Availability Zone.
  + Enter the name of an existing vCenter Cluster to use as an Availability Zone.
  + (Optional) Enter the name of a Resource Pool in the vCenter cluster that you specified above. The jobs running in this Availability Zone share the CPU and memory resources defined by the pool.

1. On the Create Networks page we create virtual networks which allows the PCF Director to deploy VMs in specific networks. We will create 2 networks, one for the management of PKS, and one for the kubernetes clusters themselves.

To add **Networks** run the following:

* + Select Enable ICMP checks to enable ICMP on your networks. Ops Manager uses ICMP checks to confirm that components within your network are reachable.
  + PKS deploys Kubernetes Clusters on demand via a CLI command. The PKS API Manager itself requires a single VM which will listen for CLI commands and then deploy the cluster. This VM will be deployed in the network we create below:
    - Click Add Network.
    - Enter PKS-Management for the network.
    - Click Add Subnet to create one or more subnets for the network.
    - Enter the full path and name of the vSphere Network Switch we will use to connect VMs to. For example, VM Network. If your vSphere Network Name contains a forward slash character, replace the forward slash with the URL-encoded forward slash character %2F. For example, YOUR-DIRECTORY-NAME%2FYOUR-NETWORK-NAME.
    - For CIDR, enter a valid CIDR block in which to deploy VMs. For example, enter 192.0.2.0/24. **This should be a range of IPs that can be used for the management plane of PKS. Only 10 IPs will be needed for this development env.**
    - For Reserved IP Ranges, enter any IP addresses from the CIDR that you want to blacklist from the installation. PKS will not deploy any VM to any address in this range.
    - Enter your DNS and Gateway IP addresses for this network.
    - Select which Availability Zones to use with the network.
  + PKS deploys Kubernetes Clusters on demand via a CLI command. Those clusters will be deployed in the network we create below:
    - Click Add Network.
    - Enter PKS-Clusters for the network.
    - In order to dynamically provision PKS clusters in this network, select the Service Networks checkbox.
    - Click Add Subnet to create one or more subnets for the network.
    - Enter the full path and name of the vSphere Network Switch we will use to connect VMs to. For example, VM Network. If your vSphere Network Name contains a forward slash character, replace the forward slash with the URL-encoded forward slash character %2F. For example, YOUR-DIRECTORY-NAME%2FYOUR-NETWORK-NAME.
    - For CIDR, enter a valid CIDR block in which to deploy VMs. For example, enter 192.0.2.0/24. **This should be a range of IPs that can be used for all of the PKS clusters. EACH Kubernetes cluster you want to deploy will use at least 2 IPs.**
    - For Reserved IP Ranges, enter any IP addresses from the CIDR that you want to blacklist from the installation. PKS will not deploy any VM to any address in this range.
    - Enter your DNS and Gateway IP addresses.
    - Select which Availability Zones to use with the network.
  + Click Save.

1. On the Assign AZs and Networks page we decide to which network and AZ PCF Director should be deployed to as follows:
   * Since the Director is a single VM, use the drop-down menu to select a Singleton Availability Zone for it.
   * Use the drop-down menu to select a Network for your PCF Ops Manager Director. We will use PKS-Management.
   * Click Save.

**Complete the PCF Director Installation**

1. Click the Installation Dashboard link to return to the Installation Dashboard.
   * *Notice the tile has changed form orange to green signifying it it ready to be applied!*
2. Click "Apply Changes" on the right navigation to begin deploying!

**Pivotal Container Service Tile (PKS)**

**Goal**

Configure and deploy the PKS Tile within PCF Ops Manager.

**Prerequisites**

* Deployed Pivotal Ops Manager (2.0+)
* Deployed Pivotal Director Tile
  + Post-deploy scripts enabled
* VMware vCenter
  + Admin Username/Password
  + Datacenter Name
  + Cluster Name
  + Datastore Name

**Download PKS Tile**

1. Navigate to [Pivotal Network](https://network.pivotal.io/)
2. Register and login.
3. Download the [Pivotal Container Service (PKS) Tile](https://network.pivotal.io/products/pivotal-container-service)
4. Accept the EULA

**Upload PKS Tile to Pivotal Ops Manager**

1. Navigate to the Pivotal Ops Manager FQDN (hostname).
2. Login.
3. Select Import a Product.
4. Browse to the product downloaded earlier.
5. Wait while the product is uploaded, this may take time depending on your connection to the Pivotal Ops Manager. *When completed notice the new product in the side panel of Pivotal Ops Manager.*

**Configure the PKS Tile**

1. Click the green + on the product tile.
2. Click the Pivotal Container Service tile. PKS deploys kubernetes clusters on demand, where those clusters will live, and what features are enabled in those clusters will be defined in this tile.
3. Configure the Assign AZs and Networks page.
   * Select an AZ for both the singleton jobs, and all other jobs to run in.
   * Select PKS-Management for network if having followed earlier demos. This network is where the PKS Management API itself will live.
   * Select PKS-Clusters for service network if having followed earlier demos. This network is where PKS will provision Kubernetes Clusters on demand.
   * Click Save.
4. Configure the PKS API page.
   * Either generate a self signed certificate for our PKS API endpoint or copy/paste in the certificate PEM and private key PEM. This certificate will be used when connecting to the PKS management API endpoint. We will use this endpoint for deploying PKS clusters.
     + The certificate must be valid for \*.pks.<INSERT DOMAIN HERE>.
   * Click Save.
5. Configure the Plan 1 - Plan 3 pages. These plans will define the type of Kubernetes clusters available for on demand deployment.

For each plan:

* + Select active or inactive. At least one is required to be active.
  + Fill in the Name and Description fields for each active plan.
  + Select the AZ that the cluster will be deployed to.
  + Select RBAC for Authorization Mode.
  + Adjust the ETCD/Master VM Type depending on what you require for the plan.

This will apply to **ALL** kubernetes clusters deployed with PKS and this plan

* + Adjust the Master Disk Type depending on what you require for the plan.

This will apply to **ALL** kubernetes clusters deployed with PKS and this plan

* + Adjust the Worker VM Type depending on what you require for the plan.

This will apply to **ALL** kubernetes clusters deployed with PKS and this plan

* + Adjust the Worker Persistent Disk Type depending on what you require for the plan.

This will apply to **ALL** kubernetes clusters deployed with PKS and this plan

* + Adjust the Worker Node Instances depending on what you require for the plan.

This can be changed on the fly when deploying kubernetes clusters with the PKS CLI

* + Use the Add-Ons box to define what a cluster includes out of the box. This will add any additional deployments to the deployed Kubernetes cluster in the system namespace. For example, this could be a pod that gathers metrics or does logging for your cluster. This should be in a yaml format. You can specify multiple files using --- as a separator.
  + If you want users to be able to create pods with privileged containers, select the Enable Privileged Containers - Use with caution option.
  + Click Save

1. Configure the Kubernetes Cloud Provider page. This section will configure where the deployed Kubernetes clusters will store any persistent disks.
   * Choose vSphere.
   * Fill in the corresponding values:

vCenter Credentials

vCenter Host

Datacenter Name

Datastore Name

* + For the VM Folder we must tell PKS where our Kubernetes Clusters are currently deployed. To retrieve the name of the folder, navigate to your Pivotal Ops Manager Director tile, click vCenter Config, and locate the value for VM Folder. The default folder name is pcf\_vms.
  + Click Save.

1. Configure the Networking page. This section will configure how routing in Kubernetes is performed.
   * Select Flannel.
   * Click Save.
2. Configure the UAA page.
   * Enter api.pks.<INSERT DOMAIN HERE> for the UAA URL. The UAA (User Authentication & Authorization) service runs on the same VM that the PKS API will run on. In order to authenticate with it we must give it a valid hostname. *UAA will run on port 8443*
   * Click Save.
3. Configure the Errands page. Errands are scripts that run at designated points during an installation.
   * Enable the Upgrade all clusters errand.

Because PKS uses floating stemcells, updating the PKS tile with a new stemcell triggers the rolling of every VM in all Kubernetes clusters. Also, updating other product tiles in your deployment with a new stemcell causes the PKS tile to roll VMs. This rolling is enabled by the Upgrade all clusters errand. Pivotal recommends that you keep this errand turned on because automatic rolling of VMs ensures that all deployed cluster VMs are patched. However, automatic rolling can cause downtime in your deployment.

**Complete the PKS Installation**

1. Click the Installation Dashboard link to return to the Installation Dashboard.
   * *Notice the tile has changed form orange to green signifying it it ready to be applied!*
2. Click Apply Changes on the right navigation.

**Setting up the PKS CLI**

**Goal**

Install the PKS CLI and understand the options available to us via the CLI.

**Install the PKS CLI**

1. Navigate to https://network.pivotal.io/products/pivotal-container-service
2. Select PKS CLI
3. Download the correct CLI version for your operating system.
4. Add the binary to your PATH using the following instructions depending on your Operating System.
   * Mac OS X

Make the pks binary executable:

$ chmod +x ~/Downloads/pks-darwin\*

Move the binary in to your PATH.

$ sudo mv ~/Downloads/pks-darwin /usr/local/bin/pks

* + Windows
    - Rename the binary to pks.exe
    - Add the binary in to your PATH.
      1. Right click Computer, Select Properties
      2. Select Advanced Tab
      3. Select Environment Variables
      4. select PATH
      5. Select edit
      6. add ;<PATH-TO-PKS-BINARY>
  + Linux

Make the pks binary executable.

$ chmod +x ~/Download/pks-darwin\*

Move the binary in to your PATH.

$ sudo mv ~/Download/pks-darwin\* /usr/local/bin/pks

**Show the PKS CLI options available**

Use the CLI to show you the options available to us!

$ pks

This should return something similar to the following:

The Pivotal Container Service (PKS) CLI is used to **create**, manage, **and** **delete** Kubernetes clusters. **To** deploy workloads **to** a Kubernetes cluster created **using** the PKS CLI, **use** the Kubernetes CLI, kubectl.

**Version**: 1.0.0-**build**.3

Note: The PKS CLI **is** **under** development, **and** **is** subject **to** **change** **at** **any** time.

**Usage**:

pks [command]

Available Commands:

cluster **View** the details **of** the cluster

clusters **Show** all clusters created **with** PKS

**create**-cluster Creates a kubernetes cluster, requires cluster **name** **and** an **external** host **name**

**delete**-cluster Deletes a kubernetes cluster, requires cluster **name**

**get**-credentials Allows you **to** **connect** **to** a cluster **and** **use** kubectl

**help** **Help** about **any** command

login Login **to** PKS

logout **Logs** **user** **out** **of** the PKS API

plans **View** the preconfigured plans available

**resize** Increases the **number** **of** worker nodes **for** a cluster

Flags:

-h, *--help help for pks*

*--version version for pks*

**Use** "pks [command] --help" **for** more information about a command.

**Manage Kubernetes Clusters**

**Goal**

Deploy a Kubernetes cluster, and setup our host to connect to it.

**Prerequisites**

* Deployed Pivotal Ops Manager (2.0+)
* Deployed Pivotal Director Tile
  + Post-deploy scripts enabled
* Deployed Pivotal Container Service Tile
* Installed PKS CLI
* UAA User Created with PKS create permissions

**Determine PKS Management API IP**

In order to reach the PKS Management API service we must ensure either our DNS or Load Balancer is sending us to the correct PKS Management API url. We cannot utilize IPs due to the default security enforcement built into the PKS Management API. **Skip this step if you deployed a load balancer and/or registered the PKS Management API hostname with your DNS**

1. Navigate to the Ops Manager FQDN (hostname).
2. Login with the Ops Manager credentials setup earlier.
3. Click on Pivotal Container Service Tile.
4. Click on Status.
5. Note the IP for the Pivotal Container Service job down.

**Access the deployed PKS Management API Service**

In order to reach the PKS Management API service we must ensure either our DNS or Load Balancer is sending us to the correct PKS Management API url. We cannot utilize IPs due to the default security enforcement built into the PKS Management API. **Skip this step if you deployed a load balancer and/or registered the PKS Management API hostname with your DNS**

1. In order to have our ***Local*** DNS system resolve the PKS Management API hostname to the correct IP lets edit the local host resolution file. Replace the IP used below with your IP noted earlier.
   * Mac OS X

$ sudo vim /etc/hosts

**---**

IP Address Host

192.0.2.1 api.pks.<INSERT DOMAIN HERE>

* + Windows

Use Notepad and open c:\Windows\System32\Drivers\etc\hosts with Administrator rights.

**---**

IP Address Host

192.0.2.1 api.pks.<INSERT DOMAIN HERE>

* + Linux

$ sudo vim /etc/hosts

**---**

IP Address Host

192.0.2.1 api.pks.<INSERT DOMAIN HERE>

**Deploy a Kubernetes Cluster**

1. Using the PKS CLI, Login to the PKS API by typing pks login -a https://<API-HOSTNAME>:9021 -u <UAA-USERNAME> -p <UAA-USER-PASSWORD>

$ pks login -a https:*//api.pks.pivotal.io:9021 -u pks\_admin -p secretpassword -k*

* + The API hostname if following this tutorial should be api.pks.<INSERT-DOMAIN-HERE>
  + The Username would be the user you created in UAA using the UAAC cli [here](https://pks-workshop.education.pivotal.io/pks/uaa/demo.html).
  + The Password would be the password for the user you created in UAA using the UAAC cli [here](https://pks-workshop.education.pivotal.io/pks/uaa/demo.html).

1. Provision a Kubernetes Cluster by typing pks create-cluster <CLUSTER-NAME> --external-hostname pks.internal --plan default
   * The Cluster Name can be replaced with any name you choose.

$ pks **create**-cluster dev *--external-hostname pks.internal --plan medium*

* + *Note: The --external-hostname flag determines which hostname PKS will add to the certificate to authenticate to the Kubernetes cluster with. The hostname you place here you will have to be load-balanced to the Kubernetes Master IP. We utilize the local resolution file to do this later on.*

1. Watch the status of our cluster. It can take up to 10 minute to create the first cluster.

$ pks **show**-cluster dev

**Load Balancing a Kubernetes Cluster**

This guide is for proof of concepts, and non-production use. Typically for each Kubernetes cluster deployed with PKS you would point a load balancer to the master node. Here we do not deploy a load balancer for the PKS cluster. Instead we will use our local domain name resolution file to "load balance" us to the Kubernetes Master node.

1. Determine the Kubernetes Master Node IP
   * Since this is our first cluster we can expect the first IP in the range of the PKS-Clusters network to be our master node.
   * To determine this IP follow the [Using BOSH to show PKS Clusters](https://pks-workshop.education.pivotal.io/pks/using-bosh-to-show-pks-clusters/demo.html) guide. This will also show you how to view the Kubernetes cluster for debugging and obtaining logs. ***Don't forget to come back and finish this demo!***
2. Setup Local Domain Name Resolution to the Kubernetes Master Node
   * In order to have our DNS system resolve the kubernetes cluster hostname to the correct IP lets edit the local host resolution file. Replace the IP with your IP noted earlier.
   * Mac OS X

$ sudo vim /etc/hosts

*---*

IP Address Host

192.0.2.1 pks.internal

* + Windows

Use Notepad and open c:\Windows\System32\Drivers\etc\hosts with Administrator rights.

*---*

IP Address Host

192.0.2.1 pks.internal

* + Linux

$ sudo vim /etc/hosts

*---*

IP Address Host

192.0.2.1 pks.internal

**Using BOSH to show PKS Clusters**

**Goal**

To explore the PKS Cluster VMs/IPs/Logs we must dive deeper into the PCF Director, also known as BOSH.

**Prerequisites**

* Deployed Pivotal Ops Manager (2.0+)
* Deployed Pivotal Director Tile
  + Post-deploy scripts enabled
* Deployed Pivotal Container Service Tile

**Access the BOSH CLI**

1. Navigate to the Pivotal Ops Manager FQDN.
2. Login.
3. Click on the Pivotal Director Tile (vSphere).
4. Click the tab labeled status. Here is the list of VMs deployed by the platform and the current status.
5. Note the IP of the Ops Manager Director job down, this is the Director. The director has the knowledge of all kubernetes clusters deployed.
6. Click the tab labeled credentials. Here is the Pivotal Director credentials that the platform auto generates when deploying tiles.
7. Click Link to Credential under Director Credentials.
8. Note the identity, and password down. This is the username and password we will use to connect to the Director.
9. Using SSH and the password set when deploying Pivotal Ops Manager, SSH to the Ops Manager VM.

$ ssh ubuntu@<PIVOTAL-OPS-MANAGER-FQDN>

1. Use the bosh cli to connect to the Director using the IP found earlier.

$ bosh **alias**-env pcf -e <DIRECTOR-IP> --ca-cert /**var**/tempest/workspaces/**default**/root\_ca\_certificate

1. Login to the bosh cli using the username, and password found earlier.

$ bosh -e pcf login

**View BOSH Deployments**

1. BOSH deploys our PKS clusters. Lets list those deployments.

$ bosh -e pcf deployments

Using environment '10.0.2.1' **as** user 'director' (bosh.\*.read, openid, bosh.\*.admin, bosh.read, bosh.admin)

Name Release(s) Stemcell(s) Team(s) Cloud Config

pivotal-container-service-fc2740c309c815ffadf5 bosh-dns/0.0.11 bosh-vsphere-esxi-ubuntu-trusty-go\_agent/3468.13 - latest

docker/30.1.4

kubo/0.10.0

kubo-etcd/6

kubo-service-adapter/0.5.0-dev.191

on-demand-service-broker/0.19.0

pks-api/0.0.0-dev.219

pks-helpers/15.0.0

pks-nsx-t/0.1.0-dev.75

pks-nsx-t-precheck/0.1.0-dev.72

postgres/23

service-instance\_2c4ca713-f66e-4837-b5e5-d91c324f8cb0 bosh-dns/0.0.11 bosh-vsphere-esxi-ubuntu-trusty-go\_agent/3468.13 pivotal-container-service-fc2740c309c815ffadf5 latest

docker/30.1.4

kubo/0.10.0

kubo-etcd/6

pks-helpers/15.0.0

pks-nsx-t/0.1.0-dev.75

* + Notice the Pivotal-container-service deployment which was deployed from the PKS tile. Also notice the service-instance\_\* which is our deployed PKS cluster, note it down.

1. With the deployment name we can list the vms part of it. Notice the master IP of the PKS cluster.

$ bosh -e pcf -d service-instance\_2c4ca713-f66e-4837-b5e5-d91c324f8cb0 instances

Using environment '10.0.2.1' as user 'director' (bosh.\*.read, openid, bosh.\*.admin, bosh.read, bosh.admin)

Task 57. Done

Deployment 'service-instance\_2c4ca713-f66e-4837-b5e5-d91c324f8cb0'

Instance Process State AZ IPs

master/40eea419-3f84-43f0-b6b4-a3c1e99b7f4f running AZ1 10.0.2.51

worker/ae7edd19-2356-4333-8b70-41d8cc7ffc9c running AZ1 10.0.2.52

2 instances

Succeeded

1. Exit out of the SSH Client

$ exit

**Interacting With Kubernetes Clusters**

**Goal**

Connect to a Kubernetes cluster and view the Kubernetes Dashboard.

**Prerequisites**

* Deployed Pivotal Ops Manager (2.0+)
* Deployed Pivotal Director Tile
  + Post-deploy scripts enabled
* Deployed Pivotal Container Service Tile
* Provisioned Kubernetes Cluster
* Installed PKS CLI

**Install the Kubernetes CLI - kubectl**

1. In order to run containers (pods) on Kubernetes we need the Kubernetes CLI -- kubectl
   * Mac OS X

Download the release with the command:

$ curl -LO https://storage.googleapis.com/kubernetes-**release**/**release**/`curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt`/**bin**/darwin/amd64/kubectl

Make the kubectl binary executable.

$ chmod +x ./kubectl

Move the binary in to your PATH.

$ sudo mv ./kubectl /usr/local/bin/kubectl

* + Windows

Download the release from this [link](https://storage.googleapis.com/kubernetes-release/release/v1.9.0/bin/windows/amd64/kubectl.exe).

Or if you have curl installed, use this command:

$ curl -LO https://storage.googleapis.com/kubernetes-**release**/**release**/v1.9.0/**bin**/windows/amd64/kubectl.exe

Add the binary in to your PATH:

* + - Right click Computer, Select Properties
    - Select Advanced Tab
    - Select Environment Variables
    - select PATH
    - Select edit
    - add ;<PATH-TO-PKS-BINARY>
  + Linux

Download the release with the command:

$ curl -LO https://storage.googleapis.com/kubernetes-**release**/**release**/$(curl -s https://storage.googleapis.com/kubernetes-**release**/**release**/stable.txt)/**bin**/linux/amd64/kubectl

Make the kubectl binary executable.

$ chmod +x ./kubectl

Move the binary in to your PATH.

$ sudo mv ./kubectl /usr/local/bin/kubectl

**Authenticate to the Kubernetes Cluster**

1. Now that we have the CLI we must authenticate to our cluster. First list the kubernetes clusters deployed by PKS.

pks list-clusters

1. Once we know the kubernetes cluster we want to communicate with we need to setup an admin user in the cluster and setup a local ~/.kube/config for the kubectl CLI.

pks get-credentials <CLUSTER-NAME>

*Note: If OIDC is enabled you must provide the UAA Admin Password*

**Viewing Kubernetes Internals**

1. Congratulations! You now are in control of a kubernetes cluster!
   * To view the cluster info and list of available worker nodes in your terminal type kubectl cluster-info:

$ kubectl cluster-info

Kubernetes master **is** running at https:*//api.pks.cloud.pivotal.io:8443*

Heapster **is** running at https:*//api.pks.cloud.pivotal.io:8443/api/v1/namespaces/kube-system/services/heapster/proxy*

KubeDNS **is** running at https:*//api.pks.cloud.pivotal.io:8443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy*

monitoring-influxdb **is** running at https:*//api.pks.cloud.pivotal.io:8443/api/v1/namespaces/kube-system/services/monitoring-influxdb:http/proxy*

* + To further debug and diagnose cluster problems, use kubectl cluster-info dump.
  + To view the worker nodes that BOSH deployed and is monitoring for us in your terminal type kubectl get nodes -o wide:

$ kubectl get nodes -o wide

NAME STATUS ROLES AGE VERSION EXTERNAL-IP OS-IMAGE KERNEL-VERSION CONTAINER-RUNTIME

vm-422fd7e6-a815-40b9-552a-b2a25033c9f7 Ready <none> 20m v1.8.4 35.229.101.179 Ubuntu 14.04.5 LTS 4.4.0-103-generic docker:*//1.13.1*

vm-623978bf-6ccc-412e-5673-ee7fc5d3d4a1 Ready <none> 24m v1.8.4 35.227.65.230 Ubuntu 14.04.5 LTS 4.4.0-103-generic docker:*//1.13.1*

vm-c56d46b6-a652-4159-775f-88cd64abf6e0 Ready <none> 22m v1.8.4 35.196.87.120 Ubuntu 14.04.5 LTS 4.4.0-103-generic docker:*//1.13.1*

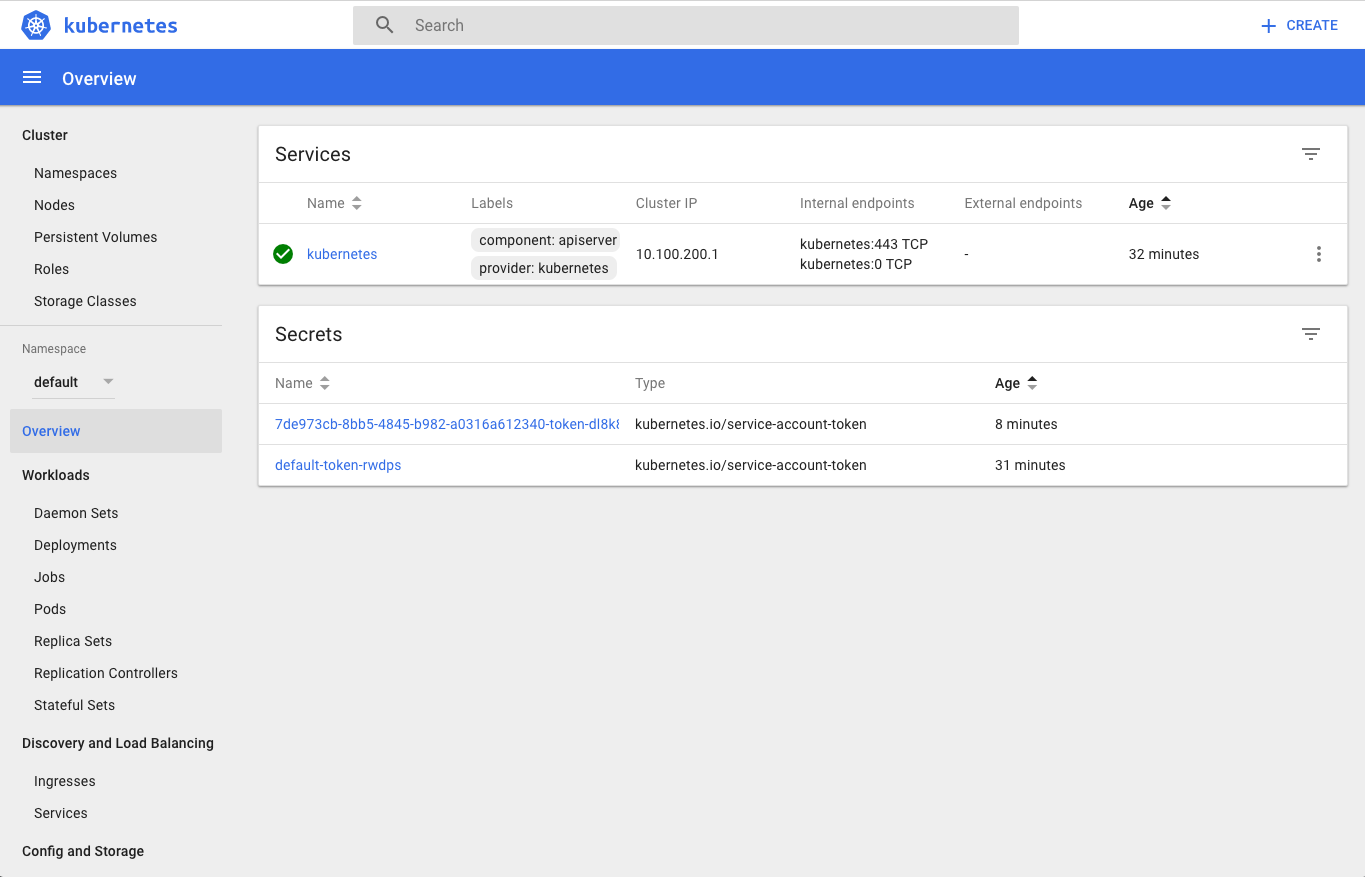
**Enjoy your Kubernetes Cluster!**

1. Congratulations! You now are in control of a kubernetes cluster!
2. Though much of your interaction with Kubernetes will be using the kubectl CLI, Kubernetes also includes a dashboard application that gives you insight into the cluster and its API objects.
   * This dashboard can be accessed by setting up a proxy using the kubectl CLI. In your terminal type kubectl proxy and leave your command window running to keep the proxy open.

$ kubectl proxy

Starting to serve on 127.0.0.1:8001

* + To access the dashboard open a web browser and navigate to http://localhost:8001/ui
  + For security you will be asked for a kubernetes config file - kubeconfig. PKS created us one with the PKS get-credentials command. Browse to it and then you will have access to your cluster dashboard!
    - Mac OS X - $HOME/.kube/config
    - Linux - $HOME/.kube/config
    - Windows - %USERPROFILE%/.kube/config



# PKS Pipelines

In order to control the sprawl of kubernetes, and to guarantee each kubernetes cluster is deployed with the exact same configuration both [concourse](https://concourse-ci.org/) and [PKS](https://pivotal.io/platform/pivotal-container-service) are recommended to be used.

## Pre-Requisites

* Concourse Deployed
* PKS Deployed
* git Compatible Repository Available
* fly CLI to communicate with concourse
* Access to github.com to retrieve Pivotal created pipelines

## (Basic) Creating PKS Clusters

In this part of the demo a simple concourse pipeline is used to deploy K8s with PKS in a pre-configured way.

### Setup the Create-Cluster Pipeline

1. Clone https://github.com/Pivotal-Field-Engineering/pks-workshop.git to obtain create-pks-clusters-pipeline.yml and params.yml.
2. Open the params.yml in your text editor. Read the comments and supply the values requested under **CREATE PKS CLUSTERS PIPELINE**.
3. Login to your concourse instance with fly.

fly -t <YOUR-TARGET> login -k --concourse-url=<YOUR-CONCOURSE-URL>

1. Set the Pipeline with both create-pks-clusters-pipeline.yml and params.yml to begin monitoring your PKS Config.

fly -t <YOUR-TARGET> **set**-pipeline -p pks-clusters-create -c create-pks-clusters-pipeline.yml -l params.yml

1. Notice all clusters are manually deployed by triggering the create-cluster step. Let’s do this now!

* Log into the concourse GUI by navigating to your concourse URL.
* Un-pause the pipeline
* Click on the create-cluster job.
* Click the + icon in the upper right-hand corner.

If all was successful you should see the output of your kube.conf which you can now use with kubectl to login to your newly create K8s cluster.

## (Advanced) Monitor Desired PKS Clusters vs Current PKS Clusters

Building on the above we now understand how Concourse can deploy a single PKS cluster in a pre-configured way, but as an enterprise we will have many that change constantly overtime.

In this part of the demo a concourse pipeline is used to track the desired state of PKS and resolve any differences. This enables us to scale clusters, add clusters, delete clusters, all while maintaining a change log, and audit log for who owns each cluster.

### Store Desired PKS State

For storing our desired PKS clusters state we will utilize git. This demo utilizes github but any compatible repository should work.

1. Create a repository for storing your PKS clusters.

* An example can be found [here](https://github.com/Oskoss/peters-pks-clusters)

1. Create a basic PKS configuration similar to below, name it desired.json and place it in the root of the repository. This will be your PKS clusters configuration JSON. It will store all clusters this PKS install will deploy, how many workers each one has, and which plan each one was created with. Ensure your PKS environment has been configured with the small plan otherwise choose the correct plan names for your PKS environment.

pks-workshop/desired\_1.json

[

{

"name": "dev-cluster",

"plan\_name": "small",

"parameters": {

"kubernetes\_worker\_instances": 3

}

}

]

Note: The JSON format complies with the format determined from the PKS CLI: *pks clusters --json*

1. Commit, and push the desired.json file.

### Setup the Monitor PKS Clusters Pipeline

1. Clone https://github.com/Pivotal-Field-Engineering/pks-workshop.git to obtain monitor-pks-clusters-pipeline.yml and params.yml.
2. Open the params.yml in your text editor. Read the comments and supply the values requested under both **CREATE PKS CLUSTERS PIPELINE** and **MONITOR PKS CLUSTERS PIPELINE**.
3. Login to your concourse instance with fly.

fly -t <YOUR-TARGET> login -k --concourse-url=<YOUR-CONCOURSE-URL>

1. Set the Pipeline with both monitor-pks-clusters-pipeline.yml and params.yml to begin monitoring your PKS Config.

fly -t <YOUR-TARGET> **set**-pipeline -p pks-clusters-monitor -c monitor-pks-clusters-pipeline.yml -l params.yml

1. Notice since our desired.json contained a single PKS cluster definition Concourse has kicked PKS off to create a new K8s cluster for us.

### Create a New PKS Cluster with our External Configuration

1. Inside our PKS State git repository open desired.json in your text editor.
2. Add a new cluster by adding the following block to the JSON.

{

"name": "prod-cluster",

"plan\_name": "medium",

"parameters": {

"kubernetes\_worker\_instances": 5

}

}

1. The end result should look similar to the following. Ensure your PKS environment has been configured with small and medium plans otherwise rename them to the correct plan names for your PKS environment.

pks-workshop/desired\_2.json

[

{

"name": "dev-cluster",

"plan\_name": "small",

"parameters": {

"kubernetes\_worker\_instances": 3

}

},

{

"name": "prod-cluster",

"plan\_name": "medium",

"parameters": {

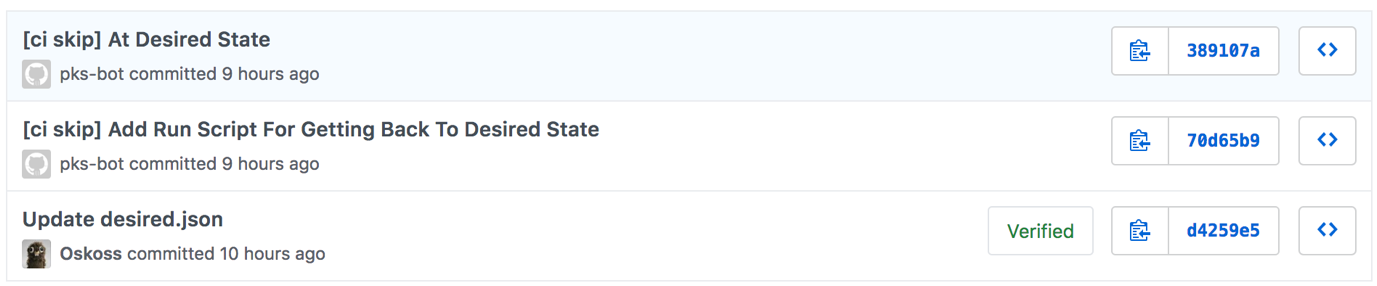
"kubernetes\_worker\_instances": 5

}

}

]

1. Commit, and push the new desired.json file.
2. Watch as your pipeline automatically detects the new changes and creates the new PKS cluster for us.
3. Once all clusters have been created examine the git log of the PKS State git repository. It should look similar to the below.



* Notice our commit adding the prod-cluster to the desired.json file. Followed by a commit that added a runPKS.sh script, and finally by a commit that removed that script. While the first commit was our commit the second and third commits were done by our pipeline!

### Scale and Delete Existing PKS Clusters with our External Configuration

1. Inside our PKS State git repository open desired.json in your text editor.
2. Lets delete the prod-cluster and scale our dev-cluster by editing the JSON parameters.
3. The end result should look similar to the following. Ensure your PKS environment has been configured with the smallplan otherwise rename it to the correct plan name for your PKS environment.

pks-workshop/desired\_3.json

[

{

"name": "dev-cluster",

"plan\_name": "small",

"parameters": {

"kubernetes\_worker\_instances": 5

}

}

]

1. Commit, and push the new desired.json file.
2. Watch as your pipeline automatically detects the new changes, deletes our prod-cluster and scales our dev-cluster for us.

**Using the UAA Client (UAAC) with PKS**

In order to create a Kubernetes Cluster with PKS we utilize the PKS CLI. Since PKS is backed by UAA for authorization, we must have a username and password stored in UAA which we then will utilize to log into PKS.

While external identity resources are supported for backing UAA this demo is focused on using local users stored in UAA.

**Pre-Requisites**

* Pivotal Ops Manager deployed
* Pivotal Director Tile deployed
  + Post-deploy scripts enabled
* Pivotal Container Service Tile deployed

**Goal**

Create a new local user in UAA with the UAA Client (UAAC) and then login PKS with this user.

**Determine UAA Admin Credentials**

1. Navigate to the Ops Manager FQDN (hostname) in your browser of choice.
2. Login with the Ops Manager credentials setup earlier.
3. Click on Pivotal Container Service Tile.
4. Click on Credentials.
5. Find the **Pks Uaa Management Admin Client** entry and click on Link to Credential.
6. Note down the secret.
7. Click on Status.
8. Note the IP for the Pivotal Container Service job down.

**Access the UAAC CLI**

In order to create a new user in UAA we need to use the UAA Client (UAAC) CLI , which requires Ruby locally.

* If you do not wish to install Ruby and UAAC locally, skip the following 2 steps and utilize UAAC on the Ops Manger VM.
  + To access the Ops Manager, use SSH and the password set when deploying the Pivotal Ops Manager

$ ssh ubuntu@<PIVOTAL-OPS-MANAGER-FQDN>

1. Install Ruby, help with this can be found [here](https://www.ruby-lang.org/en/documentation/installation/).
2. Install the cf-uaac RubyGem

$ gem **install** cf-uaac

**Access the deployed UAA Service**

In order to reach the UAA service we must ensure either our DNS or Load Balancer is sending us to the correct UAA url. We cannot utilize IPs due to the default security enforcement built into UAA. **Skip this step if you deployed a load balancer and/or registered the PKS Management API hostname with your DNS**

1. In order to have our ***Local*** DNS system resolve the PKS Management API hostname to the correct IP lets edit the local host resolution file. Replace the IP used below with your IP noted earlier. If using the PCF Ops Manager VM use Linux below.
   * Mac OS X

$ sudo vim /etc/hosts

**---**

IP Address Host

192.0.2.1 api.pks.<INSERT DOMAIN HERE>

* + Windows

Use Notepad and open c:\Windows\System32\Drivers\etc\hosts with Administrator rights.

**---**

IP Address Host

192.0.2.1 api.pks.<INSERT DOMAIN HERE>

* + Linux

$ sudo vim /etc/hosts

**---**

IP Address Host

192.0.2.1 api.pks.<INSERT DOMAIN HERE>

**Grant Cluster Access to a User**

1. Target your UAA API endpoint using uaac target https://<PKS-API>:8443 --skip-ssl-validation.

$ uaac target https:*//api.pks.pivotal.io:8443 --skip-ssl-validation*

* Replace with the URL to your PKS API server.You configured this URL in the PKS API section of Installing PKS for your IaaS.

1. Authenticate with UAA using the secret you retrieved in the previous section using uaac token client get admin -s <INSERT GENERATED UAA ADMIN SECRET>.

$ uaac token client get admin -s generated-**super**-secret

1. Create a user by running uaac user add <USERNAME> --emails <USER-EMAIL> -p <USER-PASSWORD>.

$ uaac user add pks\_admin --emails pks\_admin@pivotal.io -p password

1. Assign a scope to the user to allow them to access Kubernetes clusters. Use uaac member add <UAA-SCOPE> <USERNAME>.

$ uaac member add pks.clusters.admin pks\_admin

* + Replacing UAA-SCOPE with one of the following UAA scopes depending on the role of the user:
    - pks.clusters.admin: Users with this scope have full access to all clusters.
    - pks.clusters.manage: Users with this scope can only access clusters they create.

**Login to PKS With Newly Created User**

1. Login to your PKS API Server using pks login -a <PKS-API> -u <USERNAME> -p <USER-PASSWORD> -k

$ pks login -a api.pks.pivotal.io -u dev01 -p devpassword -k

# Setting User Roles with Kubernetes Role Bindings

For a developer to kubectl any object in a Kubernetes cluster we must define what and how much access they must have.

While external identity resources (LDAP) are supported for backing UAA this demo is focused on using local users stored in UAA.

## Pre-Requisites

* Pivotal Ops Manager deployed
* Pivotal Director Tile deployed
  + Post-deploy scripts enabled
* Pivotal Container Service Tile
  + UAA as OIDC Provider enabled
* Kubernetes Cluster deployed

## Goal

Create a new user which has access to all objects in the default kubernetes namespace.

## Create Local UAA User - (Not required if UAA is integrated with LDAP)

### Determine UAA Admin Credentials

1. Navigate to the Ops Manager FQDN (hostname) in your browser of choice.
2. Login with the Ops Manager credentials setup earlier.
3. Click on Pivotal Container Service Tile.
4. Click on Credentials.
5. Find the **Pks Uaa Management Admin Client** entry and click on Link to Credential.
6. Note down the secret.
7. Click on Status.
8. Note the IP for the Pivotal Container Service job down.

### Access the UAAC CLI

In order to create a new user in UAA we need to use the UAA Client (UAAC) CLI , which requires Ruby locally.

* If you do not wish to install Ruby and UAAC locally, skip the following 2 steps and utilize UAAC on the Ops Manger VM.
  + To access the Ops Manager, use SSH and the password set when deploying the Pivotal Ops Manager

$ ssh ubuntu@<PIVOTAL-OPS-MANAGER-FQDN>

1. Install Ruby, help with this can be found [here](https://www.ruby-lang.org/en/documentation/installation/).
2. Install the cf-uaac RubyGem

$ gem **install** cf-uaac

### Access the deployed UAA Service

In order to reach the UAA service we must ensure either our DNS or Load Balancer is sending us to the correct UAA url. We cannot utilize IPs due to the default security enforcement built into UAA. **Skip this step if you deployed a load balancer and/or registered the PKS Management API hostname with your DNS**

1. In order to have our ***Local*** DNS system resolve the PKS Management API hostname to the correct IP lets edit the local host resolution file. Replace the IP used below with your IP noted earlier. If using the PCF Ops Manager VM use Linux below.
   * Mac OS X

$ sudo vim /etc/hosts

**---**

IP Address Host

192.0.2.1 api.pks.<INSERT DOMAIN HERE>

* + Windows

Use Notepad and open c:\Windows\System32\Drivers\etc\hosts with Administrator rights.

**---**

IP Address Host

192.0.2.1 api.pks.<INSERT DOMAIN HERE>

* + Linux

$ sudo vim /etc/hosts

**---**

IP Address Host

192.0.2.1 api.pks.<INSERT DOMAIN HERE>

### Create New Local User

1. Target your UAA API endpoint using uaac target https://<PKS-API>:8443 --skip-ssl-validation.

$ uaac target https:*//api.pks.pivotal.io:8443 --skip-ssl-validation*

* + Replace with the URL to your PKS API server. You configured this URL in the PKS API section of Installing PKS for your IaaS.

1. Authenticate with UAA using the secret you retrieved in the previous section using uaac token client get admin -s <INSERT GENERATED UAA ADMIN SECRET>.

$ uaac token client get admin -s generated-**super**-secret

1. Create a user by running uaac user add <USERNAME> --emails <USER-EMAIL> -p <USER-PASSWORD>.

$ uaac user add dev01 --emails dev01@pivotal.io -p password

## Create a Kubernetes Role and Role Binding

Once a user is created in UAA (or obtained from LDAP through UAA), Kubernetes needs to understand what rights this user must have. Kubernetes does this through the use of Kubernetes Roles, and Role Bindings.

### View the default Kubernetes Namespaces, and Roles To Assign Our User

1. Log into your Kubernetes Cluster with an admin role. Refer to [Manage Kubernetes Clusters](https://pks-workshop.education.pivotal.io/pks/manage-kubernetes-clusters/demo.html), and [Interacting with Kubernetes Clusters](https://pks-workshop.education.pivotal.io/pks/interacting-kubernetes-clusters/demo.html) for more information.
2. View the preconfigured Kubernetes ClusterRoles using kubectl get clusterroles.

$ kubectl get clusterroles

NAME AGE

admin 1h

cluster-admin 1h

edit 1h

heapster-node-stats 1h

kubo:internal:kubelet-drain 1h

kubo:route-sync 1h

namespace-sink-modifier 1h

ncp-cluster-role 1h

ncp-patch-role 1h

nsx-node-agent-cluster-role 1h

system:aggregate-to-admin 1h

system:aggregate-to-edit 1h

system:aggregate-to-view 1h

system:auth-delegator 1h

system:aws-cloud-provider 1h

system:basic-user 1h

system:certificates.k8s.io:certificatesigningrequests:nodeclient 1h

system:certificates.k8s.io:certificatesigningrequests:selfnodeclient 1h

system:controller:attachdetach-controller 1h

system:controller:certificate-controller 1h

system:controller:clusterrole-aggregation-controller 1h

system:controller:cronjob-controller 1h

system:controller:daemon-set-controller 1h

system:controller:deployment-controller 1h

system:controller:disruption-controller 1h

system:controller:endpoint-controller 1h

system:controller:expand-controller 1h

system:controller:generic-garbage-collector 1h

system:controller:horizontal-pod-autoscaler 1h

system:controller:job-controller 1h

system:controller:namespace-controller 1h

system:controller:node-controller 1h

system:controller:persistent-volume-binder 1h

system:controller:pod-garbage-collector 1h

system:controller:pv-protection-controller 1h

system:controller:pvc-protection-controller 1h

system:controller:replicaset-controller 1h

system:controller:replication-controller 1h

system:controller:resourcequota-controller 1h

system:controller:route-controller 1h

system:controller:service-account-controller 1h

system:controller:service-controller 1h

system:controller:statefulset-controller 1h

system:controller:ttl-controller 1h

system:csi-external-attacher 1h

system:csi-external-provisioner 1h

system:discovery 1h

system:heapster 1h

system:kube-aggregator 1h

system:kube-controller-manager 1h

system:kube-dns 1h

system:kube-scheduler 1h

system:kubelet-api-admin 1h

system:metrics-server 1h

system:node 1h

system:node-bootstrapper 1h

system:node-problem-detector 1h

system:node-proxier 1h

system:persistent-volume-provisioner 1h

system:volume-scheduler 1h

telemetry-metrics 58m

view 1h

1. Describe the edit role using kubectl describe clusterrole edit. This is the role we will be binding to our new user dev01.

$ kubectl **describe** clusterrole edit

**Name**: edit

Labels: kubernetes.io/bootstrapping=rbac-**defaults**

Annotations: rbac.authorization.kubernetes.io/autoupdate=true

PolicyRule:

Resources Non-**Resource** URLs **Resource** **Names** Verbs

*--------- ----------------- -------------- -----*

serviceaccounts [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch impersonate]

configmaps [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

endpoints [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

persistentvolumeclaims [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

pods/attach [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

pods/exec [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

pods/portforward [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

pods/proxy [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

pods [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

replicationcontrollers/scale [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

replicationcontrollers [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

secrets [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

services/proxy [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

services [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

daemonsets.apps [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

deployments.apps/**rollback** [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

deployments.apps/scale [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

deployments.apps [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

replicasets.apps/scale [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

replicasets.apps [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

statefulsets.apps/scale [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

statefulsets.apps [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

horizontalpodautoscalers.autoscaling [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

cronjobs.batch [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

jobs.batch [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

daemonsets.extensions [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

deployments.extensions/**rollback** [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

deployments.extensions/scale [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

deployments.extensions [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

ingresses.extensions [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

networkpolicies.extensions [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

replicasets.extensions/scale [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

replicasets.extensions [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

replicationcontrollers.extensions/scale [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

networkpolicies.networking.k8s.io [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

poddisruptionbudgets.policy [] [] [**create** **delete** deletecollection **get** **list** **patch** **update** watch]

bindings [] [] [**get** **list** watch]

**events** [] [] [**get** **list** watch]

limitranges [] [] [**get** **list** watch]

namespaces/**status** [] [] [**get** **list** watch]

namespaces [] [] [**get** **list** watch]

pods/**log** [] [] [**get** **list** watch]

pods/**status** [] [] [**get** **list** watch]

replicationcontrollers/**status** [] [] [**get** **list** watch]

resourcequotas/**status** [] [] [**get** **list** watch]

resourcequotas [] [] [**get** **list** watch]

1. To see which namespaces we should give our dev01 user access to list them using kubectl get namespaces.

$ kubectl **get** namespaces

NAME STATUS AGE

**default** Active 1h

kube-**public** Active 1h

kube-system Active 1h

pks-system Active 1h

### Bind the Kubernetes Namespace, User, and Role together

1. Create a basic KubernetesRoleBinding similar to below, name it k8-role-binding.yaml. This will "bind" our dev01 user created earlier to a kubernetes rule edit in the default namespace. Ensure the user created earlier was dev01 otherwise change the file appropriately. Ensure the namespace default was found earlier otherwise change the file appropriately.

[Hide k8-role-binding.yaml](https://pks-workshop.education.pivotal.io/pks/role-bindings/demo.html" \l "pks-workshop23f07b66-252e-45a7-891c-6fe39e621154)

pks-workshop/k8-role-binding.yaml

*# This role binding allows "dev01" to read/write all objects in the "default" namespace.*

kind: RoleBinding

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: namespace-user

namespace: default *# This only grants permissions within the "default" namespace.*

subjects:

- kind: User

name: dev01 *# Name is case sensitive*

apiGroup: rbac.authorization.k8s.io

roleRef:

kind: ClusterRole

name: edit *# Default K8s Role - grants read/write permissions to all object in namespace except for Roles and RoleBindings.*

apiGroup: rbac.authorization.k8s.io

1. Apply the object to Kubernetes using kubectl apply -f k8-role-binding.yaml.
2. Finally ensure the object was accepted using kubectl get rolebindings.

$ kubectl get rolebindings

NAME AGE

namespace-user 10s

## Login to Kubernetes as a developer (The Local User or LDAP User)

### Setup the Kubeconfig

1. Since there are no native Kubernetes helpers to build a kubeconfig we have built a helper for this. It is a simple BASH helper script that can be run from a local machine, jumpbox, or CI/CD pipeline. The helper does the following for us:
   * Login to UAA with user specified, obtain a token
   * Login to K8s, obtain cluster certificate
   * Merge both the above into a single kubeconfig file found under ~/.kube/config
2. To obtain the helper [download it](https://github.com/Pivotal-Field-Engineering/pks-workshop/releases/download/v0.1/get-pks-k8s-config.sh) or clone the [PKS-Workshop repository](https://github.com/Pivotal-Field-Engineering/pks-workshop) where you can find the helper. The helper is named get-pks-k8s-config.sh. Ensure the helper is executable via chmod +x ./get-pks-k8s-config.sh. You can either add the helper to your $PATH or refer to it directly with each command.
3. Use the helper to setup your kubeconfig, as a user you must know the Kubernetes Cluster Master VIP (Or DNS Record), PKS API URL, Username, and Password. Remember if you have been following this tutorial we are using the user dev01. The Kubernetes Cluster Master VIP can be found from pks cluster <CLUSTER-NAME>.

$ ./get-pks-k8s-config.sh *--API=api.pks.haas-69.pez.pivotal.io --CLUSTER=10.195.75.136 --USER=dev01*

Password:Cluster "10.195.75.136" set.

**Context** "10.195.75.136" created.

Switched **to** **context** "10.195.75.136".

**User** "dev01" set.

### Use Kubernetes as a Developer!

1. Attempt to access resources in the default and kube-system namespaces. Notice we are allowed access to the default namespace but not the kube-system namespace as expected!

$ kubectl **get** all -n kube-system

No resources found.

Error from server (Forbidden): pods **is** forbidden: User "dev01" cannot list pods **in** the namespace "kube-system"

Error from server (Forbidden): replicationcontrollers **is** forbidden: User "dev01" cannot list replicationcontrollers **in** the namespace "kube-system"

Error from server (Forbidden): services **is** forbidden: User "dev01" cannot list services **in** the namespace "kube-system"

Error from server (Forbidden): daemonsets.apps **is** forbidden: User "dev01" cannot list daemonsets.apps **in** the namespace "kube-system"

Error from server (Forbidden): deployments.apps **is** forbidden: User "dev01" cannot list deployments.apps **in** the namespace "kube-system"

Error from server (Forbidden): replicasets.apps **is** forbidden: User "dev01" cannot list replicasets.apps **in** the namespace "kube-system"

Error from server (Forbidden): statefulsets.apps **is** forbidden: User "dev01" cannot list statefulsets.apps **in** the namespace "kube-system"

Error from server (Forbidden): horizontalpodautoscalers.autoscaling **is** forbidden: User "dev01" cannot list horizontalpodautoscalers.autoscaling **in** the namespace "kube-system"

Error from server (Forbidden): jobs.batch **is** forbidden: User "dev01" cannot list jobs.batch **in** the namespace "kube-system"

Error from server (Forbidden): cronjobs.batch **is** forbidden: User "dev01" cannot list cronjobs.batch **in** the namespace "kube-system"

$ kubectl **get** all -n **default**

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

service/kubernetes ClusterIP 10.100.200.1 <none> 443/TCP 3h

**PKS Networking with NSX-T**

PKS includes software defined networking with NSX. NSX supports logical networking from the Kubernetes cluster VMs to the pods themselves providing a single network management and control plane for your container based applications. This section will not be an exhaustive look at all of the NSX Kubernetes integration but will focus on a few examples. Also, this section assumes some knowledge of kubernetes, kubectl and yaml configuration files.

**Pre-Requisites**

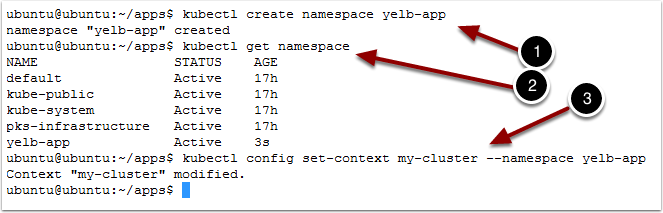
* Pivotal Ops Manager
* Pivotal Container Service Tile
* NSX-T

**Goal**

Familiarize users with NSX Manager functionalities and successfully view NSX-T networking objects.

**Create a namespace**

We will now create a new namespace and set the context so that the cli is pointed to the new namespace.



1. Type kubectl create namespace yelb-app
2. Type kubectl get namespace
3. Type kubectl config set-context my-cluster --namespace yelb-app

This command changes the context for kubectl so that the default namespace to use is the new yelb-app. It keeps you from having to specify the namespace on each command.

**View new objects with NSX-Manager**

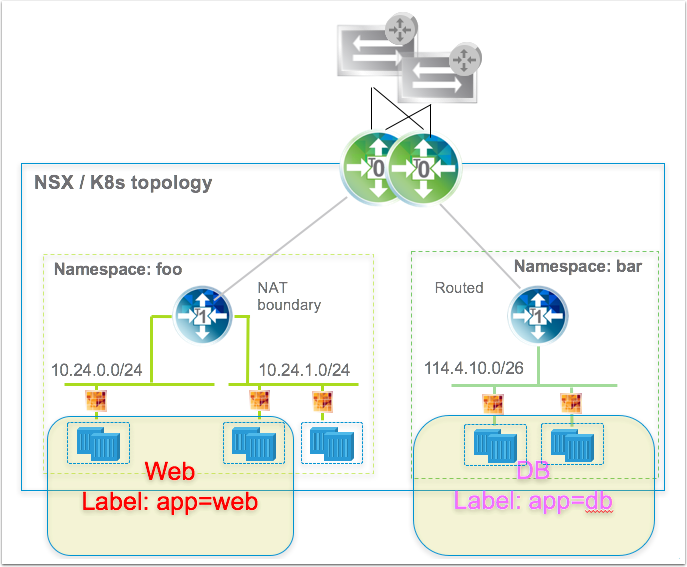
1. Open your browser
2. Navigate to NSX-Manager
3. Enter the username and password
4. Click Log In

**View logical router created automatically**

1. Click on Routing
2. Click on T1 Router created for the yelb-app namespace

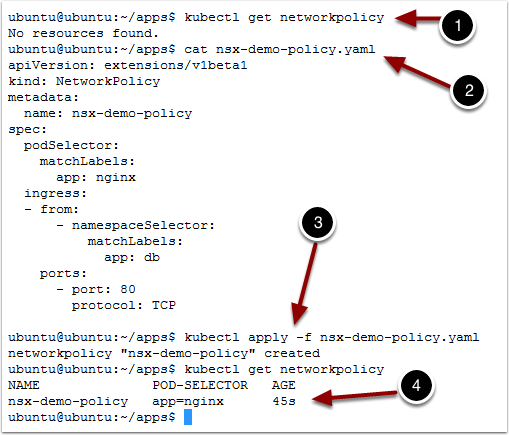
There are T1 routers created for each of our namespaces and the yelb-app T1 router was automatically added when we created the Namespace. If you click on Switching you would see a similar list of Logical Switches. When pods are deployed, Ports are created on the appropriate switch and an IP from the pool is assigned to the pod.

**Kubernetes network policy and microsegmentation**



Using Network Policy, users can define firewall rules to allow traffic into a Namespace and between Pods. The network policy is a Namespace property. Network Admins can define policy in NSX through labels that can then be applied to pods or namespaces. Here we will show how the Kubernetes Network Policy definition causes the firewall rules to be automatically generated in NSX. By default, pods are non-isolated; they accept traffic from any source. Pods become isolated by having a NetworkPolicy that selects them. Once there is any NetworkPolicy in a namespace selecting a particular pod, that pod will reject any connections that are not allowed by any NetworkPolicy. Other pods in the namespace that are not selected by any NetworkPolicy will continue to accept all traffic. In our case, we will add a policy to only allow access to our nginx app from pods in a namespace with label app:db.

**Create a network policy**



We will first check that there are no Network Policies created for this Namespace

1. Type kubectl get NetworkPolicy

Next we look at the network policy we want to create. This one establishes a rule about connectivity to pods with label app:nginx from namespaces with label app:db. Pods that are not in a namespace that matches the label will not be able to connect.

1. Type cat nsx-demo-policy.yaml

[Hide nsx-demo-policy.yaml](https://pks-workshop.education.pivotal.io/pks/kubernetes-networking-nsx-t-dev/demo.html" \l "pks-workshopb027ea85-83df-4855-9183-af66906b0c4f)

pks-workshop/nsx-demo-policy.yaml

kind: NetworkPolicy

apiVersion: extensions/v1beta1

metadata:

name: nsx-demo-policy

spec:

podSelector:

matchLabels:

app: nginx

ingress:

- from:

- namespaceSelector:

matchLabels:

app: dv

port:

- port: 80

protocol: TCP

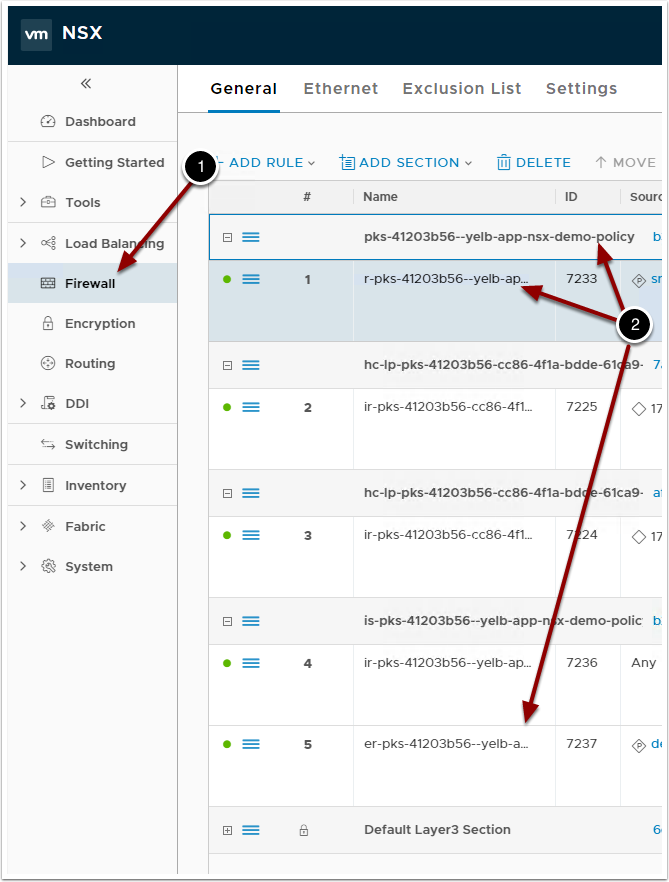
Let's apply that network policy

1. Type kubectl apply -f nsx-demo-policy.yaml

Let's see what we created

1. Type kubectl get NetworkPolicy

**View firewall rules created automatically**

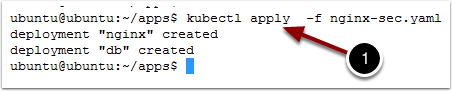


From NSX-Mgr we can see that rules have been created based on our policy. NSX has dynamically created Source and Destination security groups and will apply the right policy

1. Click on Firewall
2. Note the Network Policy Name and the scope being the Namespace we created it from.

**Traceflow**

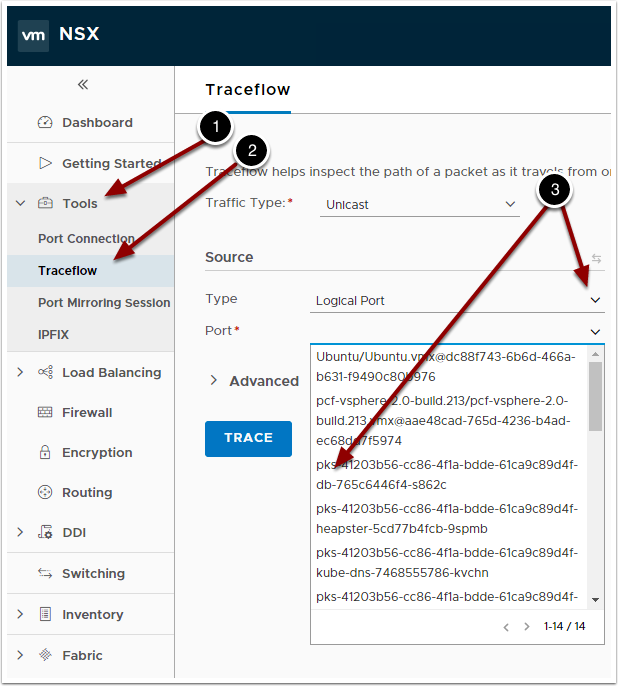
NSX provides the capability to do detailed packet tracing across VMs and between pods. You can tell where a packet might have been dropped between two pods that you have deployed. We will deploy two pods in our namespace. We did not add any labels to our namespace when we created it, so our network policy should prevent communication between the two. Let's create the pods.



1. Type kubectl apply -f /home/ubuntu/apps/nginx-sec.yaml

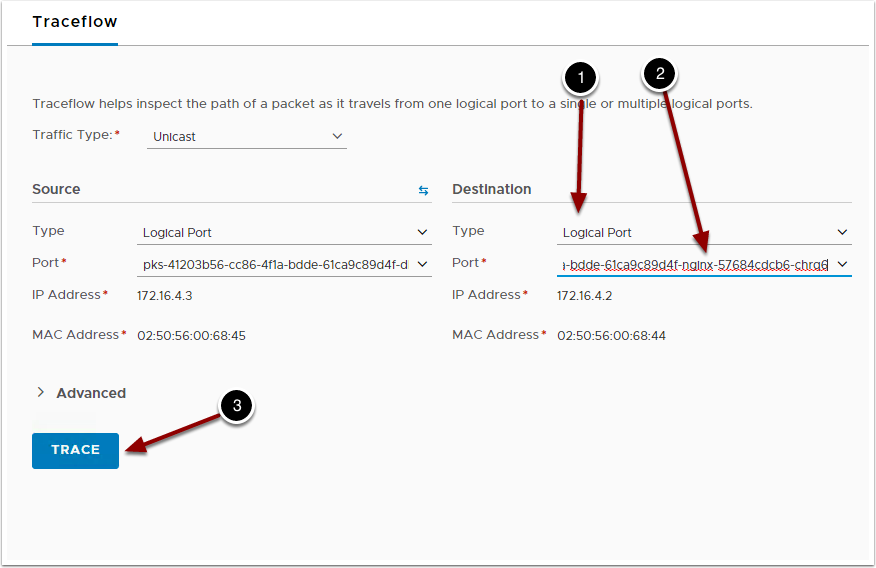
**Configure Traceflow source**

Return to NSX-Mgr in the Browser



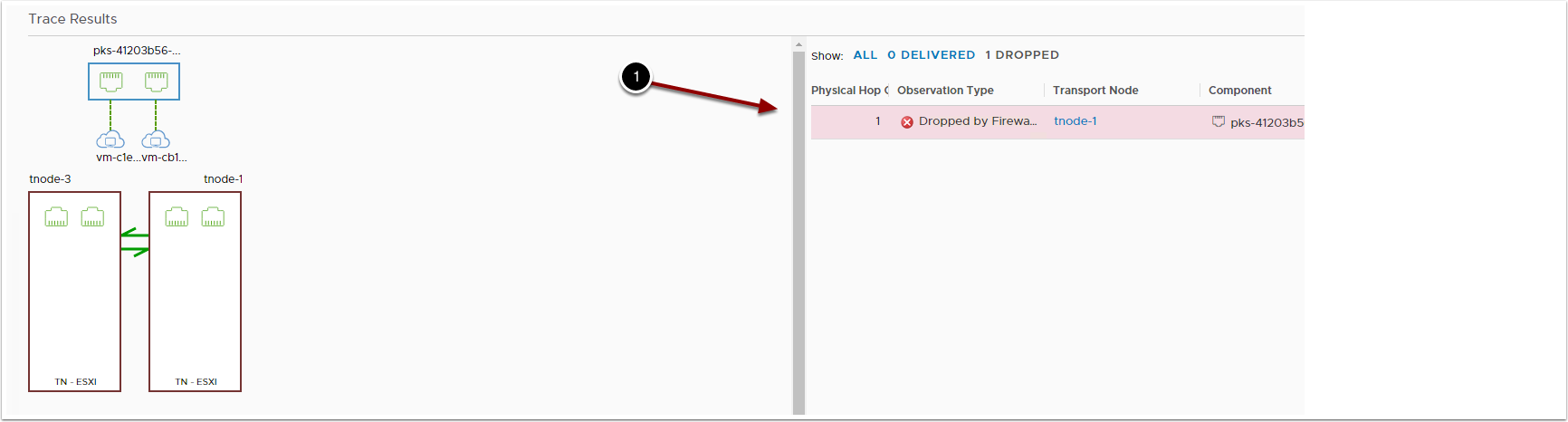
1. Click on Tools
2. Select Traceflow
3. Choose the Logical Port and find a port with "db" in the name as the source

**Configure Traceflow destination**



1. Under Destination, choose Logical Port
2. Choose one of the Ports with Nginx in the name
3. Click Trace

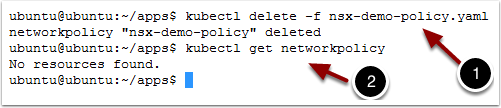
**Verify packets are dropped**



1. The packet was dropped by the firewall

Let's remove the network policy and try this again.

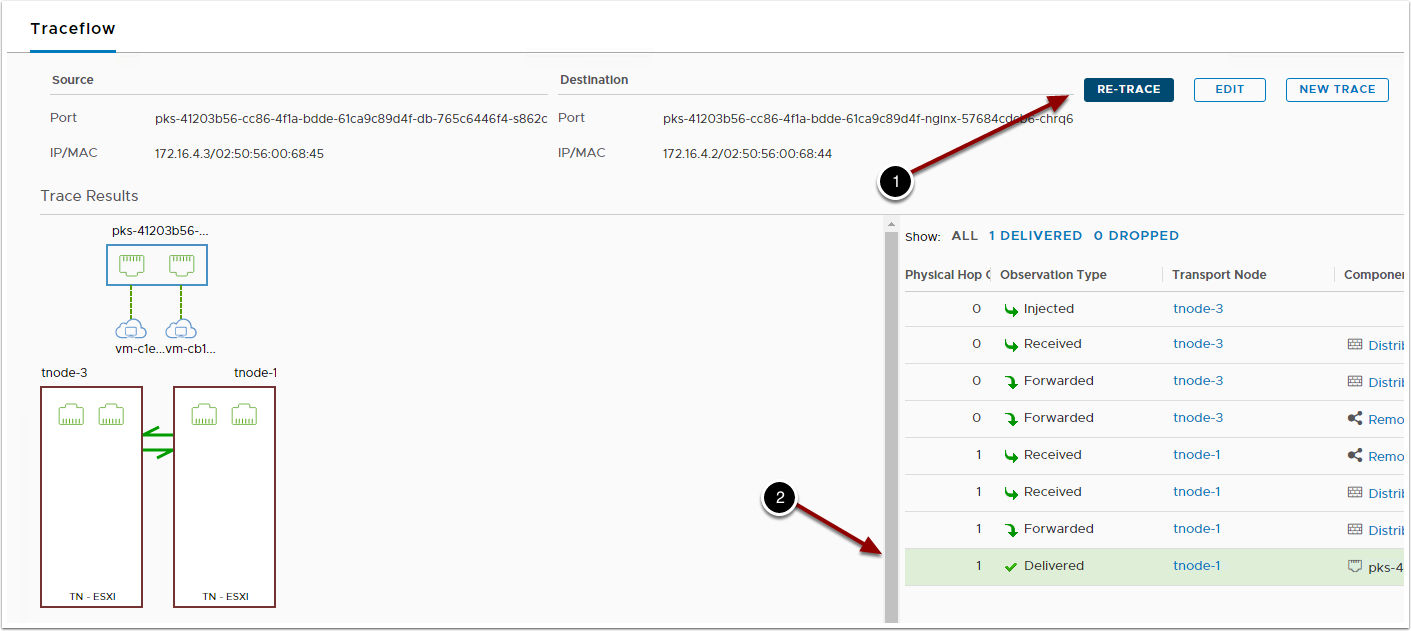
**Remove network policy**



Return to the command line

1. Type kubectl delete -f nsx-demo-policy.yaml
2. Type kubectl get networkpolicy

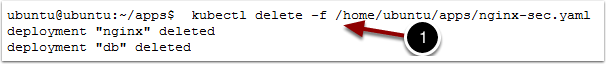
**Re-trace your application**



1. Click the Re-Trace button
2. Once the network policy was removed, the network packet made it to its destination succesfully

Traceflow is a very powerful capability that can also trace from VM to pod, VM to VM and IP to IP. Try out a few more traces on your own.

**Cleanup deployments**



1. Type kubectl delete -f /home/ubuntu/apps/nginx-sec.yaml
2. Type kubectl config set-context my-cluster --namespace default

Returns the kubectl context to the default namespace.

**Harbor (Container Registry)**

**Goal**

Configure and deploy VMware Harbor within PCF Ops Manager. This will enable us to push and pull docker images to a secured image repository while still being managed, and monitored by the PCF Platform.

**Prerequisites**

* Deployed Pivotal Ops Manager (2.0+)
* Deployed Pivotal Director Tile
  + Post-deploy scripts enabled
* Deployed Pivotal Container Service Tile

**Download Harbor Tile**

1. Navigate to [Pivotal Network](https://network.pivotal.io/)
2. Register and login.
3. Download the [Harbor Tile](https://network.pivotal.io/products/pivotal-container-service)
4. Accept the EULA

**Upload PKS Tile to Pivotal Ops Manager**

1. Navigate to the Pivotal Ops Manager FQDN (hostname).
2. Login.
3. Select Import a Product.
4. Browse to the product downloaded earlier.
5. Wait while the product is uploaded, this may take time depending on your connection to the Pivotal Ops Manager. *When completed notice the new product in the side panel of Pivotal Ops Manager.*

**Configure the Harbor Tile**

1. Click the green + on the product tile.
2. Click the VMware Harbor Registry tile. Harbor deploys a vm which contains a docker registry on it which Kubernetes then can utilize to pull images from.
3. Configure the Assign AZs and Networks page.
   * Select an AZ for both the singleton jobs, and all other jobs to run in.
   * Select PKS-Management for network if having followed earlier demos. This network is where the Harbor VM itself will live.
4. Configure the General page.
   * Enter in the hostname that the the VM will assume. We will use harbor.pks.<INSERT DOMAIN HERE>.
5. Configure the Certificate page. Either generate a self signed certificate for our harbor endpoint or copy/paste in the certificate PEM and private key PEM. This certificate will be used when pushing and pulling docker images to the harbor image repository.
   * The certificate must be valid for the hostname we created above and must be an \* asterisk record.
   * The certificate must be valid for the above host.
   * We will use the following: \*.pks.<INSERT DOMAIN HERE>.
6. Configure the Credentials page.
   * Enter an admin password that will be set on the Harbor service itself.
7. Configure the Authentication Page. This enables us to integrate Harbor with either our PKS users, PAS users, an external LDAP service, or just use local auth.

* Select Internal as we will be using local auth.

1. Click the Installation Dashboard link to return to the Installation Dashboard.
   * *Notice the tile has changed form orange to green signifying it it ready to be applied!*
2. Click Apply Changes on the right navigation.

**Configure Harbor DNS**

1. **IT FAILED**...But thats expected. The first deployment attempt we did not know the IP that BOSH would assign our Harbor VM. Therefore we did not setup the DNS resolution. Lets set that up now.
2. Navigate to the Pivotal Ops Manager FQDN.
3. Login.
4. Click on the VMware Harbor Tile.
5. Click the tab labeled status. Here is the list of VMs deployed by the platform and the current status.
6. Note the IP of the harbor-app job down, this is the Harbor VM.
7. Configure your dns to resolve the hostname we specified in the earlier steps to the above IP.

**Repository Management**

**Prerequisites**

* Deployed Pivotal Ops Manager (2.0+)
* Deployed Pivotal Director Tile
  + Post-deploy scripts enabled
* Deployed Pivotal Container Service Tile
* vmWare Harbor Installed
* Docker

**Goal**

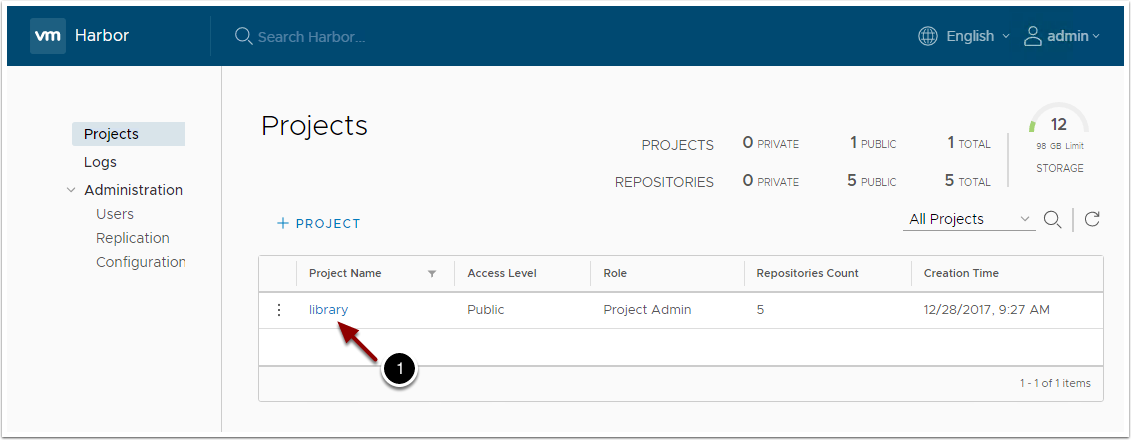
The application deployments in this workshop make use of a private container registry. We are using software from a VMware opensource project called Harbor as our registry. Harbor is included as an enterprise supported product with Pivotal Container Service (PKS). In this section, you will become familiar with the core capability of Harbor. You will create a project and see how to push and pull images from the repos. You will also enable content trust so that images are signed by the publisher and only signed images may be pulled from the project repo. You will also be introduced to the vulnerability scanning capability of Harbor. Most organizations will use a private registry rather than public Docker hub to improve security and latency for their applications. Although Harbor can be deployed as a highly available application, we have not done that for this workshop.

**Login to Harbor UI**

1. Click on Google Chrome
2. Navigate to the Harbor URL
3. Login to Harbor with the username and password

**View Projects and Repositories**

Harbor organizes images into a set of projects and repositories within those projects. Repositories can have one or more images associated with them. Each of the images are tagged. Projects can have RBAC (Role Based Access Control) and replication policies associated with them so that administrators can regulate access to images and create image distribution pipelines across registries that might be geographically dispersed. You should now be at a summary screen that shows all of the projects in this registry. There is only a single project called library.



The library project contains four repositories and has no access control. It is available to the public.

1. Click on library to see the repos

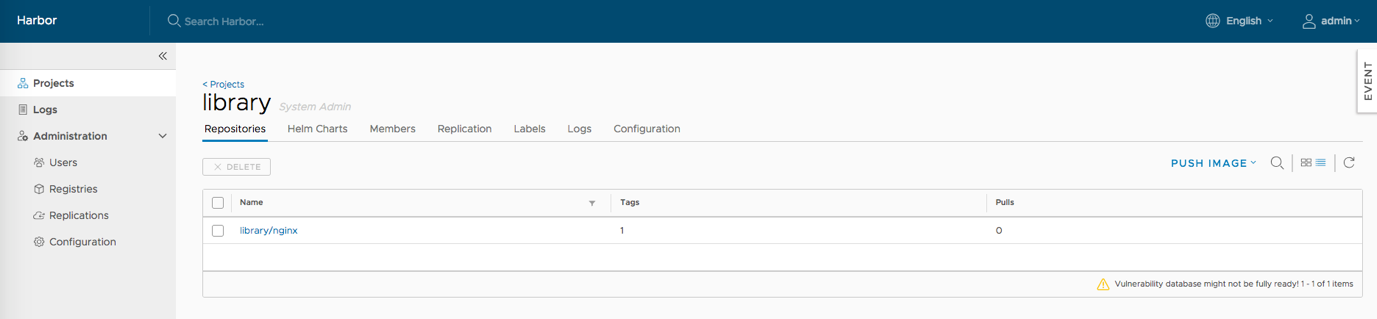
There are currently no repositories in the library, so let's add one using Docker.

**NOTE:** If you are using self-signed certs in Harbor, you will need to add an insecure registry to your local Docker daemon. Go to 'Docker -> Preferences -> Daemon' and add your registry to the list of insecure registries. Then click 'Apply & Restart'.

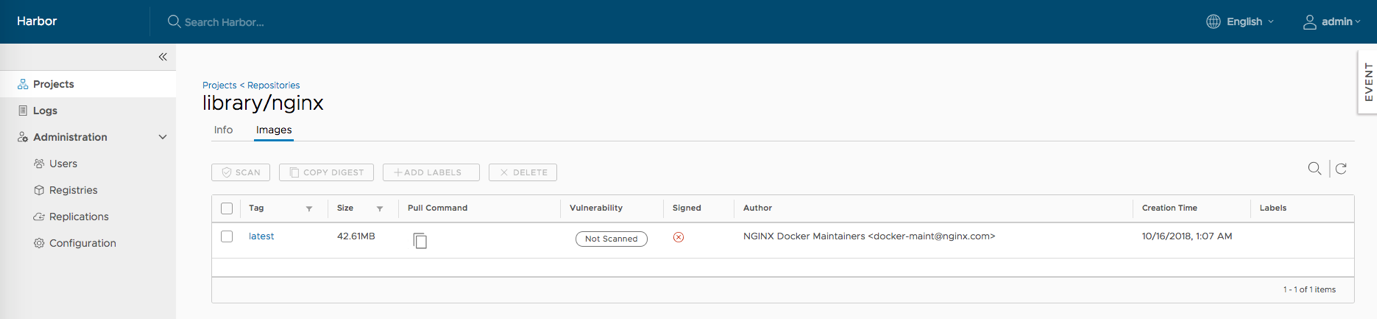
In a terminal, type:

1. docker pull nginx
2. docker tag nginx <YOUR\_HARBOR\_FQDN>/library/nginx
3. docker push <YOUR\_HARBOR\_FQDN>/library/nginx

Refresh Harbor and you should see your new repository.



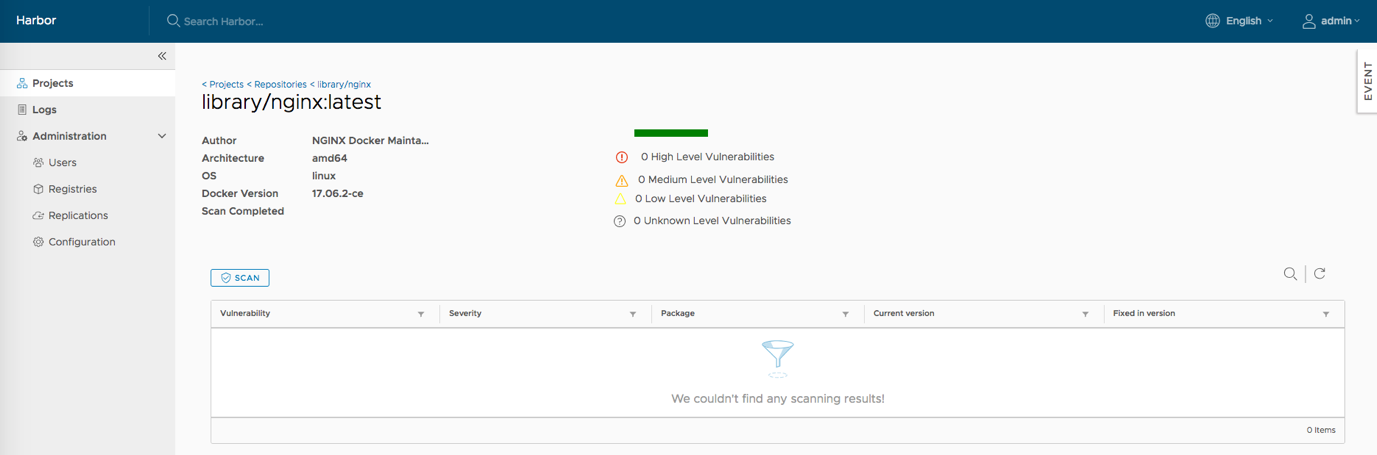
1. Click on the name of the repository library/nginx to view details
2. On the Images tab we can see that this image is unsigned and has not yet been scanned for vulnerabilities.



**Vulnerability Scans**

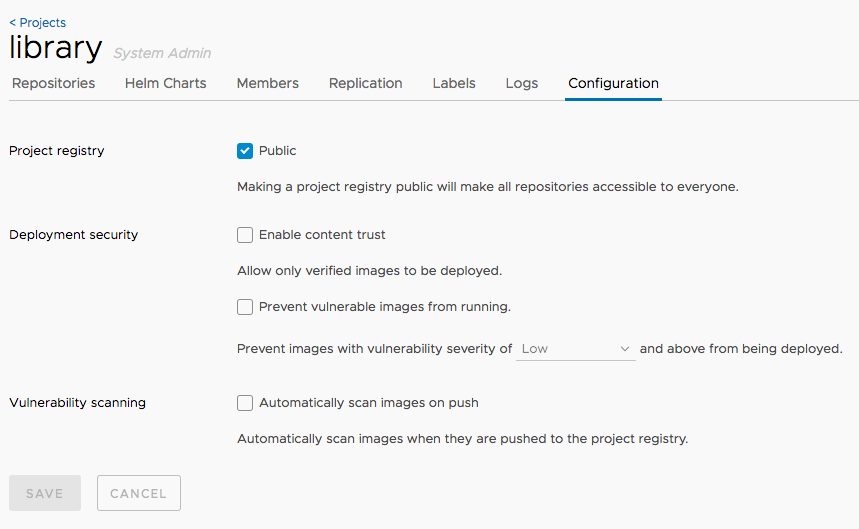
In order to view an image vulnerability report, we must first perform an image scan.

1. Click on the tag latest to drill down to the vulnerability report page
2. Click on the Scan button to perform an image scan. You should see the progress change from Queued to Scanning.
3. After the scan is complete, you should see a full report with no vulnerabilities! Our image is now safe to deploy to Kubernetes.



**Configuration Options**

There are additional security policies that can be enforced from Harbor through configurations.



1. Click on the default project: Projects -> library
2. Click on the Configuration tab
3. Here you will see several options for additional security implementation:
   * **Project Registry:** Toggling the Public checkbox will allow access to everyone or keep the registry private.
   * **Deployment Security:**
     + **Enable Content Trust:** Allows only signed images to be deployed.
     + **Prevent Vulnerable Images From Running:** Images with vulnerabilities cannot be run. Severity tolerance can be set as well.
   * **Vulnerability Scanning:** Set this value to automatically scan images when they are pushed to the registry.