

# Tanzu Lab Manual

LAB Access: .....	2
TKG:.....	2
Cluster Life-Cycle Management (Cluster API).....	2
Login to a management cluster .....	2
Provision TKG Cluster.....	3
Manage your workload cluster with Tanzu Mission Control.....	4
Scale workload cluster .....	5
Upgrade workload cluster .....	5
Push / Pull images to private registry .....	5
TMC:.....	6
TSM: .....	7
Deploy to cluster1:.....	7
Deploy to Cluster 2: .....	9
Create GNS.....	9

## LAB Access:

Ubuntu VM SSH:  
10.254.202.10  
Ubuntu\WWTwwt1!

VCenter login:

poc6207-vc01.wwtpoc.local  
10.254.176.150  
Administrator@vsphere.local\WWTatc123!

### TKG-VIP

Network: 10.254.202.160/27 \* VLAN ID: 470  
Default Gateway: 10.254.202.161 Type: ATC Core Hosted

### TKG-MGMT

Network: 10.254.207.0/24 \* VLAN ID: 471  
Default Gateway: 10.254.207.1 Type: ATC Core Hosted

DHCP Scope: 10.254.207.41-252

### TKG-WKLD

Network: 10.254.208.0/24 \* VLAN ID: 472  
Default Gateway: 10.254.208.1 Type: ATC Core Hosted

DHCP Scope: 10.254.208.41-252

## TKG:

**Open Two putty sessions. Once you have two terminals, export this env variables.  
export TANZU\_CLI\_PINNIPED\_AUTH\_LOGIN\_SKIP\_BROWSER=true**

## Cluster Life-Cycle Management (Cluster API)

Login to a management cluster

tanzu login

```
ubuntu@ubuntu:~/TSM/acme_fitness_demo/kubernetes-manifests$ tanzu login
? Select a server [Use arrows to move, type to filter]
  tkg-mgmt          ()
> tkg-mgmt-ha       ()
+ new server
```

**tanzu cluster list --include-management-cluster**

```
ubuntu@ubuntu:~/TSM/acme_fitness_demo/kubernetes-manifests$ tanzu cluster list --include-management-cluster
```

NAME	NAMESPACE	STATUS	CONTROLPLANE	WORKERS	KUBERNETES	ROLES	PLAN
tkg-acmefitness-2	default	running	1/1	3/3	v1.20.5+vmware.2	<none>	dev
tkg-cluster-acmefitness-1	default	running	1/1	3/3	v1.20.5+vmware.2	<none>	dev
tkg-services	default	running	3/3	3/3	v1.20.5+vmware.2	tanzu-services	prod
tkg-with-harbor	default	running	1/1	1/1	v1.20.5+vmware.2	<none>	dev
tkg-mgmt-ha	tkg-system	running	3/3	1/1	v1.20.5+vmware.2	management	prod

**Get admin kubeconfig for a specific workload cluster. In this case its tkg-with-harbor**

**tanzu cluster kubeconfig get tkg-with-harbor --admin**

**kubectl config use-context tkg-with-harbor-admin@tkg-with-harbor**

```
ubuntu@ubuntu:~/TSM/acme_fitness_demo/kubernetes-manifests$ tanzu cluster kubeconfig get tkg-with-harbor --admin
Credentials of cluster 'tkg-with-harbor' have been saved
You can now access the cluster by running 'kubectl config use-context tkg-with-harbor-admin@tkg-with-harbor'
```

Provision TKG Cluster

**cd ~/.tanzu/tkg/clusterconfigs/**

**cp tkg-acmefitenss-cluster-2.yaml tkg-cluster-1-<yourname>.yaml**

**Vi tkg-cluster-1-<yourname>.yaml**

Change clutser name:

**CLUSTER\_NAME: tkg-cluster-1-<yourname>**

Provide a static ip to workload control plane node: (make sure that this ip is not in use or someone is not planning to use. Start with 10.254.208.15

VSPHERE\_CONTROL\_PLANE\_ENDPOINT: 10.254.208.15

```
tanzu cluster create --file tkg-cluster-1-<yourname>.yaml --tkr v1.19.9---vmware.2-tkg.1
```

```
tanzu cluster kubeconfig get tkg-cluster-1-<yourname> --export-file tkg-cluster-1-<yourname>.cred
```

```
Kubectl get ns --kubeconfig tkg-cluster-1-<yourname>.cred
```

This would be an OIDC enabled cluster. Since this ubuntu machine does not have a browser, it will give you a url that you need to paste in browser in windows jump box and enter credentials.

```
Please log in: https://10.254.207.10:31234/oauth2/authorize?access_type=offline&client_id=127.0.0.1&redirect_uri=http%3A%2F%2F127.0.0.1%3A38753%2Fcallback&response_type=code&scope=offline_access
^C [0625 16:26:31.430758 1268984 transport.go:260] Unable to cancel request for *exec.roundtrip
```

Alana\VMware1VMware1

Once authenticated, copy the localhost url from browser and paste it in another putty session. This way you would be able to login to workload cluster.

```
- -> 127.0.0.1/callback?error=invalid_request&error_description=The+request+is+missing+a+required+parameter%2C+includes+an+invalid+parameter+value%2C+ii
```

Curl -L 'url'

## Manage your workload cluster with Tanzu Mission Control.

You will not be able to list resources since there are no role bindings for user alana. Let's create a role binding by adding cluster to TMC.

- In the left navigation pane of the Tanzu Mission Control console, click Administration, and then click the Management clusters tab.
- In the table of management clusters, click the management cluster that contains the workload cluster you want to add.
- On the management cluster detail page, click the Workload clusters tab.
- In the list of workload clusters, select the clusters you want to add by clicking the checkbox next to the name, and then click Manage # Cluster(s).
- In the confirmation dialog, select the cluster group(pgrdveops) to which you want to add the clusters, and then click Manage.

## Scale workload cluster

```
tanzu cluster scale tkg-cluster-1-<yourname> --worker-machine-count=3
```

## Upgrade workload cluster

```
tanzu kubernetes-release get
```

```
tanzu cluster upgrade tkg-cluster-1-<yourname>
```

## Push / Pull images to private registry

Harbor is already setup as an in-cluster extension in shared services cluster. Let's inspect shared services. Cluster.

```
kubectl config use-context tkg-services-admin@tkg-services
```

```
kubectl get po -n tanzu-system-registry
```

```
ubuntu@ubuntu:~/tanzu/tkg/clusterconfigs$ kubectl get po -n tanzu-system-registry
NAME                                READY   STATUS    RESTARTS   AGE
harbor-clair-6b45ccc6d7-9ncc2       2/2     Running   532        3d16h
harbor-core-7d7bfdffd4-gtn9c        1/1     Running   0          3d16h
harbor-database-0                   1/1     Running   0          3d16h
harbor-jobservice-764dcf7b69-wl87l   1/1     Running   0          3d16h
harbor-notary-server-747bcdcdcb-5h7dv 1/1     Running   0          3d16h
harbor-notary-signer-9db7d5984-mzfj7 1/1     Running   0          3d16h
harbor-portal-cb5756ffd-7hd2n        1/1     Running   0          3d16h
harbor-redis-0                      1/1     Running   0          3d16h
harbor-registry-7847bc96fd-b9224     2/2     Running   0          3d16h
harbor-trivy-0                       1/1     Running   0          3d16h
```

```
docker login pgrharbor.wwtpoc.local -u admin
```

## TMC:

TMC runs pinniped on all clusters and configures it with a webhook which points to a TMC backend endpoint. The endpoint is just used by pinniped to validate the incoming opaque token. Kubeconfig provided by TMC uses the tmc CLI to perform the credential exchange. When the user issues a kubectl command using the kubeconfig, this is the high level flow that gets triggered-

- tmc CLI calls TMC backend to fetch a user cluster opaque token in exchange for CSP access + ID token
- CLI calls the tokencredentialrequest endpoint on pinniped passing in the user opaque token
- Pinniped uses the webhook endpoint to validate the incoming token.
- Once validated, pinniped generates a cert + key pair and returns that to the tmc CLI
- The tmc CLI uses that as the ExecCredential response to kubectl which allows kubectl to use the credentials to authenticate all outgoing calls to the apiserver

Follow this git repo below.

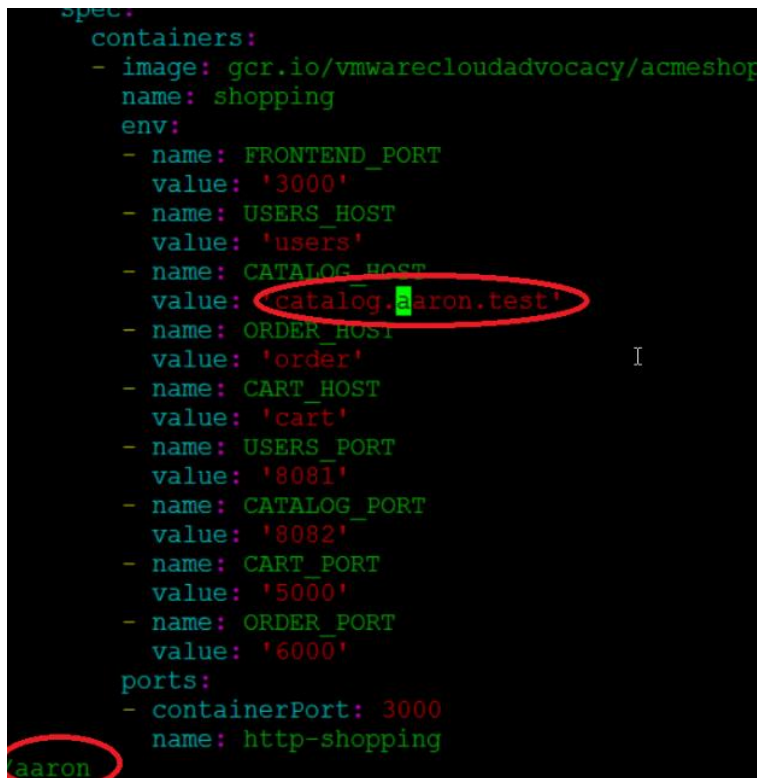
<https://github.com/bmullan-pivotal/tmc-policy-demonstration>

TSM:

Deploy to cluster1:

Change your kubeconfig context to your cluster 1.

- Deploy the Istio gateways - “kubectl apply -f acme\_fitness\_demo/istio-manifests/gateway.yaml”
- Deploy the secrets – “kubectl apply -f acme\_fitness\_demo/kubernetes-manifests/secrets.yaml”
- Before deploying the application itself, we need to change the YAML file to indicate what will be the name of the Catalog service the Frontend will try to reach. Select a GNS domains name (e.g. gntest.local)
- Edit the manifest with “vim acme\_fitness\_demo/kubernetes-manifests/acme\_fitness\_cluster1.yaml” and search for “catalog” (in VIM you should press “/catalog”), you should see the placeholder name of GNS like this:



```
spec:
  containers:
  - image: gcr.io/vmwarecloudadvocacy/acmeshop
    name: shopping
    env:
    - name: FRONTEND_PORT
      value: '3000'
    - name: USERS_HOST
      value: 'users'
    - name: CATALOG_HOST
      value: 'catalog.aaron.test'
    - name: ORDER_HOST
      value: 'order'
    - name: CART_HOST
      value: 'cart'
    - name: USERS_PORT
      value: '8081'
    - name: CATALOG_PORT
      value: '8082'
    - name: CART_PORT
      value: '5000'
    - name: ORDER_PORT
      value: '6000'
    ports:
    - containerPort: 3000
      name: http-shopping
```

aaron

Change the suffix to your GNS domain name of choosing (e.g. acmegns.local) and save the file

```
spec:
  containers:
  - image: gcr.io/vmwarecloudad
    name: shopping
    env:
    - name: FRONTEND_PORT
      value: '3000'
    - name: USERS_HOST
      value: 'users'
    - name: CATALOG_HOST
      value: 'catalog.acmegns.1
    - name: ORDER_HOST
      value: 'order'
    - name: CART_HOST
      value: 'cart'
    - name: USERS_PORT
      value: '8081'
    - name: CATALOG_PORT
      value: '8082'
    - name: CART_PORT
      value: '5000'
    - name: ORDER_PORT
      value: '6000'
    ports:
    - containerPort: 3000
      name: http-shopping
---
apiVersion: v1
kind: Service
metadata:
  name: shopping
  labels:
    app: shopping
```

- Now you can deploy the cluster1 services using “kubectl apply -f acme\_fitness\_demo/kubernetes-manifests/acme\_fitness\_cluster1.yaml”
- Monitor the services until you see the following services up: watch kubectl get po

```
+ kubectl get pods
NAME                                READY   STATUS    RESTARTS
cart-74d99f6c4-kx42x                2/2     Running   0
cart-redis-7fcbbc4f64-ldpqf         2/2     Running   0
order-8f64cc8d-gdhx9                2/2     Running   0
order-mongo-994ddf9d4-h9kdd         2/2     Running   0
payment-6767b9444f-7bxbp           2/2     Running   0
shopping-74fd757986-nvpws           2/2     Running   0
users-66984ff6f4-88rdf              2/2     Running   0
users-mongo-7979895bf8-tm589        2/2     Running   0
```

- You can see that each pod has 2 containers, this shows that the Envoy proxy has been injected
  - At this point figure out the ingress IP to access the application (depending on your load balancing and CNI solution you should get a “public” IP to interact with the application. To do this run “ kubectl get services -n istio-system | grep istio-ingressgateway” The external IP will be the second
- Navigate using a browser to <http://<ingress IP>>, you should see the Acme store but without pictures of products (we haven’t connected the catalog service yet). Also if clicking on catalog on the top should not show any products



## Deploy to Cluster 2:

Change your kubeconfig context to cluster 2:

This cluster will run the catalog service and its catalog database. We will connect The frontend on K8s1 to the catalog on K8s2 in the following step. To deploy after you labeled the Default namespace with Istio injection run:

- Deploy the Istio gateways - “`kubectl apply -f acme_fitness_demo/istio-manifests/gateway.yaml`”
- Deploy the secrets – “`kubectl apply -f acme_fitness_demo/kubernetes-manifests/secrets.yaml`”
- Deploy the cluster2 services “`kubectl apply -f acme_fitness_demo/kubernetes-manifests/acme_fitness_cluster2.yaml`”

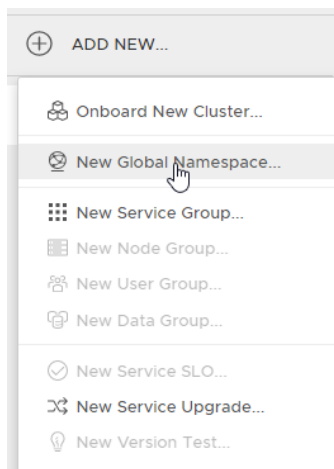
Monitor the services until you see the following services running:

```
+ kubectl get pods
NAME                                READY   STATUS    RESTARTS
catalog-69bd59b758-x8vzv            2/2     Running   0
catalog-mongo-646ffbcf9d-9lggz      2/2     Running   0
```

## Create GNS

At this point we will create the GNS, once it is created and the virtualservices and the DNS entries are deployed, the Frontend on cluster K8s1 will know how to reach the Catalog on K8s2.

- In the console of NSX-SM, click on “Add new” and “New Global Namespace”



- Give the new GNS a name (in all small letters), in this example we use acmegns, and a domain name, which is the one you configured in a previous step on the “Cluster1” manifest. In our example it is acmegns.local

Global Namespace

General Details
Service Mapping
Public Services
Health Checks
Configuration Summary

### 1. General Details

\* indicates required field

GNS Name \*
pgr-gns
2-30 characters (a-z, 0-9, -).

Description (optional)
Optional

Color (optional)

Enter the domain name for the GNS.

Domain \*
acmegns.local
Services will be available at: acmegns.local

CANCEL

NEXT

- In the next screen select the “default” namespace on each cluster, check that the system detects the services by expanding the “Service “review” section

Global Namespace

General Details
Service Mapping
Public Services
Health Checks
Configuration Summary

### 2. Service Mapping

Set up mapping rules to define the services in this global namespace. You can select multiple namespaces across multiple clusters in different geographic locations, e.g. "dev" namespaces in two public clouds and on-prem clusters.

Conditions for Service Mapping Rules
Kubernetes namespaces with different names cannot be mapped to a single Global Namespace. A specific Kubernetes namespace can only be mapped to a single Global Namespace.

Mapping Rules

tkg-acmefitness-1
default

Conditions
Enter String

All Cluster
All Namespaces

Rule: Map services in namespace(s) default in cluster(s) tkg-acmefitness-1

> SERVICE PREVIEW

Map services in Kubernetes Namespace(s)
Specify a pair of namespace and cluster names ⓘ

tkg-acmefitness-2
default

Conditions
Enter String

All Cluster
All Namespaces

Rule: Map services in namespace(s) default in cluster(s) tkg-acmefitness-2

CANCEL

BACK

NEXT

Click “add service mapping” to add all 3 clusters and then click “next”

- In “Public services” keep the default and click next

Global Namespace

General Details

Service Mapping

Public Services

Health Checks

Configuration Summary

3. Public Services (Optional)

Define the services available to the public, e.g. via a public URL.

☒ No Public Services

☐ Configure Public Service/s

CANCEL

BACK

NEXT

- Health Check optional

Global Namespace

General Details

Service Mapping

Public Services

Health Checks

Configuration Summary

4. Health Checks (Optional)

Define health checks for services in this GNS.

☒ No Service Health Checks

☐ Configure Service Health Checks

CANCEL

BACK

NEXT

- Click “Finish”
- A message should pop up that the GNS was created successfully

To make sure the GNS is being created correctly run the command “kubectl get virtualservices”

```
kubo@jumper:~$ kubectl get virtualservices
NAME                                GATEWAYS                HOSTS                                AGE
acme                                [acme-gateway]          [*]                                 28h
nsxsm.acmegns.cart                  [cart.acmegns.local]    2m40s
nsxsm.acmegns.cart-redis             [cart-redis.acmegns.local] 67s
nsxsm.acmegns.catalog               [catalog.acmegns.local]   48s
nsxsm.acmegns.catalog-mongo         [catalog-mongo.acmegns.local] 45s
nsxsm.acmegns.kubernetes            [kubernetes.acmegns.local] 42s
nsxsm.acmegns.order                 [order.acmegns.local]    2m35s
nsxsm.acmegns.order-mongo           [order-mongo.acmegns.local] 2m3s
nsxsm.acmegns.payment               [payment.acmegns.local] 100s
nsxsm.acmegns.shopping              [shopping.acmegns.local] 2m28s
nsxsm.acmegns.users                 [users.acmegns.local]    109s
```

To test now access the application again at <http://<ingress ip>> now you should see the catalog of products

