VMware Tanzu™ COMMUNITY EDITION

ABSTRACT
Freely available open-source Kubernetes platform suited for both testing and production purposes

Ryan Ang

Table of Contents

Tanzu Community Edition (TCE)

What is it

VMware Tanzu Community Edition (TCE) is a full-featured, implementation-friendly Kubernetes platform for both learners and users. It is a freely available, community supported, open-source distribution of VMware Tanzu that can be easily installed and configured on your local workstation or cloud.

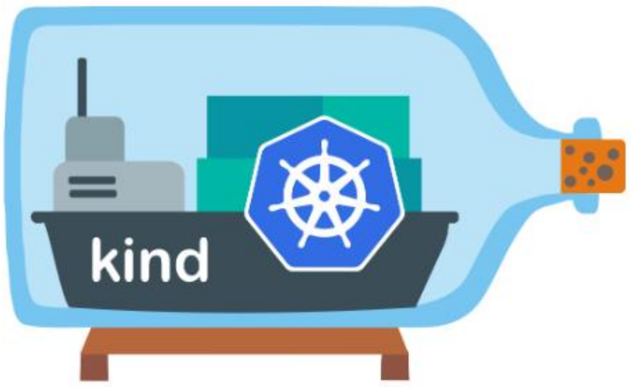
Different from other Tanzu editions, Tanzu Community Edition is freely available, and community supported, whereas other editions of it are strictly commercially supported. In addition, Tanzu Community Edition is also an open-source software that you can run and manage yourself, so it gives you some level of management and observability capabilities via software that it can manage while its counterpart editions only provide these capabilities as VMware managed services. Conversely, open-source software is used in Tanzu Community Edition to provide certain networking capabilities that Standard and Advanced editions extend with advanced proprietary software.

Who should use it

Tanzu Community Edition is for anyone who is keen to learn about, evaluate, or use Tanzu software in a community-supported environment. Regardless if you need to build practical cloud native skills, run a Kubernetes proof-of-concept, do inner loop development work on your local workstation, or simply just exploring out of curiosity and interest, Tanzu Community Edition is the perfect platform for you to start your endeavour.

Fundamentals of Kubernetes in TCE

Kubernetes in Dockers (Kind)



Before getting started, it is important to realise that Tanzu Kubernetes runs on “Kind” – Kubernetes in Dockers. Unlike Minikube, which allows its users to only provision one node at a time, Kind is a tool for running local Kubernetes clusters using Docker container “nodes”, and hence, multiple nodes can be run locally. Initially, Kind was primarily designed for testing Kubernetes itself, but now, it can also be used for local development or CI.

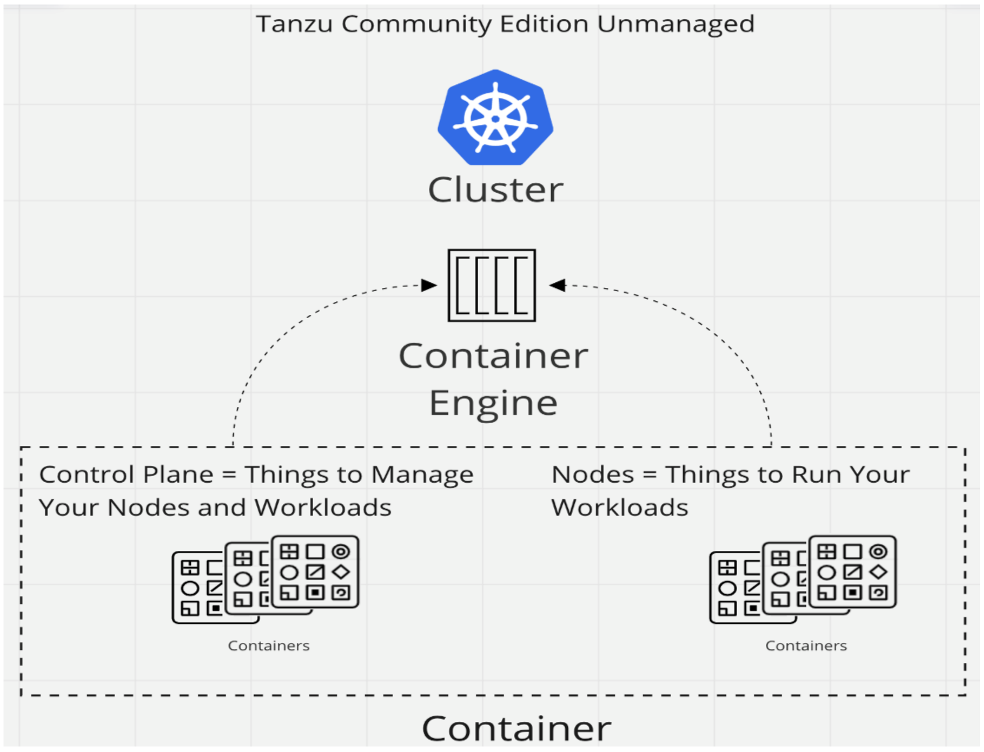
Essentially, everything that you will see in the later sections such as management clusters, control planes and worker nodes, are in fact Docker containers. In other words, Tanzu Community Edition users can access their clusters via docker commands. For instance, “docker exec -it …” to run commands in cluster nodes.

Types of TCE Kubernetes Clusters

When provisioning Kubernetes clusters in TCE, the users are presented with **two** options: (1) **Unmanaged Clusters** and (2) **Managed Clusters**

Unmanaged Clusters

An unmanaged cluster is simply a basic Kubernetes cluster spun up using tools like kubeadm. In the case of Tanzu Community Edition, all you need is a single command with the Tanzu Command Line Interface (CLI), and it will provide you a *Kind* cluster on your local machine. As mentioned in the previous section, what resides inside this cluster, including its nodes, are containers running in your local container engine, such as Docker Desktop if you are operating on Windows OS.


Tanzu Community Edition Unmanaged
Cluster
Container Engine
Control Plane = Things to Manage Your Nodes and Workloads
Nodes = Things to Run Your Workloads
Containers
Containers
Container

Some advantages to using TCE unmanaged clusters:

1. Kind clusters start a root container and then bring up all the pieces of the cluster inside that root container. Hence, it allows for a fast start-up and tear down since it is just a single container.

2. Relatively small resource usage because only a single "cluster" and containers are provisioned. Given the efficient nature of containers we will only use the memory needed to run their services.
3. All the components within the clusters are containers running in the root container, making clean up as easy as deleting a single container.

Some drawbacks to using unmanaged clusters:

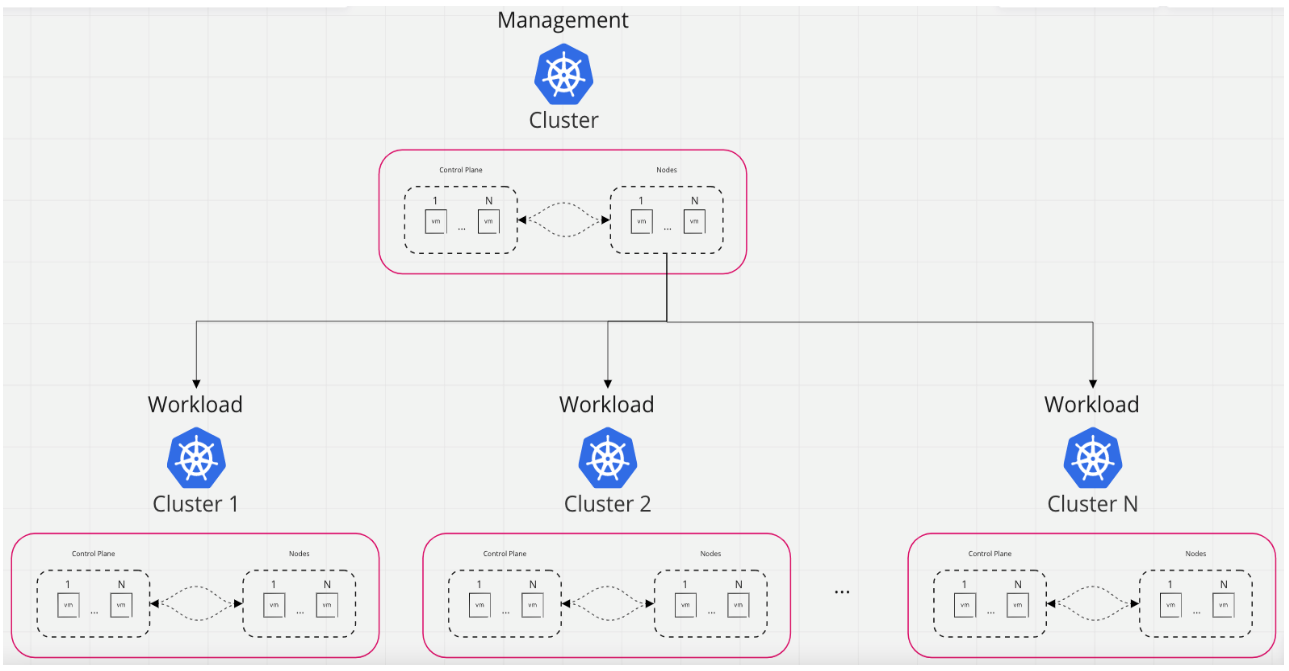
1. Unmanaged clusters are fundamentally different in several ways from managed clusters, and therefore will not be an exact representation of managed development, staging, or production clusters.
2. Currently, the clusters can only run on your local machine. Therefore, if a container engine is absent on your machine, or your machine does not have enough resources, this setup will not be feasible.

All in all, unmanaged clusters are a good start for learning the fundamentals of Kubernetes and Tanzu. It is a quick way to get started, and should you mess up your installation, it is not a lot of work to simply delete and recreate the cluster. However, what if you want a Kubernetes cluster that is production-ready? This is where TCE managed clusters will help you with that.

Managed Clusters

Managed clusters brings all the goodness we get from declarative infrastructure, Kubernetes, Custom Resource Definitions, and the Kubernetes Cluster API to allow you to use Kubernetes clusters to manage the clusters. In essence, we provision one TCE management cluster, and then through that cluster we create and manage other clusters, also known as workload clusters, where all the applications are executed.

The management cluster uses the **Kubernetes Cluster API** to oversee the full lifecycle of other workload clusters. A workload cluster is where you run all your applications, databases, and other containerized resources. Usually, the management cluster is on separate machines/infrastructure from any of the workload clusters as its main task is only to oversee and manage them. However, it, too, can be provisioned on the same machines where the workload clusters are running on.



Some <u>advantages</u> to using TCE managed clusters:

1. The control plane (master node) and worker nodes can actually exist on multiple infrastructure nodes. Because of this, you can achieve true HA and scalability.

2. There are simple, easy commands to create workload clusters.

3. As the management cluster manages all your workload clusters, you now have a centralized place where you can access, query, and manage clusters.

4. Cluster creation can be done *declaratively*. You state what resources you want available in your Workload cluster, and the management cluster automatically takes care of provisioning them.

Just like unmanaged clusters, managed clusters, too, have some disadvantages:

1. The management cluster is one more cluster to manage, support, and incur costs for the infrastructure. Even if management clusters are usually relatively small, the overhead still exists.
2. It will take longer to provision a workload cluster, especially when starting without a management cluster, than it will to deploy an unmanaged cluster. Even if a management cluster already exists, it will still take longer because you need to provision infrastructure in a cloud provider, which is slower than spinning up containers on your local machine.

In summary, while there are drawbacks to them, managed clusters should definitely be your go-to if you are looking for Kubernetes clusters that are production-ready; Scalable, able to self-heal, and reliable.

Creating TCE Clusters

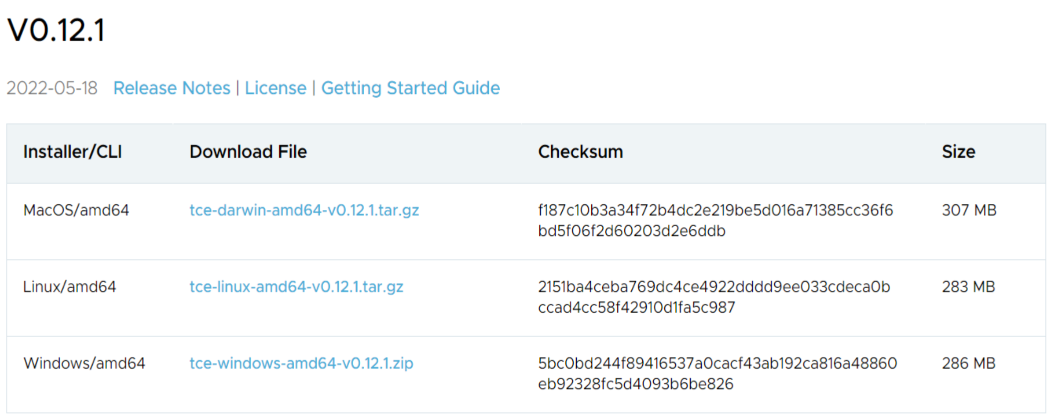
Now that we have explored the two main types of clusters that TCE provides, we will look at creating them. However, first and foremost, you have to decide, which cluster best suits your desired outcome? Do you want a single node, local workstation cluster suitable for a development/test environment? Or do you want a full-featured, scalable Kubernetes implementation suitable for a development or production environment? If it is the former, unmanaged clusters will be your best bet, otherwise, you are looking at managed clusters. Below are instructions that you can follow to create both managed and unmanaged clusters.

Take Note: The instructions below assume you have already installed Docker on your OS. If you are on Linux/Ubuntu, you can simply install Docker by running the command “sudo apt install docker.io” in a terminal, and if you are on Windows, simply download Docker Desktop together with its WSL 2 Backend.

Downloading Tanzu Community Edition

Before we can even create TCE clusters, we have to install TCE Command Line Interface (CLI):

1. Head over to this link: https://tanzucommunityedition.io/download/, and you should be seeing the page below:



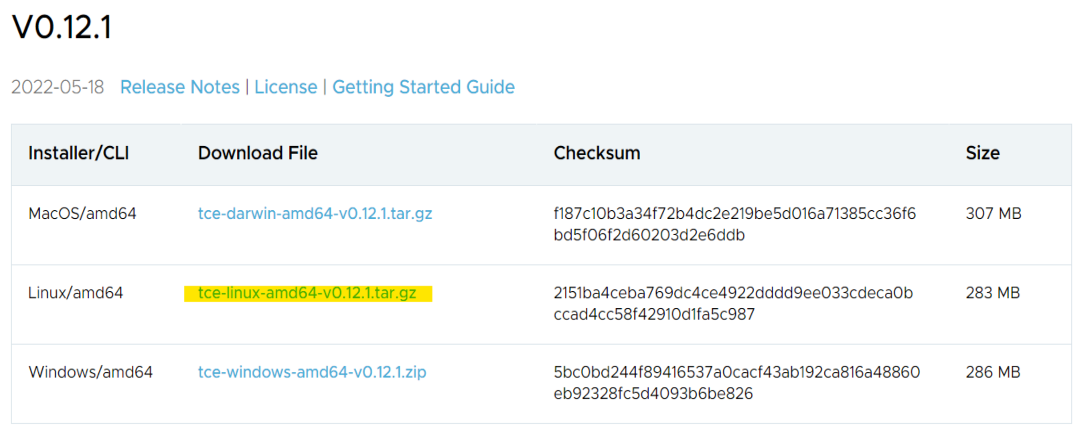
V0.12.1

2022-05-18 Release Notes | License | Getting Started Guide

Installer/CLI	Download File	Checksum	Size
MacOS/amd64	tce-darwin-amd64-v0.12.1.tar.gz	f187c10b3a34f72b4dc2e219be5d016a71385cc36f6 bd5f06f2d60203d2e6ddb	307 MB
Linux/amd64	tce-linux-amd64-v0.12.1.tar.gz	2151ba4ceba769dc4ce4922dddd9ee033cdeca0b ccad4cc58f42910d1fa5c987	283 MB
Windows/amd64	tce-windows-amd64-v0.12.1.zip	5bc0bd244f89416537a0cacf43ab192ca816a48860 eb92328fc5d4093b6be826	286 MB

At the time of producing this document, the TCE CLI version is V0.12.1, and therefore, the CLI that you are about to install might be of a different version.

2. For the purpose of this demonstration, we will assume that a Ubuntu 20.04LTS OS is used to run TCE. Therefore, select the link highlighted in yellow below (if you are operating on Windows/Mac OS, then select the link either above, or below, the highlighted one):

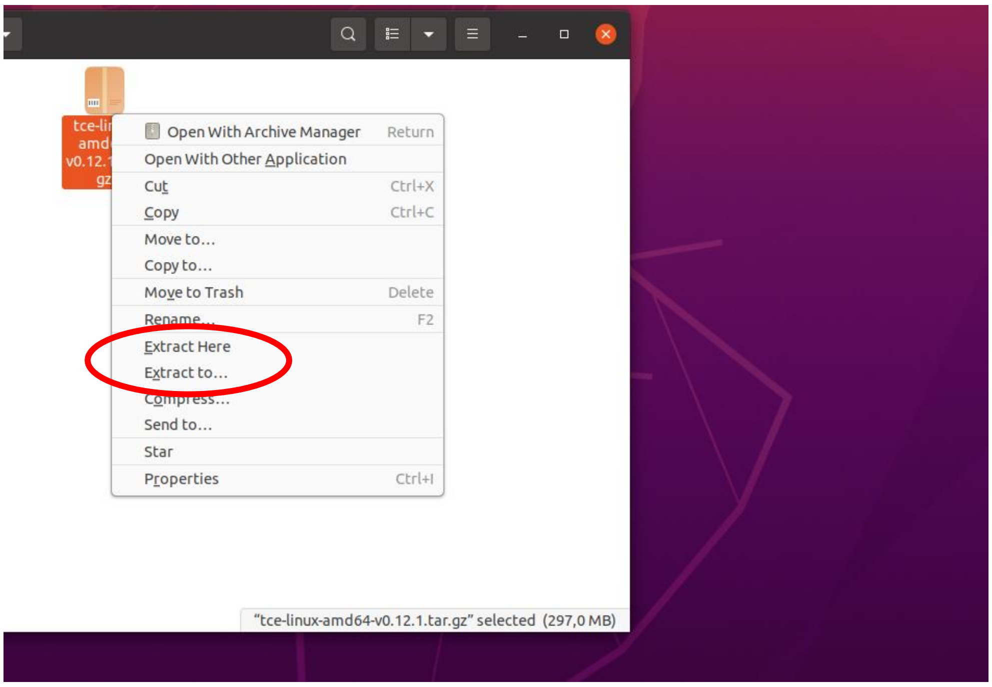


V0.12.1

2022-05-18 Release Notes | License | Getting Started Guide

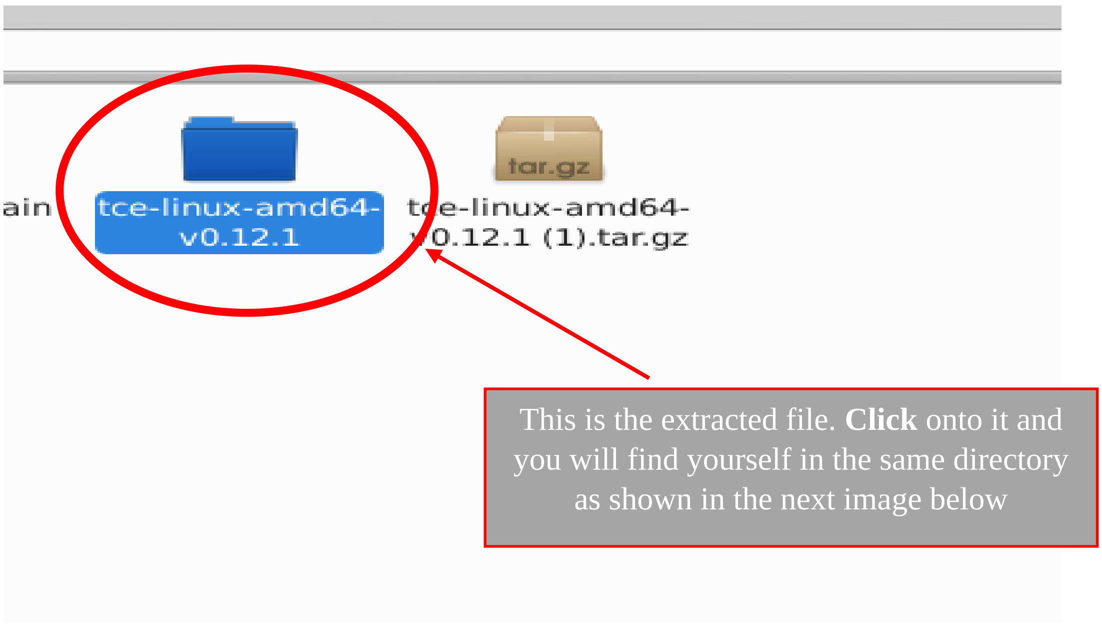
Installer/CLI	Download File	Checksum	Size
MacOS/amd64	tce-darwin-amd64-v0.12.1.tar.gz	f187c10b3a34f72b4dc2e219be5d016a71385cc36f6bd5f06f2d60203d2e6ddb	307 MB
Linux/amd64	tce-linux-amd64-v0.12.1.tar.gz	2151ba4ceba769dc4ce4922dddd9ee033cdeca0bccad4cc58f42910d1fa5c987	283 MB
Windows/amd64	tce-windows-amd64-v0.12.1.zip	5bc0bd244f89416537a0cacf43ab192ca816a48860eb92328fc5d4093b6be826	286 MB

3. Upon selecting the link, a tar file should be downloaded onto your Ubuntu OS. Locate the file and extract its contents:

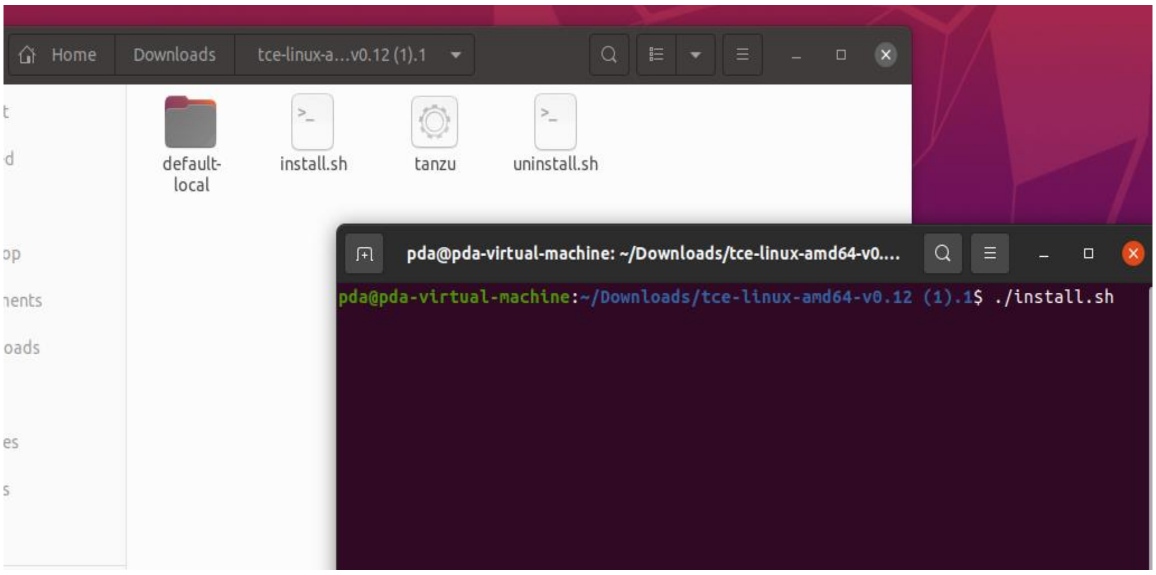


tce-linux-amd64-v0.12.1.tar.gz
Open With Archive Manager Return
Open With Other Application
Cut Ctrl+X
Copy Ctrl+C
Move to…
Copy to…
Move to Trash Delete
Rename… F2
Extract Here
Extract to…
Compress…
Send to…
Star
Properties Ctrl+I
"tce-linux-amd64-v0.12.1.tar.gz" selected (297,0 MB)

4. Once you have extracted the contents, open a terminal within the extracted folder, and run the command "./install.sh". The CLI will proceed to install on your OS, and it might take just a minute or two:

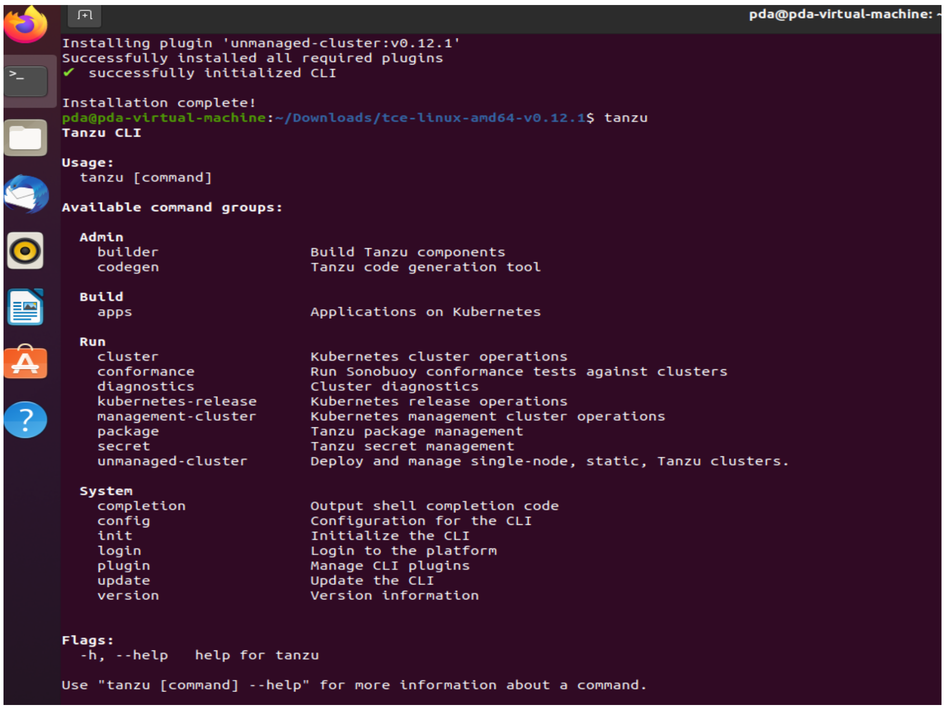


tce-linux-amd64-v0.12.1
tar.gz
tce-linux-amd64-v0.12.1 (1).tar.gz
This is the extracted file. **Click** onto it and you will find yourself in the same directory as shown in the next image below


Home
Downloads
tce-linux-a...v0.12 (1).1
default-local
install.sh
tanzu
uninstall.sh
pda@pda-virtual-machine: ~/Downloads/tce-linux-amd64-v0....

```
pda@pda-virtual-machine:~/Downloads/tce-linux-amd64-v0.12 (1).1$ ./install.sh
```

5. Once the CLI has been installed, check that the installation is indeed successful by running the command “tanzu” in your terminal. If the installation went right, you should see a display of all the tanzu commands in your terminal, like below:



```
Installing plugin 'unmanaged-cluster:v0.12.1'
Successfully installed all required plugins
✔  successfully initialized CLI

Installation complete!
pda@pda-virtual-machine:~/Downloads/tce-linux-amd64-v0.12.1$ tanzu
Tanzu CLI

Usage:
  tanzu [command]

Available command groups:

  Admin
    builder                 Build Tanzu components
    codegen                 Tanzu code generation tool

  Build
    apps                    Applications on Kubernetes

  Run
    cluster                 Kubernetes cluster operations
    conformance             Run Sonobuoy conformance tests against clusters
    diagnostics             Cluster diagnostics
    kubernetes-release      Kubernetes release operations
    management-cluster      Kubernetes management cluster operations
    package                 Tanzu package management
    secret                  Tanzu secret management
    unmanaged-cluster       Deploy and manage single-node, static, Tanzu clusters.

  System
    completion              Output shell completion code
    config                  Configuration for the CLI
    init                    Initialize the CLI
    login                   Login to the platform
    plugin                  Manage CLI plugins
    update                  Update the CLI
    version                 Version information


Flags:
  -h, --help   help for tanzu

Use "tanzu [command] --help" for more information about a command.
```

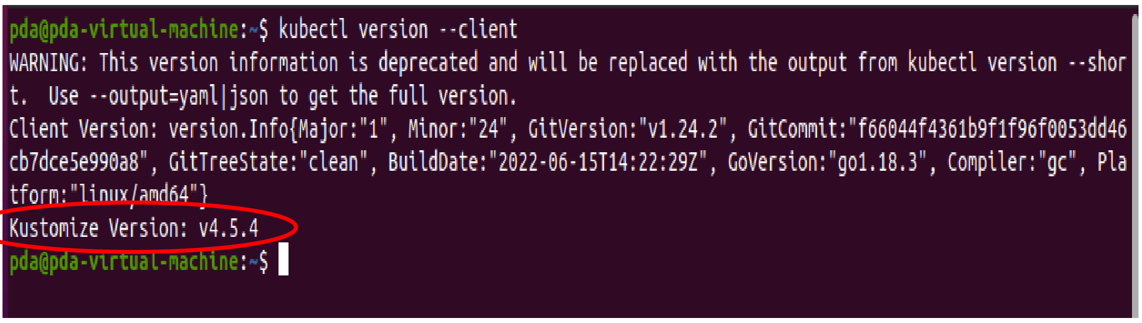
Downloading Kubectl

If you have not already installed Kubernetes command line tool, Kubectl, do click on this link: https://kubernetes.io/docs/tasks/tools/ and select the appropriate OS that you intend to have your cluster run on. Assuming that a Linux OS is used:

1. Open a terminal and run the command "curl -LO https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

2. After which, run the command "sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl"

3. Finally, to check that you have successfully installed kubectl, run the following command "kubectl version --client", and you should see the following output indicating the kubectl version installed:



```
pda@pda-virtual-machine:~$ kubectl version --client
WARNING: This version information is deprecated and will be replaced with the output from kubectl version --shor
t.  Use --output=yaml|json to get the full version.
Client Version: version.Info{Major:"1", Minor:"24", GitVersion:"v1.24.2", GitCommit:"f66044f4361b9f1f96f0053dd46
cb7dce5e990a8", GitTreeState:"clean", BuildDate:"2022-06-15T14:22:29Z", GoVersion:"go1.18.3", Compiler:"gc", Pla
tform:"linux/amd64"}
Kustomize Version: v4.5.4
pda@pda-virtual-machine:~$ 
```

Provisioning Unmanaged Clusters

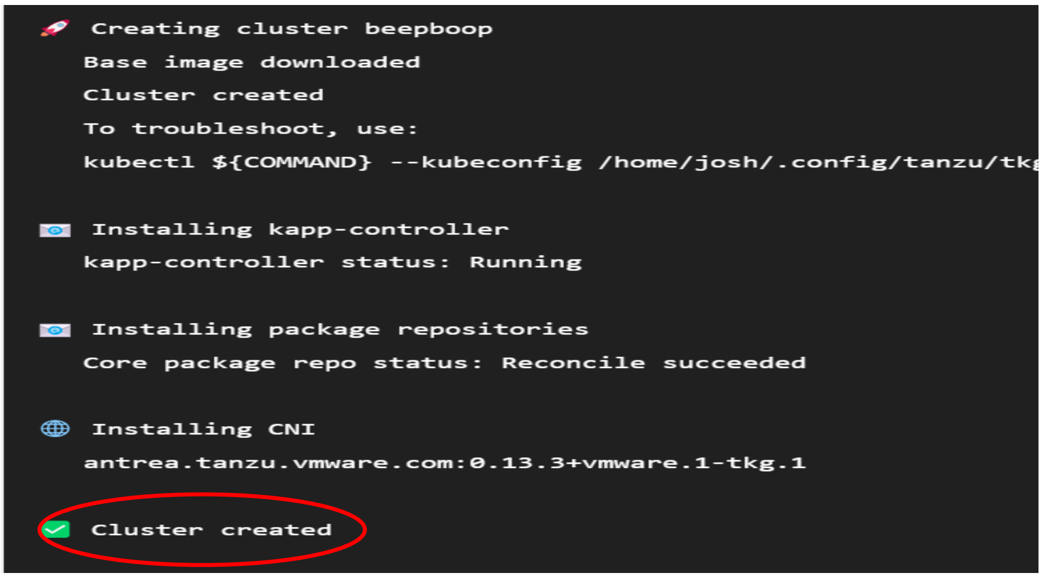
Let us look at how we can create an unmanaged cluster. The setup is very simple and should not take a long time to complete:

1. Open a new terminal, and in it, run the command "tanzu unmanaged-cluster create [CLUSTER NAME]", where [CLUSTER NAME] should be replaced by any name

that you intend to give to your unmanaged cluster. Below is an example of creating an unmanaged cluster named “beepboop”:

```
tanzu unmanaged-cluster create beepboop
```

2. After the unmanaged cluster has been created, your terminal will show you an output similar to the one below:



```
🚀 Creating cluster beepboop
   Base image downloaded
   Cluster created
   To troubleshoot, use:
   kubectl ${COMMAND} --kubeconfig /home/josh/.config/tanzu/tk

📧 Installing kapp-controller
   kapp-controller status: Running

📧 Installing package repositories
   Core package repo status: Reconcile succeeded

🌐 Installing CNI
   antrea.tanzu.vmware.com:0.13.3+vmware.1-tkg.1

✅ Cluster created
```

3. Check that the creation of the cluster is indeed successful by running the tanzu command “tanzu unmanaged-cluster list”. If the process went well, you should see the below output (following the above “beepboop” example):

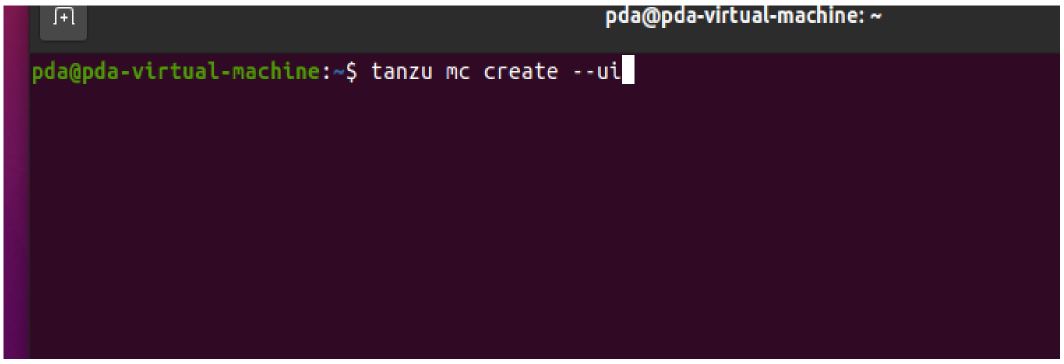
```
NAME      PROVIDER
beepboop  kind
```

4. Now that the unmanaged cluster has been provisioned, you can use kubectl commands to interact with it. For instance, running "kubectl get nodes" to check that you have both a control plane (master node) and a worker node where you can deploy workloads to it.

Provisioning Managed Clusters

Unlike unmanaged clusters, provisioning managed clusters requires more steps as you need to deploy the management and workload clusters. The instructions below assume you are running on Linux/Ubuntu OS:

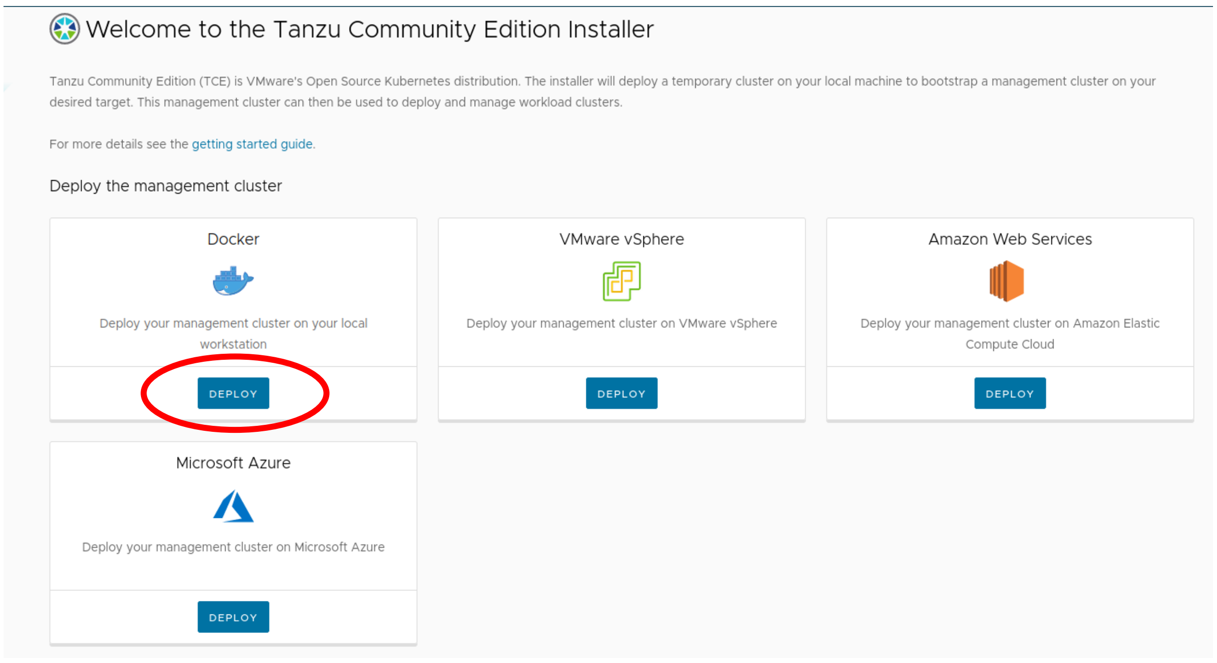
1. Open a terminal and run the following command "tanzu mc create --ui":



```
pda@pda-virtual-machine: ~
pda@pda-virtual-machine:~$ tanzu mc create --ui
```

2. The one amazing benefit you get when using TCE is that you can deploy a management cluster from a User Interface (UI) that comes with the installation of its CLI, instead of having to manually craft a YAML configuration file containing all the details of the cluster. TCE UI takes care of that for you.

   Upon running the command in the previous step, the UI will open up:


Welcome to the Tanzu Community Edition Installer

Tanzu Community Edition (TCE) is VMware's Open Source Kubernetes distribution. The installer will deploy a temporary cluster on your local machine to bootstrap a management cluster on your desired target. This management cluster can then be used to deploy and manage workload clusters.

For more details see the getting started guide.

Deploy the management cluster

Docker

Deploy your management cluster on your local workstation

VMware vSphere

Deploy your management cluster on VMware vSphere

Amazon Web Services

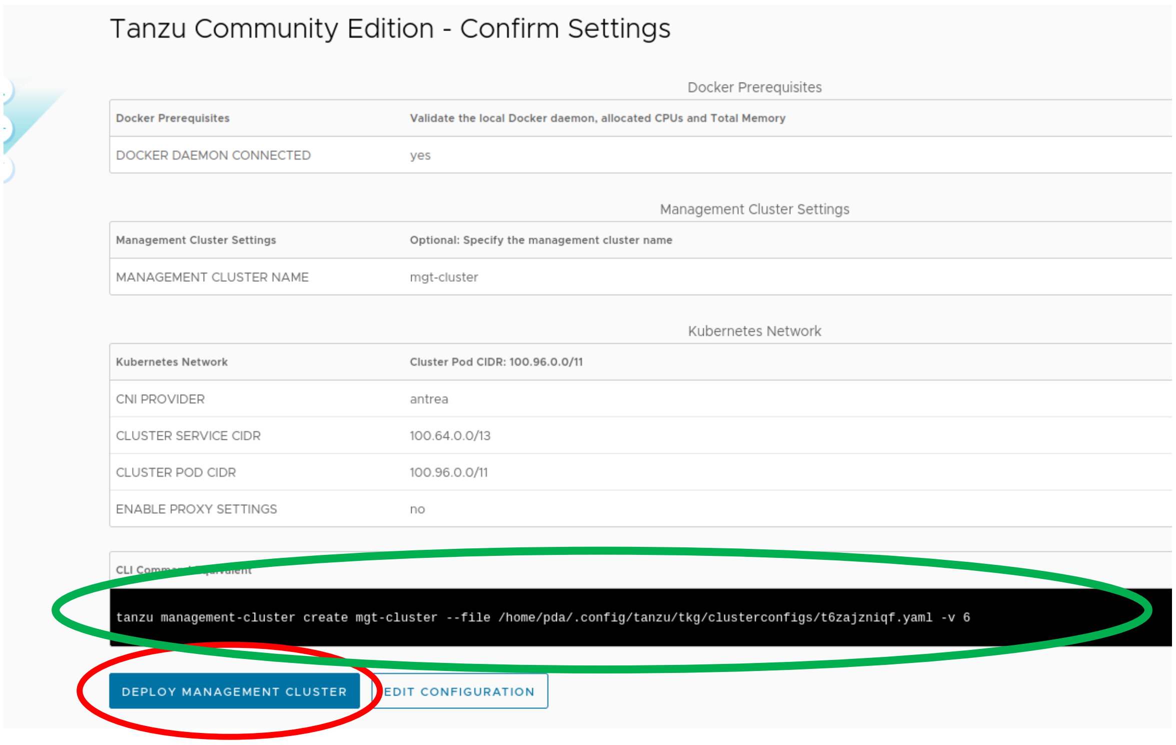
Deploy your management cluster on Amazon Elastic Compute Cloud

Microsoft Azure

Deploy your management cluster on Microsoft Azure

TCE gives you the option to deploy your management cluster on cloud platforms like Microsoft Azure and Amazon Web Services (AWS). Hence, select according to which cloud provider you are subscribed to. For the purpose of our demonstration, we will be deploying our management cluster onto Docker, or our local workstation, and hence, we will be selecting that "DEPLOY" button as seen in the image.

3. After which, you will have to fill in the necessary details that you want to have for your managed cluster. When all is done, you will be brought to the page below, which gives an overall summary of your soon-to-be-created management cluster. You can then click on the "DEPLOY MANAGEMENT CLUSTER" button:



Tanzu Community Edition - Confirm Settings

Docker Prerequisites

Docker Prerequisites	Validate the local Docker daemon, allocated CPUs and Total Memory
DOCKER DAEMON CONNECTED	yes

Management Cluster Settings

Management Cluster Settings	Optional: Specify the management cluster name
MANAGEMENT CLUSTER NAME	mgt-cluster

Kubernetes Network

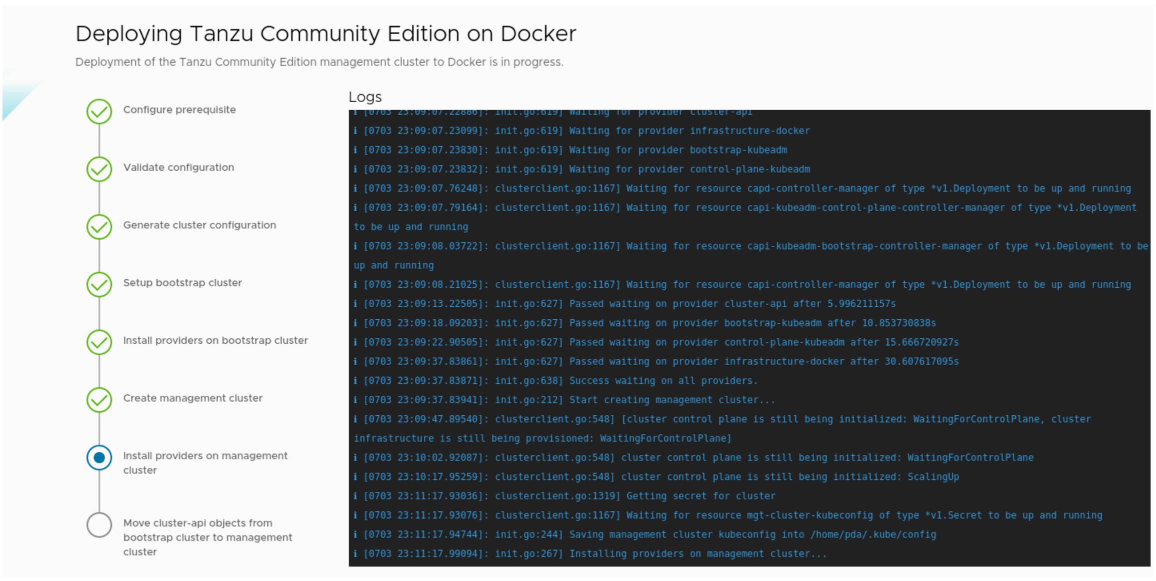
Kubernetes Network	Cluster Pod CIDR: 100.96.0.0/11
CNI PROVIDER	antrea
CLUSTER SERVICE CIDR	100.64.0.0/13
CLUSTER POD CIDR	100.96.0.0/11
ENABLE PROXY SETTINGS	no

CLI Command Equivalent

```
tanzu management-cluster create mgt-cluster --file /home/pda/.config/tanzu/tkg/clusterconfigs/t6zajzniqf.yaml -v 6
```

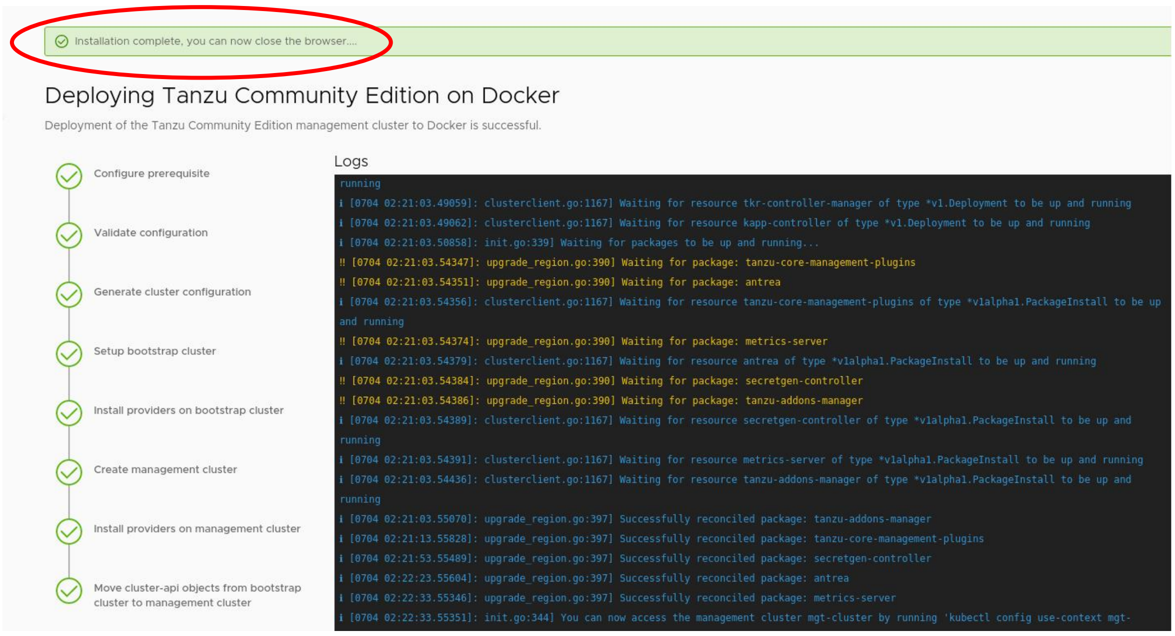
Take Note: While TCE UI sets up the management cluster, it will <u>generate</u> the YAML configuration file related to the cluster. Above, circled in green, is the location in which you can find the management cluster configuration file. In the configuration file, you can see all the parameters that make up the management cluster. In fact, if you so choose to create another management cluster, you can use the current configuration file as reference, and tweak the parameters according to your needs, before creating another TCE management cluster with the command "tanzu mc create --file path/to/cluster-config-file.yaml".

4. Afterwards, you will be directed to a page where you can see the process of deploying a management cluster before it finishes (all checkpoints are checked). This process might take a while:


Deploying Tanzu Community Edition on Docker
Deployment of the Tanzu Community Edition management cluster to Docker is in progress.
Configure prerequisite
Validate configuration
Generate cluster configuration
Setup bootstrap cluster
Install providers on bootstrap cluster
Create management cluster
Install providers on management cluster
Move cluster-api objects from bootstrap cluster to management cluster
Logs

```
i [0703 23:09:07.22886]: init.go:619] Waiting for provider cluster-api
i [0703 23:09:07.23099]: init.go:619] Waiting for provider infrastructure-docker
i [0703 23:09:07.23830]: init.go:619] Waiting for provider bootstrap-kubeadm
i [0703 23:09:07.23832]: init.go:619] Waiting for provider control-plane-kubeadm
i [0703 23:09:07.76248]: clusterclient.go:1167] Waiting for resource capd-controller-manager of type *v1.Deployment to be up and running
i [0703 23:09:07.79164]: clusterclient.go:1167] Waiting for resource capi-kubeadm-control-plane-controller-manager of type *v1.Deployment
to be up and running
i [0703 23:09:08.03722]: clusterclient.go:1167] Waiting for resource capi-kubeadm-bootstrap-controller-manager of type *v1.Deployment to be
up and running
i [0703 23:09:08.21025]: clusterclient.go:1167] Waiting for resource capi-controller-manager of type *v1.Deployment to be up and running
i [0703 23:09:13.22505]: init.go:627] Passed waiting on provider cluster-api after 5.996211157s
i [0703 23:09:18.09203]: init.go:627] Passed waiting on provider bootstrap-kubeadm after 10.853730838s
i [0703 23:09:22.90505]: init.go:627] Passed waiting on provider control-plane-kubeadm after 15.666720927s
i [0703 23:09:37.83861]: init.go:627] Passed waiting on provider infrastructure-docker after 30.607617095s
i [0703 23:09:37.83871]: init.go:638] Success waiting on all providers.
i [0703 23:09:37.83941]: init.go:212] Start creating management cluster...
i [0703 23:09:47.89540]: clusterclient.go:548] [cluster control plane is still being initialized: WaitingForControlPlane, cluster
infrastructure is still being provisioned: WaitingForControlPlane]
i [0703 23:10:02.92087]: clusterclient.go:548] cluster control plane is still being initialized: WaitingForControlPlane
i [0703 23:10:17.95259]: clusterclient.go:548] cluster control plane is still being initialized: ScalingUp
i [0703 23:11:17.93036]: clusterclient.go:1319] Getting secret for cluster
i [0703 23:11:17.93076]: clusterclient.go:1167] Waiting for resource mgt-cluster-kubeconfig of type *v1.Secret to be up and running
i [0703 23:11:17.94744]: init.go:244] Saving management cluster kubeconfig into /home/pda/.kube/config
i [0703 23:11:17.99094]: init.go:267] Installing providers on management cluster...
```

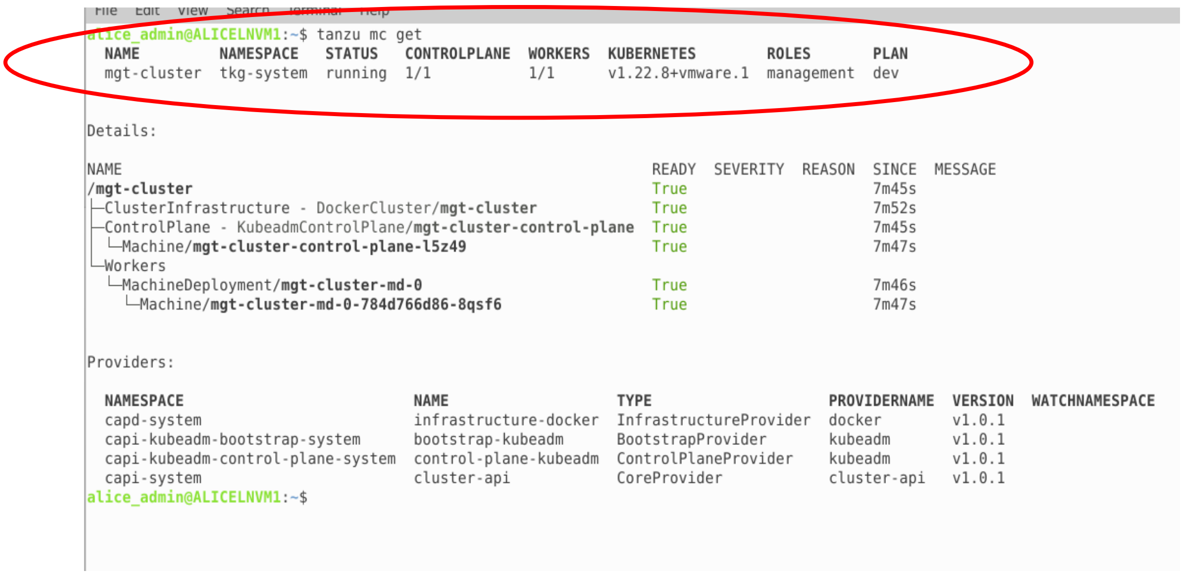
5. Once the management cluster has been successfully provisioned (you should get the same output as below), capture its kubeconfig by running the command "tanzu management-cluster kubeconfig get <MGMT-CLUSTER-NAME> --admin", where <MGMT-CLUSTER-NAME> should be replaced by your management cluster name that you have set in step 3. Take note of the command for accessing the cluster in the output message after you have ran the above command, as you will use this for setting the context in the next step.


Installation complete, you can now close the browser....
Deploying Tanzu Community Edition on Docker
Deployment of the Tanzu Community Edition management cluster to Docker is successful.
Configure prerequisite
Validate configuration
Generate cluster configuration
Setup bootstrap cluster
Install providers on bootstrap cluster
Create management cluster
Install providers on management cluster
Move cluster-api objects from bootstrap cluster to management cluster
Logs

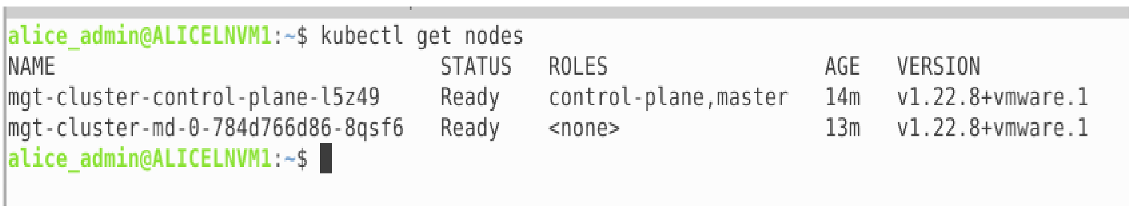
```
running
i [0704 02:21:03.49059]: clusterclient.go:1167] Waiting for resource tkr-controller-manager of type *v1.Deployment to be up and running
i [0704 02:21:03.49062]: clusterclient.go:1167] Waiting for resource kapp-controller of type *v1.Deployment to be up and running
i [0704 02:21:03.50858]: init.go:339] Waiting for packages to be up and running...
‼ [0704 02:21:03.54347]: upgrade_region.go:390] Waiting for package: tanzu-core-management-plugins
‼ [0704 02:21:03.54351]: upgrade_region.go:390] Waiting for package: antrea
i [0704 02:21:03.54356]: clusterclient.go:1167] Waiting for resource tanzu-core-management-plugins of type *v1alpha1.PackageInstall to be up
and running
‼ [0704 02:21:03.54374]: upgrade_region.go:390] Waiting for package: metrics-server
i [0704 02:21:03.54379]: clusterclient.go:1167] Waiting for resource antrea of type *v1alpha1.PackageInstall to be up and running
‼ [0704 02:21:03.54384]: upgrade_region.go:390] Waiting for package: secretgen-controller
‼ [0704 02:21:03.54386]: upgrade_region.go:390] Waiting for package: tanzu-addons-manager
i [0704 02:21:03.54389]: clusterclient.go:1167] Waiting for resource secretgen-controller of type *v1alpha1.PackageInstall to be up and
running
i [0704 02:21:03.54391]: clusterclient.go:1167] Waiting for resource metrics-server of type *v1alpha1.PackageInstall to be up and running
i [0704 02:21:03.54436]: clusterclient.go:1167] Waiting for resource tanzu-addons-manager of type *v1alpha1.PackageInstall to be up and
running
i [0704 02:21:03.55070]: upgrade_region.go:397] Successfully reconciled package: tanzu-addons-manager
i [0704 02:21:13.55828]: upgrade_region.go:397] Successfully reconciled package: tanzu-core-management-plugins
i [0704 02:21:53.55489]: upgrade_region.go:397] Successfully reconciled package: secretgen-controller
i [0704 02:22:23.55604]: upgrade_region.go:397] Successfully reconciled package: antrea
i [0704 02:22:33.55346]: upgrade_region.go:397] Successfully reconciled package: metrics-server
i [0704 02:22:33.55351]: init.go:344] You can now access the management cluster mgt-cluster by running 'kubectl config use-context mgt-
```

6. Set your kubectl context to the management cluster with the command "kubectl config use-context <MGMT-CLUSTER-NAME>-admin@<MGMT-CLUSTER-NAME>". (This command should have been shown, with your management cluster name already configured in it, in your terminal right after you have run the command in step 5. All you need to do is simply copy-paste the command)

7. If you have reached and completed step 6, you have successfully created your management cluster. You can now run commands "tanzu mc get", to check if your management cluster is up and running, and "kubectl get nodes", to see if your control plane and worker node are running.



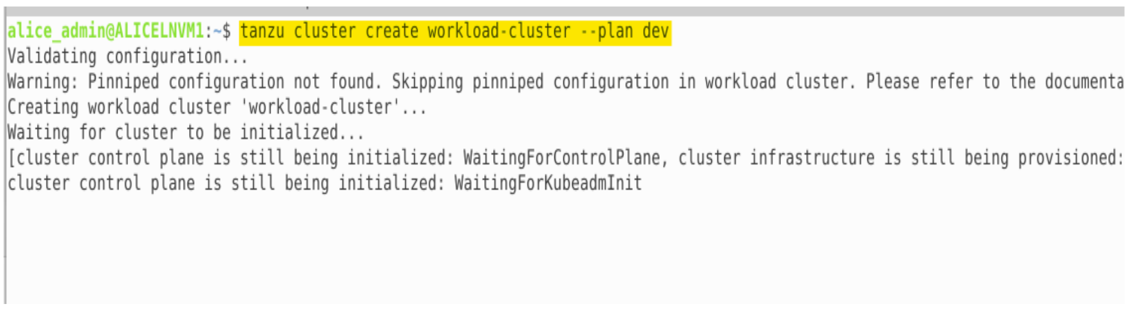
```
alice_admin@ALICELNVM1:~$ tanzu mc get
  NAME         NAMESPACE   STATUS   CONTROLPLANE  WORKERS  KUBERNETES        ROLES       PLAN
  mgt-cluster  tkg-system  running  1/1           1/1      v1.22.8+vmware.1  management  dev


Details:

NAME                                                                READY  SEVERITY  REASON  SINCE  MESSAGE
/mgt-cluster                                                        True                     7m45s
├─ClusterInfrastructure - DockerCluster/mgt-cluster                 True                     7m52s
├─ControlPlane - KubeadmControlPlane/mgt-cluster-control-plane      True                     7m45s
│ └─Machine/mgt-cluster-control-plane-l5z49                         True                     7m47s
└─Workers
  └─MachineDeployment/mgt-cluster-md-0                              True                     7m46s
    └─Machine/mgt-cluster-md-0-784d766d86-8qsf6                     True                     7m47s


Providers:

  NAMESPACE                          NAME                    TYPE                    PROVIDERNAME  VERSION  WATCHNAMESPACE
  capd-system                        infrastructure-docker   InfrastructureProvider  docker        v1.0.1
  capi-kubeadm-bootstrap-system      bootstrap-kubeadm       BootstrapProvider       kubeadm       v1.0.1
  capi-kubeadm-control-plane-system  control-plane-kubeadm   ControlPlaneProvider    kubeadm       v1.0.1
  capi-system                        cluster-api             CoreProvider            cluster-api   v1.0.1
alice_admin@ALICELNVM1:~$
```



```
alice_admin@ALICELNVM1:~$ kubectl get nodes
NAME                                STATUS   ROLES                  AGE   VERSION
mgt-cluster-control-plane-l5z49     Ready    control-plane,master   14m   v1.22.8+vmware.1
mgt-cluster-md-0-784d766d86-8qsf6   Ready    <none>                 13m   v1.22.8+vmware.1
alice_admin@ALICELNVM1:~$
```

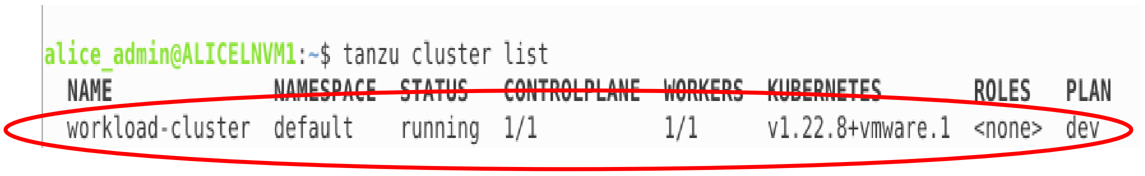
8. Now it is time we create our **workload cluster** so that we can make use of our newly created management cluster to manage it. Run the command "tanzu cluster create

<WORKLOAD-CLUSTER-NAME> --plan dev", where <WORKLOAD-CLUSTER-NAME> will be any name you intend to have for your workload cluster. Upon running this command, our management cluster will start provisioning a workload cluster. This process might take a while:



```
alice_admin@ALICELNVM1:~$ tanzu cluster create workload-cluster --plan dev
Validating configuration...
Warning: Pinniped configuration not found. Skipping pinniped configuration in workload cluster. Please refer to the documenta
Creating workload cluster 'workload-cluster'...
Waiting for cluster to be initialized...
[cluster control plane is still being initialized: WaitingForControlPlane, cluster infrastructure is still being provisioned:
cluster control plane is still being initialized: WaitingForKubeadmInit
```

9. After the workload cluster has been created, check that it is up and running by executing the command "tanzu cluster list":

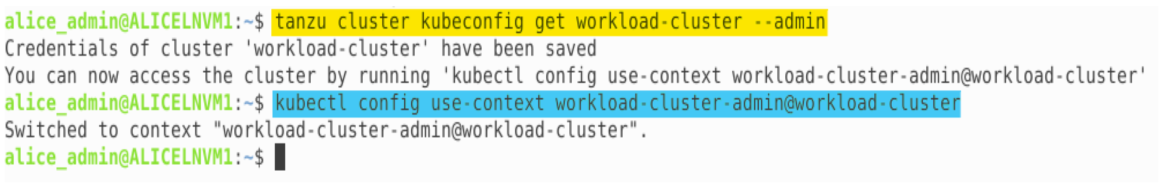


```
alice_admin@ALICELNVM1:~$ tanzu cluster list
  NAME              NAMESPACE  STATUS   CONTROLPLANE  WORKERS  KUBERNETES        ROLES   PLAN
  workload-cluster  default    running  1/1           1/1      v1.22.8+vmware.1  <none>  dev
```

10. Just like what we did for our management cluster setup, capture the workload cluster's kubeconfig by running the command "tanzu cluster kubeconfig get <WORKLOAD-CLUSTER-NAME> --admin", where <WORKLOAD-CLUSTER-NAME> is the workload cluster name that you have set in step 8. Take note of the command for accessing the cluster in the output message after you have ran the above command, as you will use this for setting the context in the next step

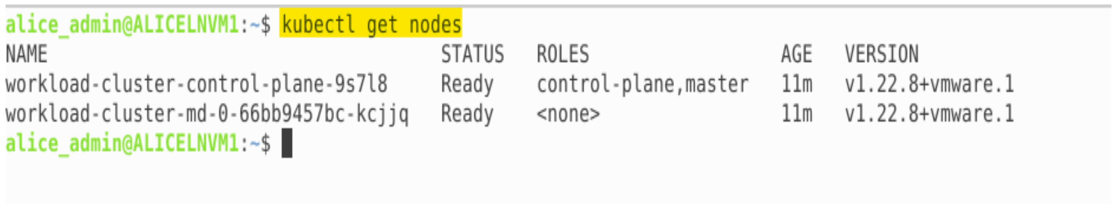
11. Set the kubectl context accordingly by running the command "kubectl config use-context <WORKLOAD-CLUSTER-NAME>-admin@<WORKLOAD-CLUSTER-NAME>". (This command should have been shown, with your workload cluster name

already configured in it, in your terminal right after you have run the command in step 10. All you need to do is simply copy-paste the command):


```
alice_admin@ALICELNVM1:~$ tanzu cluster kubeconfig get workload-cluster --admin
Credentials of cluster 'workload-cluster' have been saved
You can now access the cluster by running 'kubectl config use-context workload-cluster-admin@workload-cluster'
alice_admin@ALICELNVM1:~$ kubectl config use-context workload-cluster-admin@workload-cluster
Switched to context "workload-cluster-admin@workload-cluster".
alice_admin@ALICELNVM1:~$ 
```

Take Note: This step is essential when deploying managed clusters. As the kubectl command-line tool uses kubeconfig files to find the information it needs to **choose** a cluster and communicate with the API server of a cluster, with the above command, all kubectl commands will now be accessing the workload cluster, instead of the management cluster.

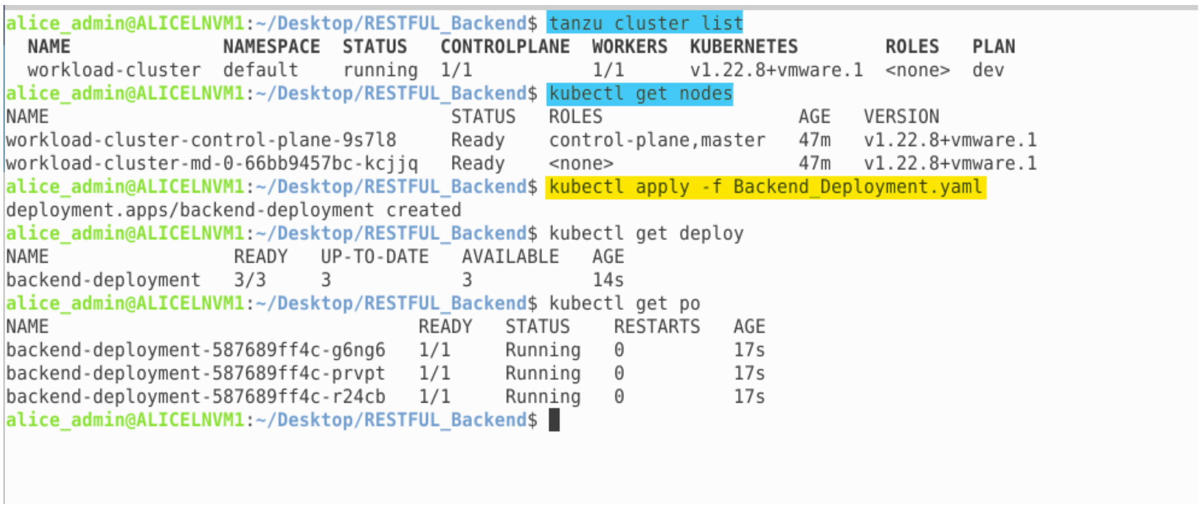
12. If you have reached and completed step 11, you have successfully created a workload cluster. You can now run commands like "kubectl get nodes" to check if the control plane and worker node of the workload cluster is up and running:


```
alice_admin@ALICELNVM1:~$ kubectl get nodes
NAME                                  STATUS   ROLES                  AGE   VERSION
workload-cluster-control-plane-9s7l8  Ready    control-plane,master   11m   v1.22.8+vmware.1
workload-cluster-md-0-66bb9457bc-kcjjq Ready   <none>                 11m   v1.22.8+vmware.1
alice_admin@ALICELNVM1:~$ 
```

Deploying Workloads

For unmanaged clusters, as long as they are provisioned successfully, workloads can be deployed. As for managed clusters, make sure both the management cluster and its workload cluster(s) are up and running before deploying workloads, if not, refer back to the previous section on the installation of it.

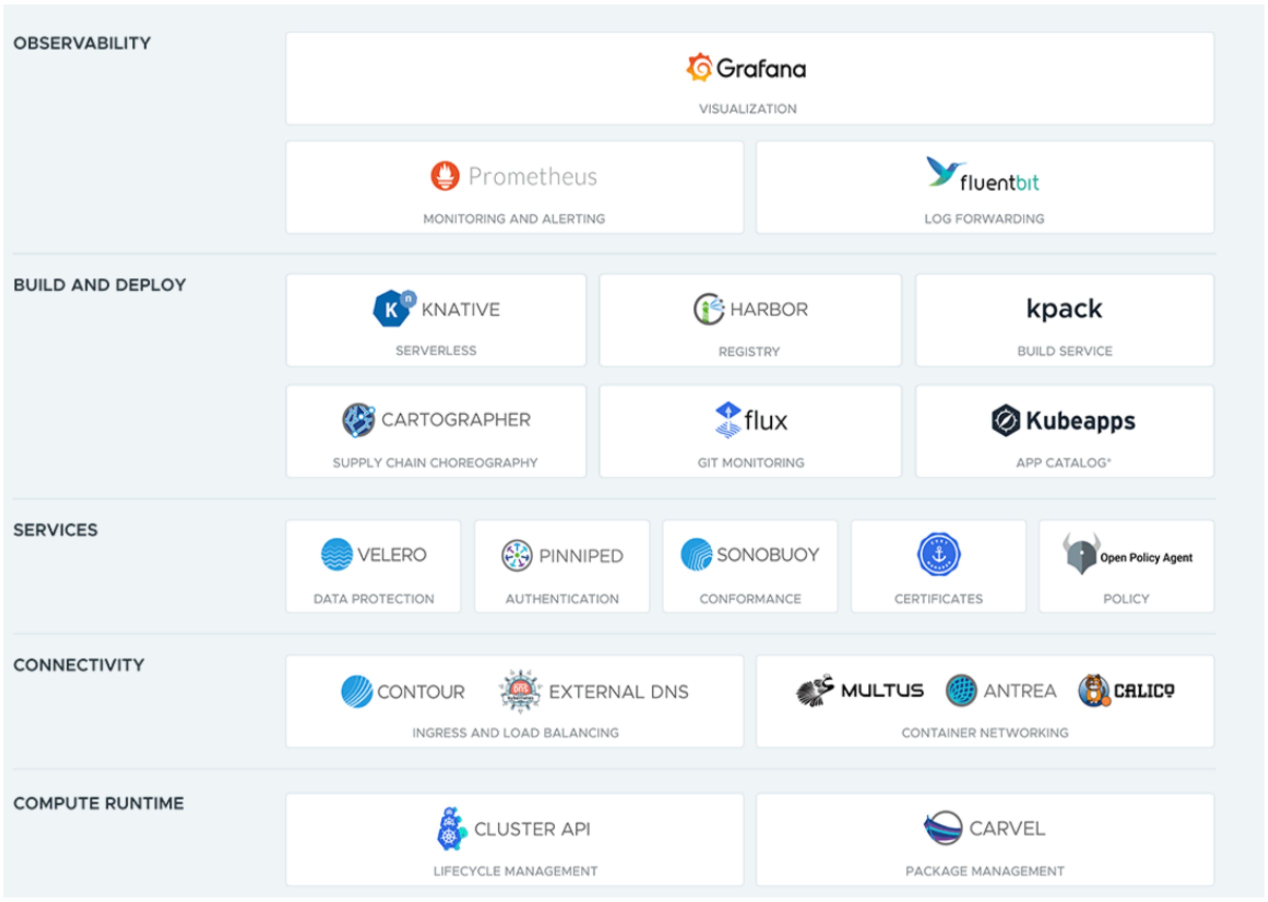
As TCE is still another native Kubernetes platform, we can provision workloads, for instance, a deployment, to the clusters using "kubectl apply -f <yaml configuration file of deployment>".


```
alice_admin@ALICELNVM1:~/Desktop/RESTFUL_Backend$ tanzu cluster list
  NAME              NAMESPACE  STATUS   CONTROLPLANE  WORKERS  KUBERNETES        ROLES   PLAN
  workload-cluster  default    running  1/1           1/1      v1.22.8+vmware.1  <none>  dev
alice_admin@ALICELNVM1:~/Desktop/RESTFUL_Backend$ kubectl get nodes
NAME                                    STATUS   ROLES                  AGE   VERSION
workload-cluster-control-plane-9s7l8    Ready    control-plane,master   47m   v1.22.8+vmware.1
workload-cluster-md-0-66bb9457bc-kcjjq  Ready    <none>                 47m   v1.22.8+vmware.1
alice_admin@ALICELNVM1:~/Desktop/RESTFUL_Backend$ kubectl apply -f Backend_Deployment.yaml
deployment.apps/backend-deployment created
alice_admin@ALICELNVM1:~/Desktop/RESTFUL_Backend$ kubectl get deploy
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
backend-deployment   3/3     3            3           14s
alice_admin@ALICELNVM1:~/Desktop/RESTFUL_Backend$ kubectl get po
NAME                                  READY   STATUS    RESTARTS   AGE
backend-deployment-587689ff4c-g6ng6   1/1     Running   0          17s
backend-deployment-587689ff4c-prvpt   1/1     Running   0          17s
backend-deployment-587689ff4c-r24cb   1/1     Running   0          17s
alice_admin@ALICELNVM1:~/Desktop/RESTFUL_Backend$ 
```

In the above example, we have deployed a management cluster and a workload cluster, and would like to provision a workload to the workload cluster. First, we check that the workload cluster is running with the command "tanzu cluster list". Afterwards, we validate that both the workload cluster's control plane and worker node are running with the command "kubectl get nodes". Finally, when all is ready, we deploy our workload by running the command "kubectl apply -f …". In this example, the workload YAML file is "Backend_Deployment.yaml".

We can do the same for Kubernetes services, secrets, configmaps and more.

TCE Package Management



One of the great benefits to using TCE is that it provides you a fully customizable Kubernetes platform as it includes the same open-source software used in Tanzu commercial offerings for absolutely no cost. Therefore, once you have deployed your TCE managed clusters, you can start layering further abstractions, or start adding monitoring and logging capabilities to them. All of this is done via Tanzu Community Edition's package management capabilities.

In your managed cluster, TCE has already installed for you a controller that understands how to install a package from TCE package repository and a Custom Resource Definition (CRD) that stores the state of packages that are available for installation or are already installed. Installation and managing of packages/open-source software can be done with the help of

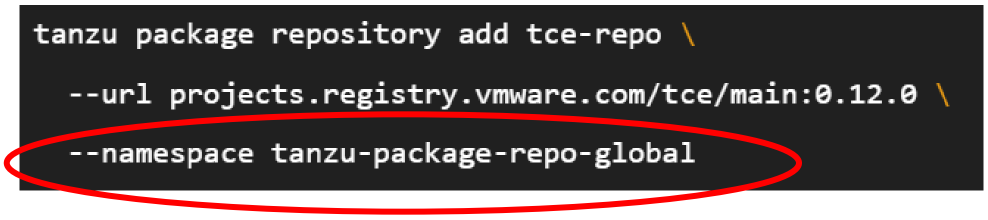
Tanzu's CLI "tanzu package" command. You can take a look at all the packages, and their respective websites, here: https://tanzucommunityedition.io/packages/

Take Note: TCE package management only works with managed clusters, and NOT unmanaged clusters.

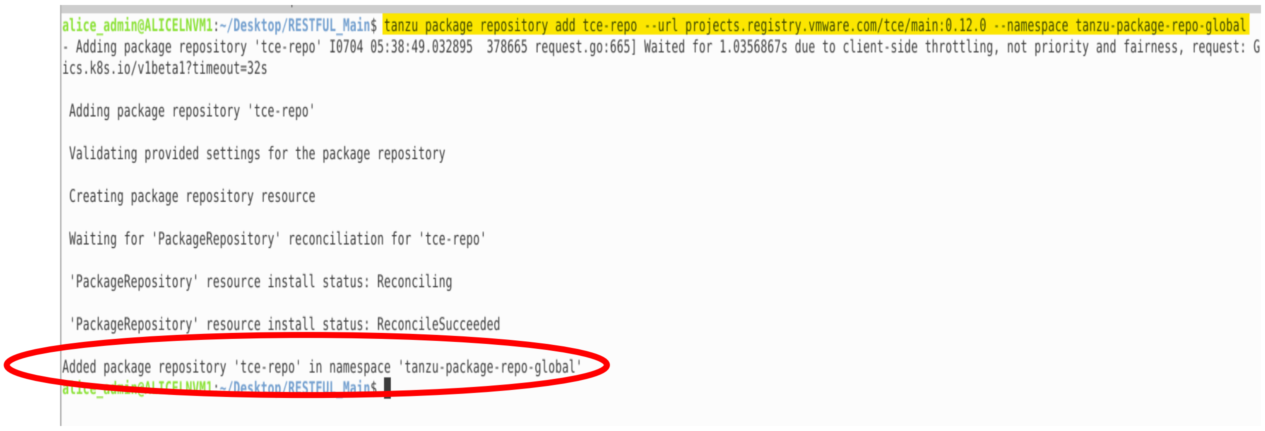
Installing a Package

Now, let us take a look at how we can install any package from TCE package repository.

1. Before installing any package, we have to first add TCE package repository to our managed cluster by the running the "tanzu package" command as shown below:



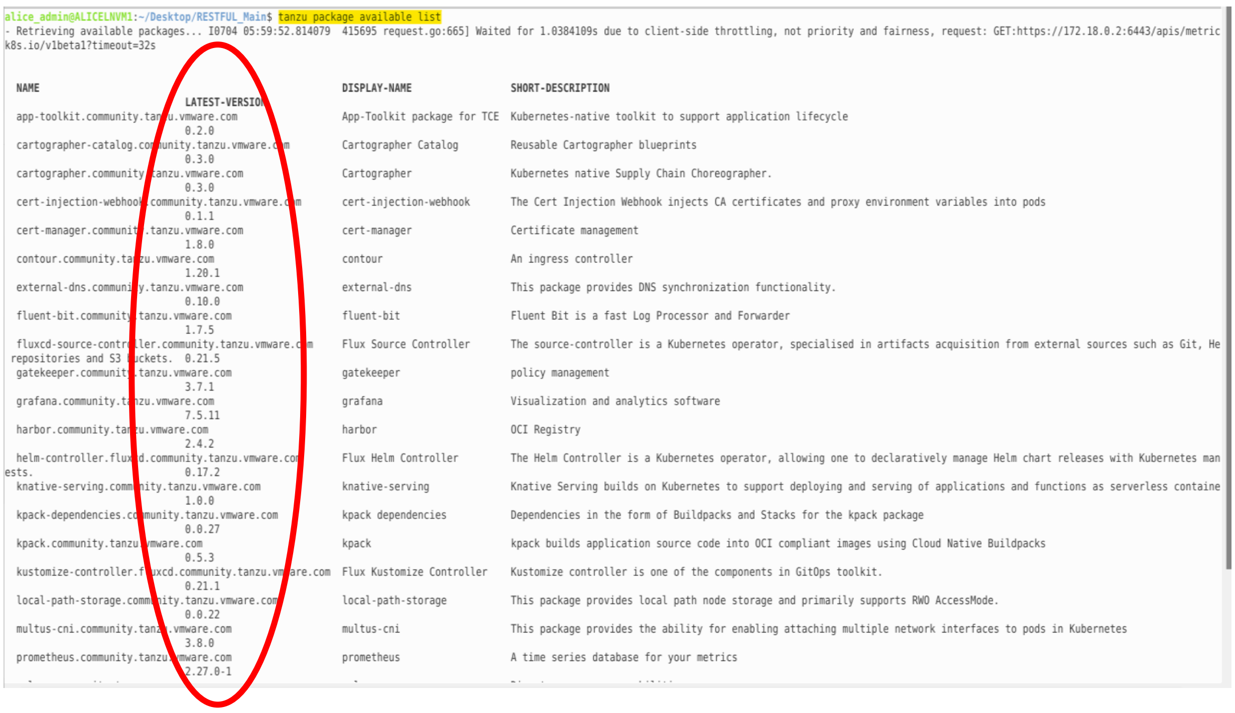
```
tanzu package repository add tce-repo \
  --url projects.registry.vmware.com/tce/main:0.12.0 \
  --namespace tanzu-package-repo-global
```



```
alice_admin@ALICELNVM1:~/Desktop/RESTFUL_Main$ tanzu package repository add tce-repo --url projects.registry.vmware.com/tce/main:0.12.0 --namespace tanzu-package-repo-global
- Adding package repository 'tce-repo' I0704 05:38:49.032895  378665 request.go:665] Waited for 1.0356867s due to client-side throttling, not priority and fairness, request: G
ics.k8s.io/v1beta1?timeout=32s

 Adding package repository 'tce-repo'

 Validating provided settings for the package repository

 Creating package repository resource

 Waiting for 'PackageRepository' reconciliation for 'tce-repo'

 'PackageRepository' resource install status: Reconciling

 'PackageRepository' resource install status: ReconcileSucceeded

Added package repository 'tce-repo' in namespace 'tanzu-package-repo-global'
alice_admin@ALICELNVM1:~/Desktop/RESTFUL_Main$ 
```

Take Note: One of the parameters of the command is "namespace". If this parameter is not specified, then you will only be downloading Tanzu package repository in your cluster's "default" namespace. However, if "tanzu-package-repo-global" is specified in the parameter, like in the above example, then the Tanzu package repository will be available to ALL namespaces residing in your workload cluster/cluster.

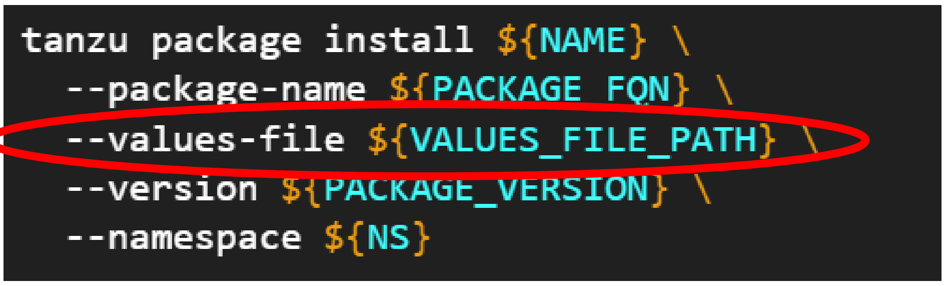
2. Now that the Tanzu package repository is added to your cluster, you can view all the available packages found within the repository just by running the command "tanzu package available list", and you should get the result similar to what is shown below:



```
alice_admin@ALICELNVM1:~/Desktop/RESTFUL_Main$ tanzu package available list
- Retrieving available packages... I0704 05:59:52.814079  415695 request.go:665] Waited for 1.0384109s due to client-side throttling, not priority and fairness, request: GET:https://172.18.0.2:6443/apis/metric
k8s.io/v1beta1?timeout=32s


  NAME                                                DISPLAY-NAME                SHORT-DESCRIPTION
                            LATEST-VERSION
  app-toolkit.community.tanzu.vmware.com              App-Toolkit package for TCE Kubernetes-native toolkit to support application lifecycle
                            0.2.0
  cartographer-catalog.community.tanzu.vmware.com     Cartographer Catalog        Reusable Cartographer blueprints
                            0.3.0
  cartographer.community.tanzu.vmware.com             Cartographer                Kubernetes native Supply Chain Choreographer.
                            0.3.0
  cert-injection-webhook.community.tanzu.vmware.com   cert-injection-webhook      The Cert Injection Webhook injects CA certificates and proxy environment variables into pods
                            0.1.1
  cert-manager.community.tanzu.vmware.com             cert-manager                Certificate management
                            1.8.0
  contour.community.tanzu.vmware.com                  contour                     An ingress controller
                            1.20.1
  external-dns.community.tanzu.vmware.com             external-dns                This package provides DNS synchronization functionality.
                            0.10.0
  fluent-bit.community.tanzu.vmware.com               fluent-bit                  Fluent Bit is a fast Log Processor and Forwarder
                            1.7.5
  fluxcd-source-controller.community.tanzu.vmware.com Flux Source Controller      The source-controller is a Kubernetes operator, specialised in artifacts acquisition from external sources such as Git, He
 repositories and S3 buckets.  0.21.5
  gatekeeper.community.tanzu.vmware.com               gatekeeper                  policy management
                            3.7.1
  grafana.community.tanzu.vmware.com                  grafana                     Visualization and analytics software
                            7.5.11
  harbor.community.tanzu.vmware.com                   harbor                      OCI Registry
                            2.4.2
  helm-controller.fluxcd.community.tanzu.vmware.com   Flux Helm Controller        The Helm Controller is a Kubernetes operator, allowing one to declaratively manage Helm chart releases with Kubernetes man
ests.                       0.17.2
  knative-serving.community.tanzu.vmware.com          knative-serving             Knative Serving builds on Kubernetes to support deploying and serving of applications and functions as serverless containe
                            1.0.0
  kpack-dependencies.community.tanzu.vmware.com       kpack dependencies          Dependencies in the form of Buildpacks and Stacks for the kpack package
                            0.0.27
  kpack.community.tanzu.vmware.com                    kpack                       kpack builds application source code into OCI compliant images using Cloud Native Buildpacks
                            0.5.3
  kustomize-controller.fluxcd.community.tanzu.vmware.com Flux Kustomize Controller Kustomize controller is one of the components in GitOps toolkit.
                            0.21.1
  local-path-storage.community.tanzu.vmware.com       local-path-storage          This package provides local path node storage and primarily supports RWO AccessMode.
                            0.0.22
  multus-cni.community.tanzu.vmware.com               multus-cni                  This package provides the ability for enabling attaching multiple network interfaces to pods in Kubernetes
                            3.8.0
  prometheus.community.tanzu.vmware.com               prometheus                  A time series database for your metrics
                            2.27.0-1
```

Under the "NAME" column are the names of different open-source packages available to you to implement to your cluster. However, it is important to take note of the <u>version</u> of each package as you will need this information when you install any of the packages later.

3. Generally, to install a package, the command to run is as shown below:

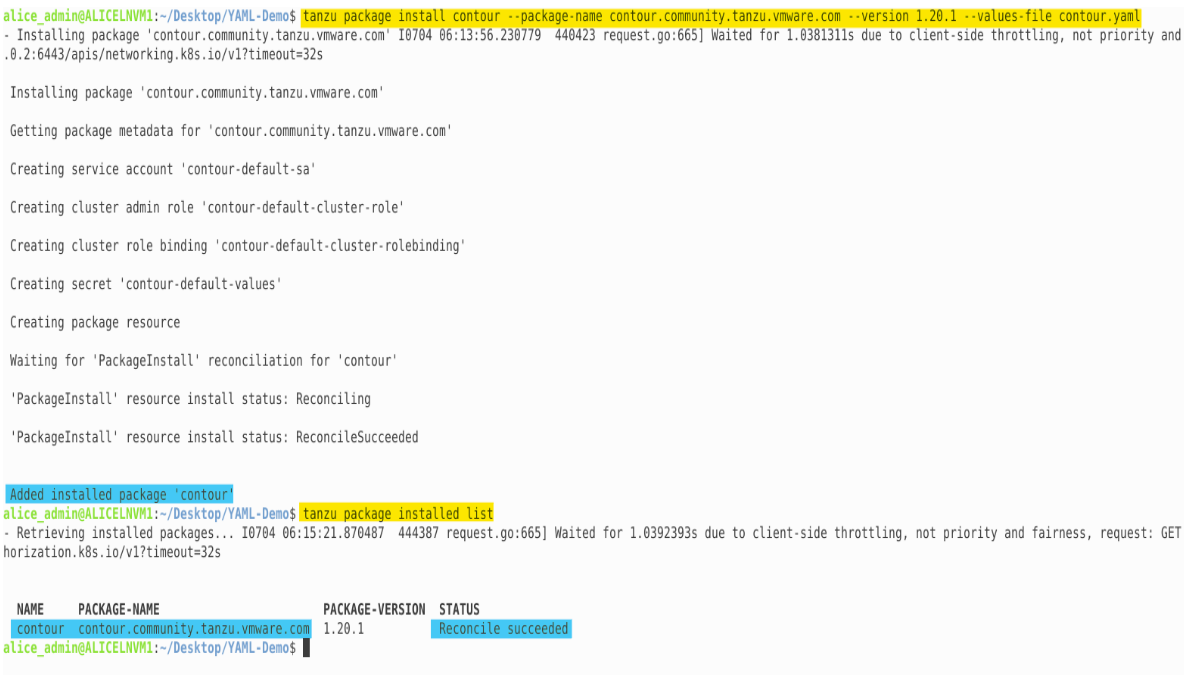


```
tanzu package install ${NAME} \
  --package-name ${PACKAGE_FQN} \
  --values-file ${VALUES_FILE_PATH} \
  --version ${PACKAGE_VERSION} \
  --namespace ${NS}
```

Where ${NAME} refers to the **DISPLAY-NAME** in step 4, ${PACKAGE_FQN} refers to NAME in step 4, ${PACKAGE_VERSION} refers to **LATEST_VERSION** in step 4, ${VALUES_FILE_PATH} refers to the **configuration file** that you are using in conjunction with the installation, and ${NS} is the **namespace** in which you want to install the package.

Take Note: The “values-file” parameter is optional and so you do not need to have it when installing a package unless you intend to have a specific configuration to your package.

For the purpose of our demonstration, we will install TCE Contour Ingress package:



```
alice_admin@ALICELNVM1:~/Desktop/YAML-Demo$ tanzu package install contour --package-name contour.community.tanzu.vmware.com --version 1.20.1 --values-file contour.yaml
- Installing package 'contour.community.tanzu.vmware.com' I0704 06:13:56.230779  440423 request.go:665] Waited for 1.0381311s due to client-side throttling, not priority and
.0.2:6443/apis/networking.k8s.io/v1?timeout=32s

 Installing package 'contour.community.tanzu.vmware.com'

 Getting package metadata for 'contour.community.tanzu.vmware.com'

 Creating service account 'contour-default-sa'

 Creating cluster admin role 'contour-default-cluster-role'

 Creating cluster role binding 'contour-default-cluster-rolebinding'

 Creating secret 'contour-default-values'

 Creating package resource

 Waiting for 'PackageInstall' reconciliation for 'contour'

 'PackageInstall' resource install status: Reconciling

 'PackageInstall' resource install status: ReconcileSucceeded


 Added installed package 'contour'
alice_admin@ALICELNVM1:~/Desktop/YAML-Demo$ tanzu package installed list
- Retrieving installed packages... I0704 06:15:21.870487  444387 request.go:665] Waited for 1.0392393s due to client-side throttling, not priority and fairness, request: GET
horization.k8s.io/v1?timeout=32s


  NAME     PACKAGE-NAME                        PACKAGE-VERSION  STATUS
  contour  contour.community.tanzu.vmware.com  1.20.1           Reconcile succeeded
alice_admin@ALICELNVM1:~/Desktop/YAML-Demo$
```

In the above example, we are installing the Contour package using the command “tanzu package install contour…”. After it has been installed, we have to check if the package is in fact installed successfully. We run the command “tanzu package installed list”, which will show us all the installed packages in our cluster, and more importantly, the STATUS.

If a package has been installed successfully, under STATUS, "Reconcile succeeded" should be seen. If not, you can delete the installed package, using the command shown below:

```
tanzu package installed delete ${NAME} --namespace ${NS}
```

${NAME} refers to the name of the installed package. In our example, it will be "contour". After deleting the package, you can reinstall the package using the general package installation command as shown above.

If all is well, then, now you have that installed package at your disposal, and you can implement it freely in your cluster to suit your needs.

Updating Packages

There are times when we have already installed a package, however, we would like to change certain parameters of it, for instance, its version, the namespace the package resides in, or even the package's configuration file. You can make these changes, or update the installed package, by running the command below:

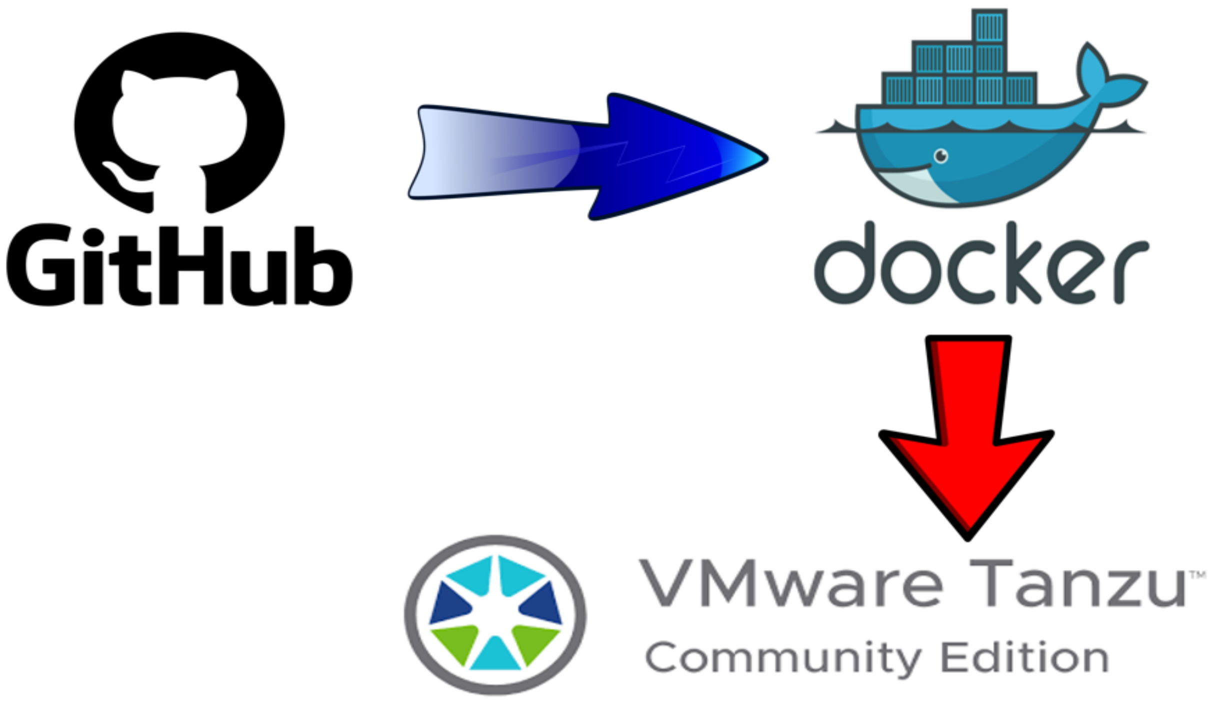
```
tanzu package installed update ${NAME} \
  --package-name ${PACKAGE_FQN} \
  --values-file ${VALUES_FILE_PATH} \
  --version ${PACKAGE_VERSION} \
  --namespace ${NS}
```

In the command above, specify ONLY the fields of the installed package that you wish to update. Fields of the installed package that you have no intention of updating should not be specified in the command.

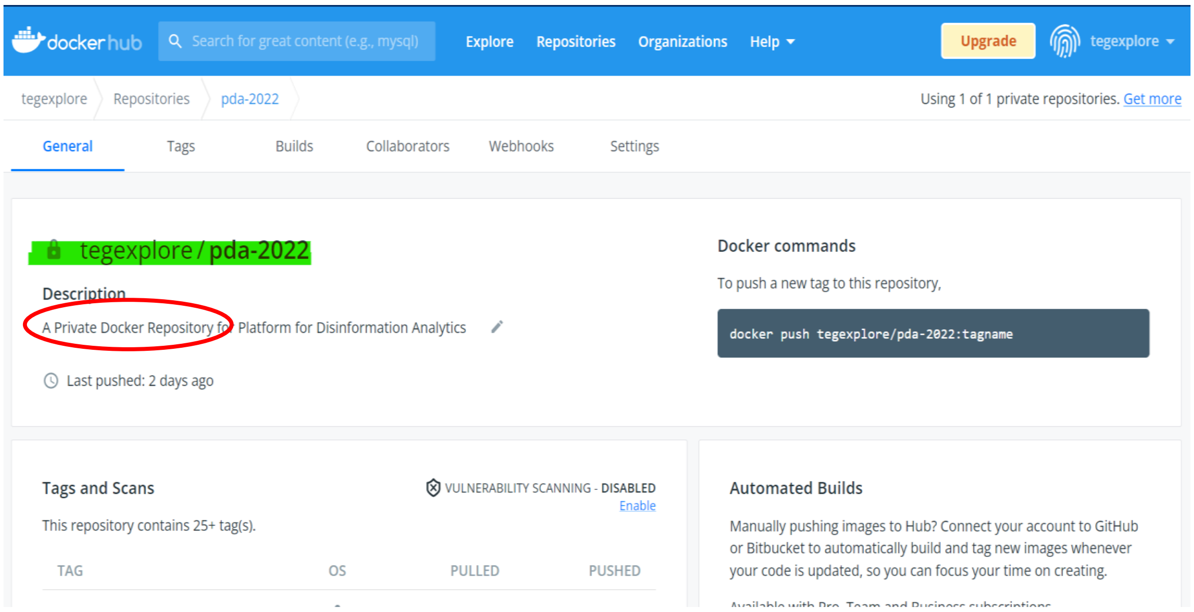
Miscellaneous

In this section, we will briefly talk about the overlooked processes of the deployment of applications, for instance, the creation of docker images, and the building and pushing of them, and also the general guidelines when doing up a YAML configuration file in order to deploy a Kubernetes resource. It is important to understand, or at the very least, have a rough idea of how each process works as it will greatly aid you while you attempt to deploy your applications onto Tanzu Community Edition, or Kubernetes.

<u>Docker Image Workflow</u>



The general workflow for creating, pushing, and pulling docker images for deployments on TCE, or Kubernetes, goes like this: we will first do up a Docker file, and upload it onto GitHub, and after which, we will take advantage of GitHub's CI/CD workflow, which will provide us with its amazing CI/CD automation workflow tool. This tool automates the process of building docker images from the docker file already uploaded on GitHub, and pushing the images to a private Docker Hub repository. Once the images are pushed to the private repository, TCE will then be able to pull the images before containerizing them for deployment. Below is a sample snapshot of a private Docker Hub repository.



Using 1 of 1 private repositories. Get more

tegexplore / pda-2022

Description

A Private Docker Repository for Platform for Disinformation Analytics

Last pushed: 2 days ago

Docker commands

To push a new tag to this repository,

```
docker push tegexplore/pda-2022:tagname
```

Tags and Scans

VULNERABILITY SCANNING - DISABLED

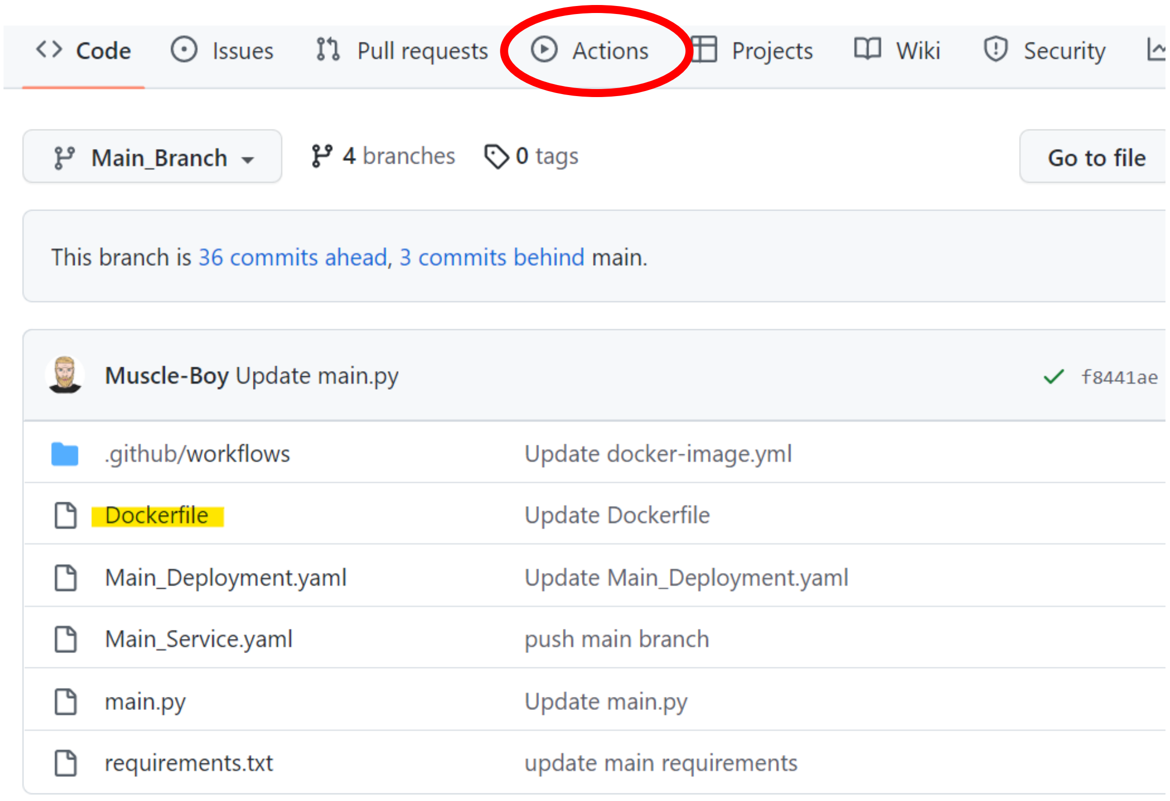
This repository contains 25+ tag(s).

TAG	OS	PULLED	PUSHED

Automated Builds

Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Now, before diving into the deployment aspect of TCE, let us look at how we can build a docker image using GitHub’s CI/CD utility.

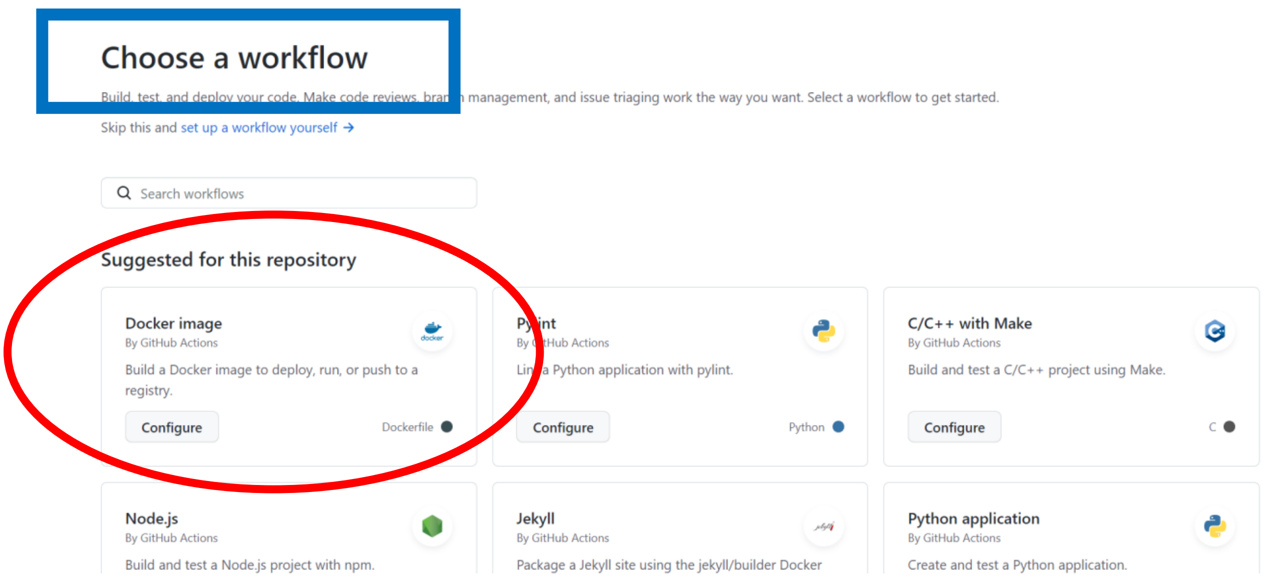


This branch is 36 commits ahead, 3 commits behind main.

Muscle-Boy Update main.py ✓ f8441ae

.github/workflows	Update docker-image.yml
Dockerfile	Update Dockerfile
Main_Deployment.yaml	Update Main_Deployment.yaml
Main_Service.yaml	push main branch
main.py	Update main.py
requirements.txt	update main requirements

Fundamentally, it is important we have our "Dockerfile" uploaded on GitHub as this file contains all the necessary configurations that our applications need when they are deployed into production. Without this file, the following steps will not work. Now, we will have to select the "Actions" option circled in red, and this will bring us the to the page where we can create our CI/CD workflow as shown below.



Choose a workflow

Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow to get started.

Skip this and set up a workflow yourself →

Search workflows

Suggested for this repository

Docker image
By GitHub Actions
Build a Docker image to deploy, run, or push to a registry.
Configure — Dockerfile

Pylint
By GitHub Actions
Lint a Python application with pylint.
Configure — Python

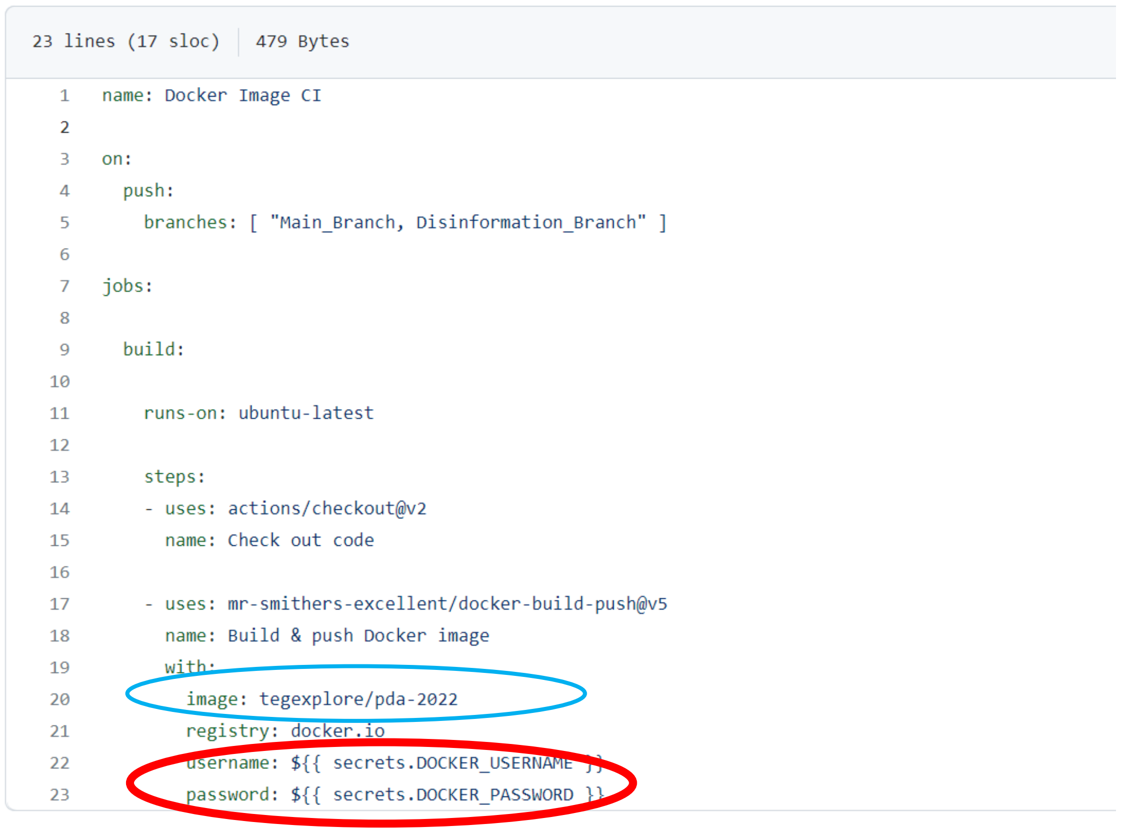
C/C++ with Make
By GitHub Actions
Build and test a C/C++ project using Make.
Configure — C

Node.js
By GitHub Actions
Build and test a Node.js project with npm.

Jekyll
By GitHub Actions
Package a Jekyll site using the jekyll/builder Docker

Python application
By GitHub Actions
Create and test a Python application.

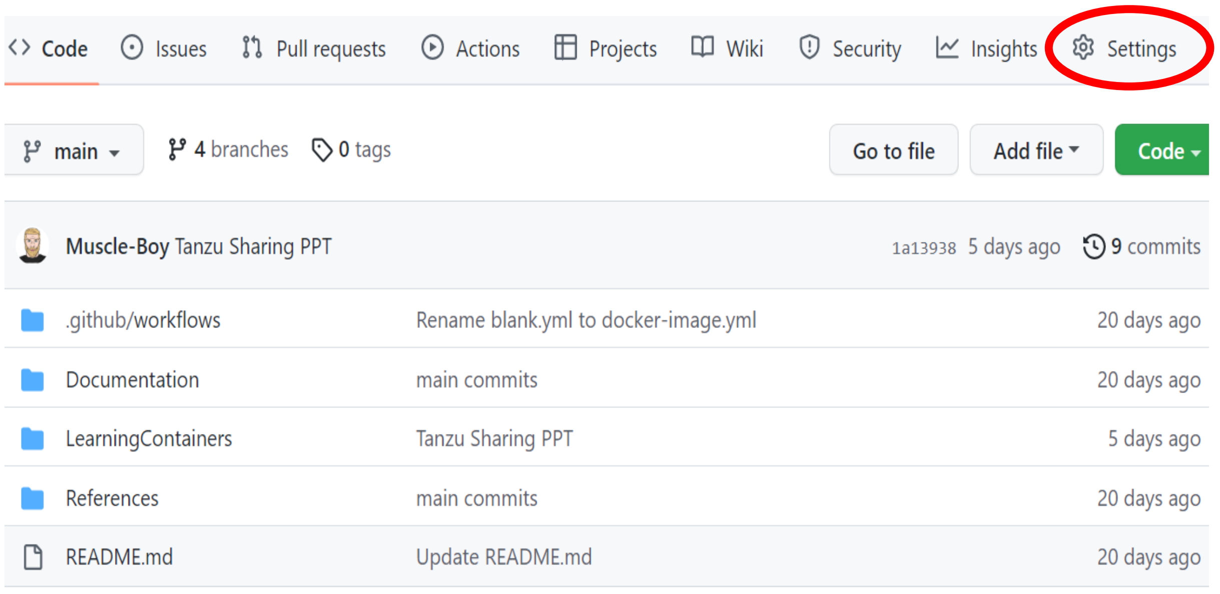
On this page, you can select from the many different “templates”, or YAML configuration files, already provided by GitHub. In our case, since we are looking to automate the building, and pushing, of Docker images, we will select the option circled in red.



23 lines (17 sloc) | 479 Bytes

```yaml
name: Docker Image CI

on:
  push:
    branches: [ "Main_Branch, Disinformation_Branch" ]

jobs:

  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
      name: Check out code

    - uses: mr-smithers-excellent/docker-build-push@v5
      name: Build & push Docker image
      with:
        image: tegexplore/pda-2022
        registry: docker.io
        username: ${{ secrets.DOCKER_USERNAME }}
        password: ${{ secrets.DOCKER_PASSWORD }}
```

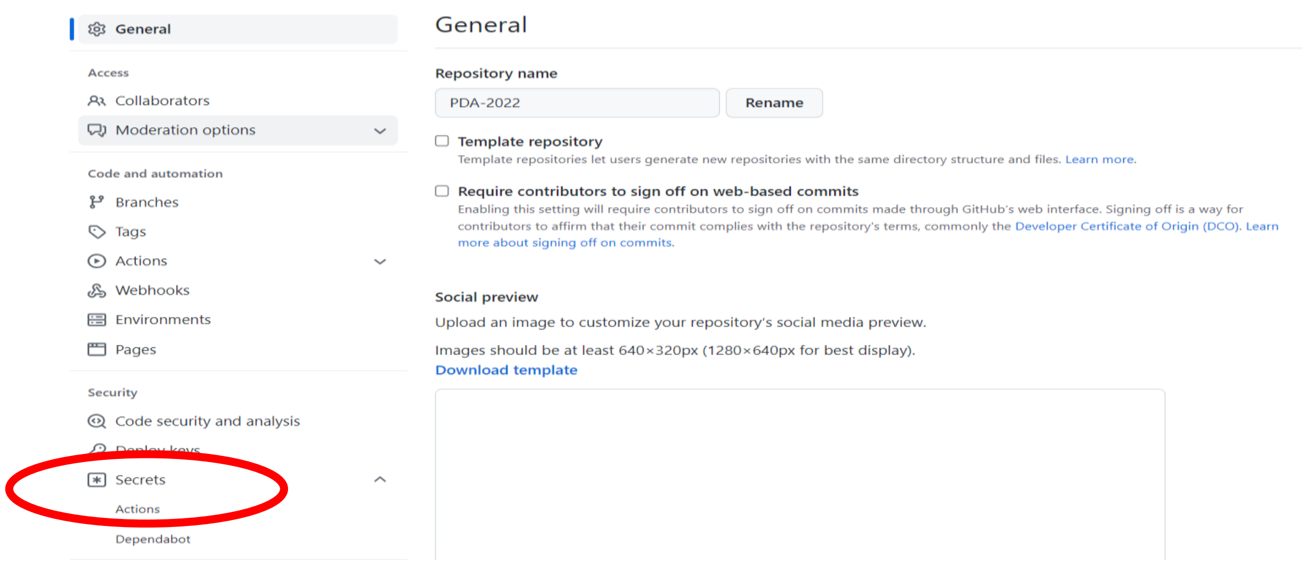
GitHub's CI/CD workflow comes in the form a YAML file. The above is a sample you can input to invoke the workflow. Pay special attention to the "image" parameter, as it will be specific to your private Docker Hub repository, and the "username" and "password" parameters, as they store your Docker Hub private repository's username and password. The values of the latter parameters, for security purposes, are recommended to only be referenced from GitHub "secret" files.



Muscle-Boy Tanzu Sharing PPT 1a13938 5 days ago 9 commits

.github/workflows	Rename blank.yml to docker-image.yml	20 days ago
Documentation	main commits	20 days ago
LearningContainers	Tanzu Sharing PPT	5 days ago
References	main commits	20 days ago
README.md	Update README.md	20 days ago

To create secret files, we have to select the "Settings" option, which will bring us to the page shown below:



General

Repository name

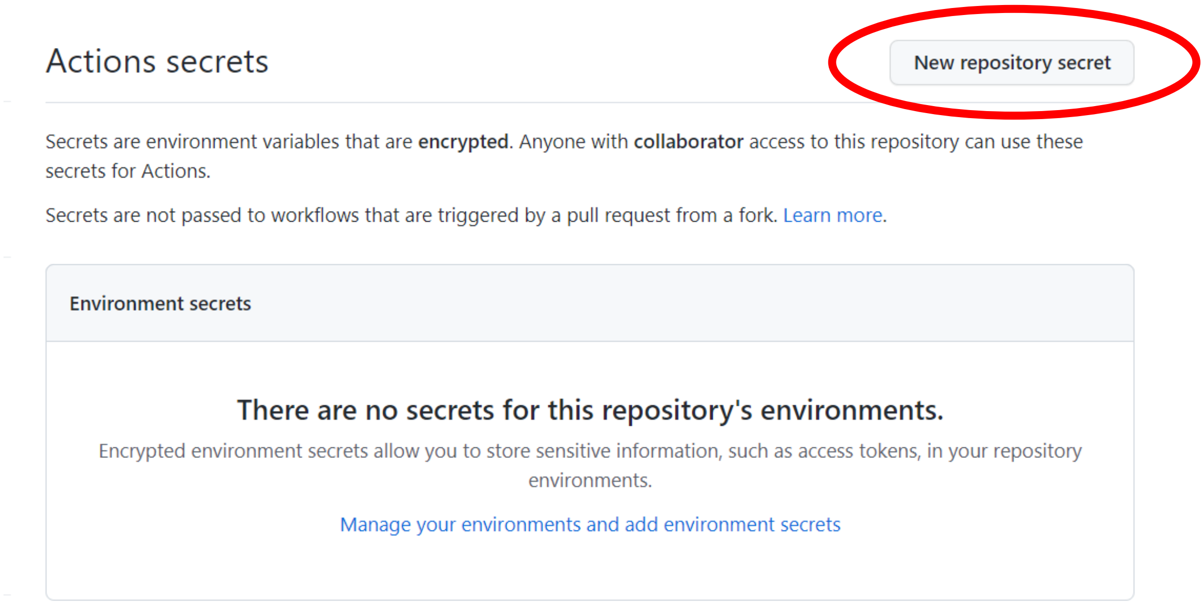
PDA-2022 Rename

Template repository
Template repositories let users generate new repositories with the same directory structure and files. Learn more.

Require contributors to sign off on web-based commits
Enabling this setting will require contributors to sign off on commits made through GitHub's web interface. Signing off is a way for contributors to affirm that their commit complies with the repository's terms, commonly the Developer Certificate of Origin (DCO). Learn more about signing off on commits.

Social preview
Upload an image to customize your repository's social media preview.
Images should be at least 640×320px (1280×640px for best display).
Download template

Next, we will have to look for the "Secrets" option, and select "Actions".



Secrets are environment variables that are **encrypted**. Anyone with **collaborator** access to this repository can use these secrets for Actions.

Secrets are not passed to workflows that are triggered by a pull request from a fork. Learn more.

Environment secrets

There are no secrets for this repository's environments.

Encrypted environment secrets allow you to store sensitive information, such as access tokens, in your repository environments.

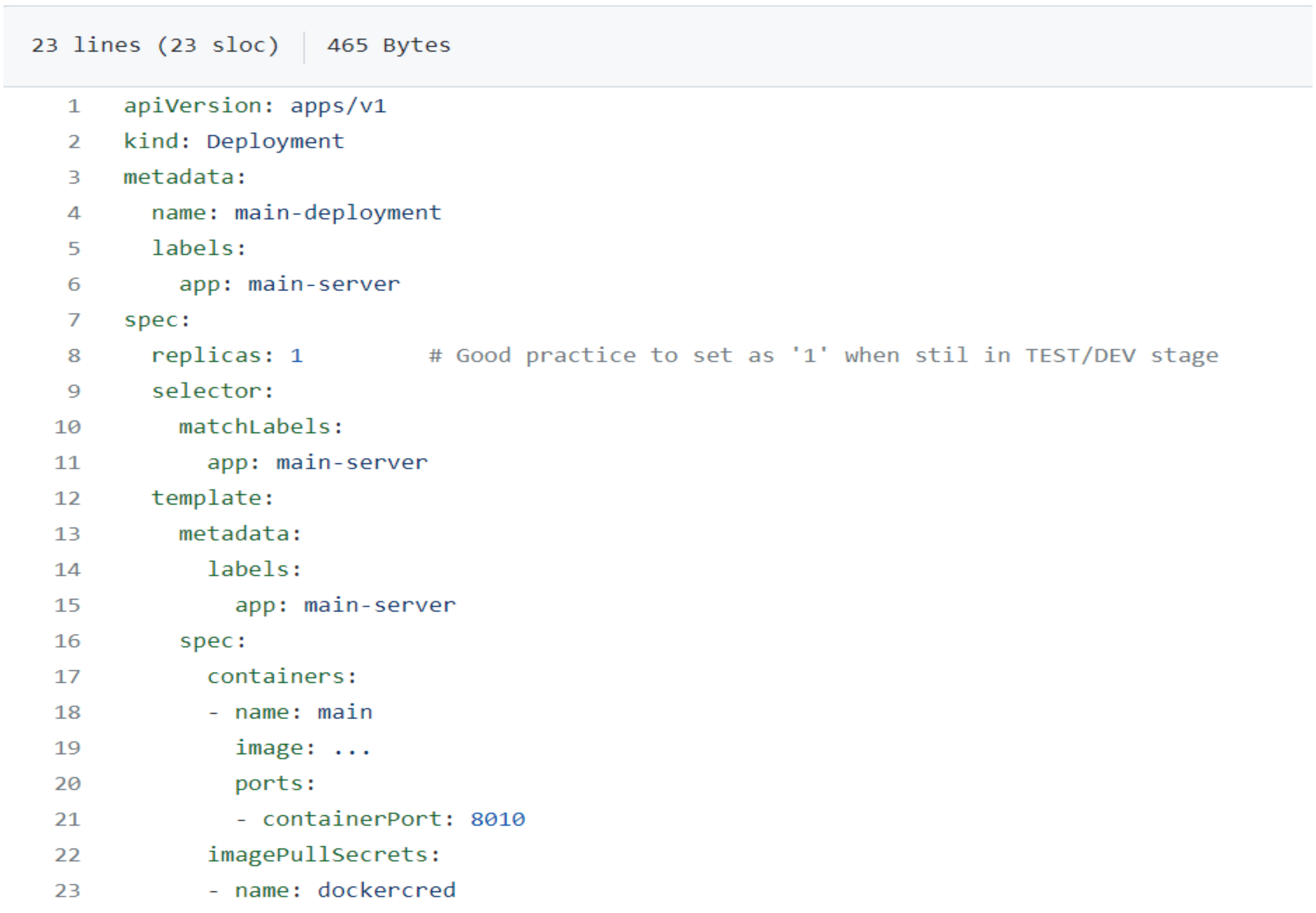
Manage your environments and add environment secrets

Finally, we will come the page above. All that is left is select the "New repository secret" option, and since GitHub secret file is key-value based, the keys should be inputted as "username" and "password" strings, and their actual values will be the values of the keys. With this, we have officially created our secret files, and in turn, our CI/CD workflow is successfully created, and should be working by now.

In summary, we will now have our docker images in our private Docker Hub repository, READY to be pulled by TCE for deployment.

Kubernetes YAML Configuration File

For this section, we will briefly describe the parameters that make up a YAML configuration file to be used to create a Kubernetes resource, for instance, a deployment.



```
23 lines (23 sloc) | 465 Bytes
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: main-deployment
  labels:
    app: main-server
spec:
  replicas: 1         # Good practice to set as '1' when stil in TEST/DEV stage
  selector:
    matchLabels:
      app: main-server
  template:
    metadata:
      labels:
        app: main-server
    spec:
      containers:
      - name: main
        image: ...
        ports:
        - containerPort: 8010
      imagePullSecrets:
      - name: dockercred
```

The above is a typical YAML file for deployment. While there are many lines of codes, the main things to pay attention to are:

1) "kind" – the value of this parameter depends on the TYPE of Kubernetes resources we are intending to create. For our above example, since we are looking to create a deployment resource, we input "Deployment" as its value. Other types include "Service", "Ingress", and "Statefulset".
2) "metadata" – The name of your resource, and its label. This information are mainly for the IDENTIFICATION of the Kubernetes resource for the Kubernetes API.
3) "replicas" – How many PODS you want to have for your application deployment. The more you have, the higher the availability of your application.
4) "image" – the Docker image name in your private Docker Hub repository. The format of the image should contain the registry of the private repository, and the image tag.

5) “imagePullSecrets” – an important parameter if you have created a secret resource to store the credentials of your private Docker Hub repository. With this secret resource, TCE will be AUTHENTICATED to access your private Docker Hub repository, and pull the images to containerize them. Without this, TCE will NOT be able to access the private repository.

The above are the main parameters to look out for when creating a YAML file for Kubernetes resources.

Conclusion

If you have reached this section, you should now have sufficient knowledge on how convenient and great Tanzu Community Edition really is, and the tools, such as Tanzu command lines, to build your first Tanzu Kubernetes cluster, be it unmanaged or managed. Furthermore, you have the flexibility to add abstract layers to your cluster with Tanzu's Package Management capabilities to keep your cluster performing at its best.

With this, have fun with Tanzu Community Edition!