DEPLOYMENT GUIDE

Domino & NVIDIA AI Enterprise Deployment Guide on VMware vSphere with Tanzu

Version 1.1 Rev A

Table of Contents

Introduction

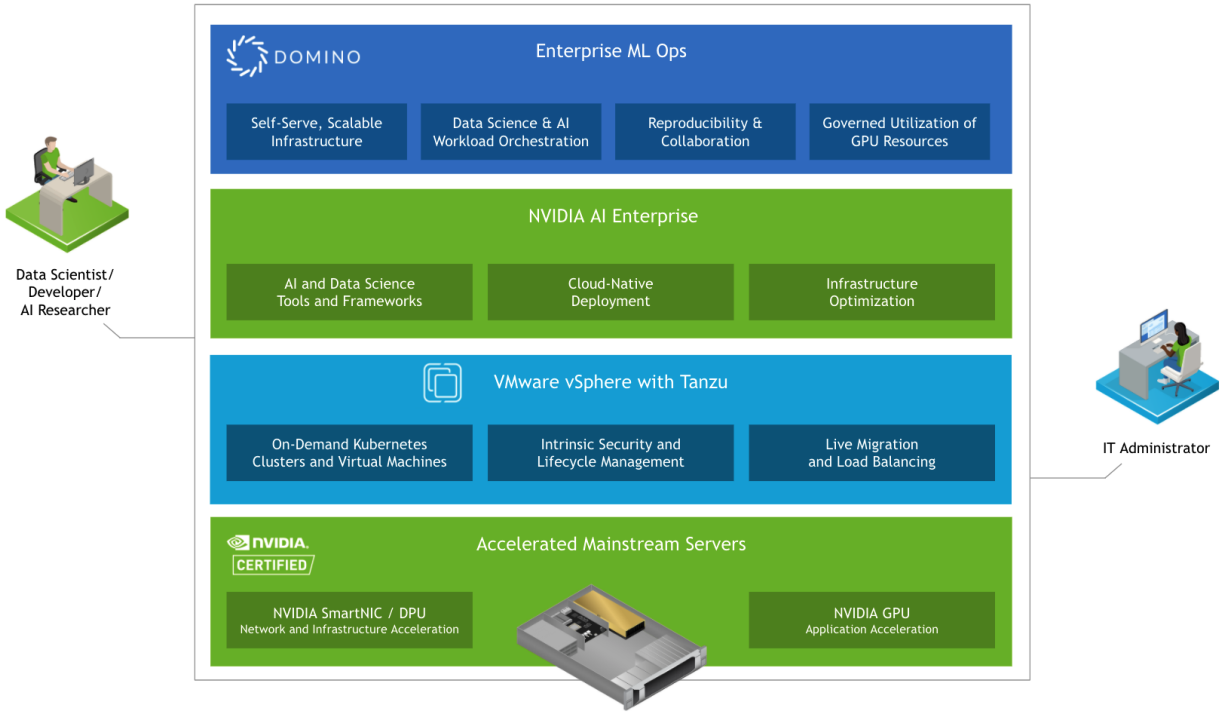
Scaling Data Science Isn't Easy

There is a reason why organizations need Domino Data Labs. They are faced with:

- **Chaotic tools and infrastructure.** Ungoverned tools, data, and infrastructure create operational, regulatory, and security risks. They drive up IT support burdens and cause friction in the data science lifecycle.
- **Silos causing complexity.** Data science has grown organically in most organizations, with different teams having different sets of tools, infrastructure, and processes to build models that may not align with IT and security standards. The complexity of supporting these silos drives up IT support burdens and increases risk.
- **Complex processes to operationalize models.** DevOps requirements and technical debt escalate when each data science team or model has its own stovepipe manual processes.

This document describes the Domino Data Lab's Enterprise MLOps Platform for NVIDIA AI Enterprise deployed into a Kubernetes cluster hosted by VMware vSphere and using VMware vSAN storage.

The Domino team worked together with the NVIDIA and VMware teams on the architectural vision and joint engineering effort to create this deployment guide. This paper is intended to provide planning, design considerations, and best practices for implementing the Domino Enterprise MLOps Platform on-premise, with NVIDIA products.

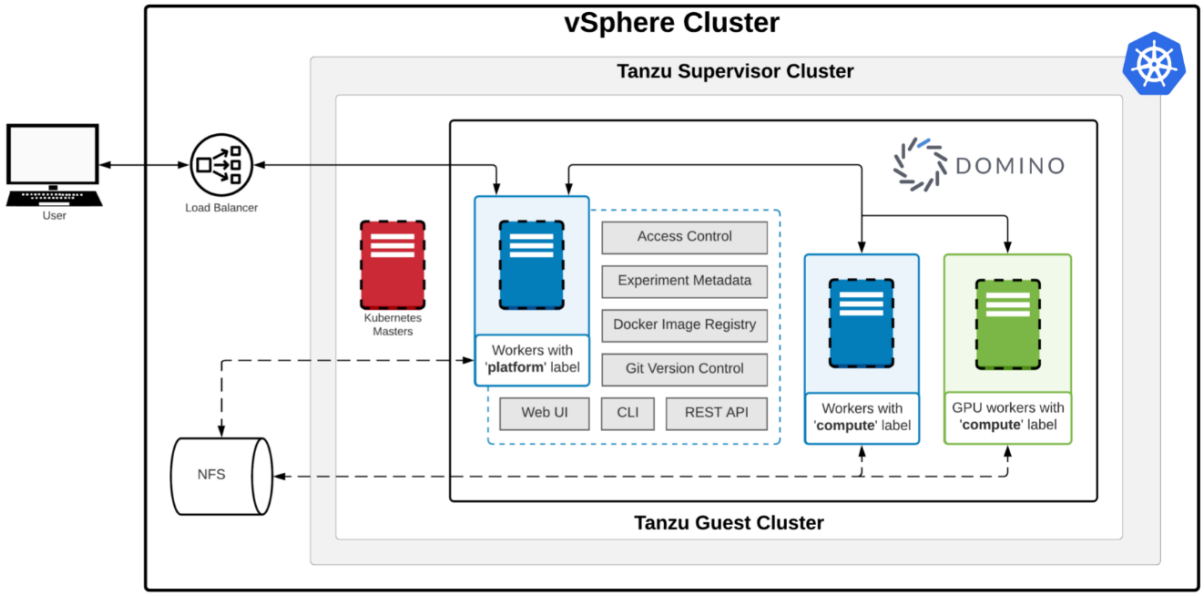


NVIDIA solutions, including AI Enterprise and DGX, are designed to deliver the capabilities you need to exceed today's needs while preparing you for the next wave of innovation. These systems are purpose-built to deliver performance, security, and agility in an open environment that won't limit your options down the road.

With the Domino Enterprise MLOps Platform, Domino can provide an integrated solution with hardware, software, and services, delivering reduced risk and faster time to value.
The intended audiences of this document are IT professionals, technical architects, sales engineers, and consultants to assist in planning, designing, and implementing Domino's platform.

Knowledge of containers, Kubernetes, cloud, and data center infrastructure architecture will be helpful.

Solution Summary



Domino Platform Architecture

The Domino Enterprise MLOps Platform runs as a containerized application in a Kubernetes environment. Kubernetes provisions and manages the containers containing the application components as needed by the Domino software. The VMware vSphere software manages the virtualized infrastructure on which Kubernetes is deployed. Using the recommended VMware vSAN storage cluster will ensure a container-native storage interface (CSI) for providing persistent storage as well as shared storage provided by vSAN file service for containers running in the Kubernetes environment.

The Domino application runs two types of workloads in Kubernetes, and there are different principles to sizing infrastructure for each:

- **Domino Platform**
  These always-on components provide user interfaces, the Domino API server, orchestration, metadata, and supporting services. The standard architecture runs the platform on a stable set of three VMs for high availability, and the capabilities of the platform are principally managed through vertical scaling, which means changing the CPU and memory resources available on those platform nodes and changing the resources requested by the platform components.

- **Domino Compute**
  These on-demand components run users' data science, engineering, and machine learning workflows. Compute workloads run on customizable collections of nodes organized into node pools. The number of these nodes can be variable and elastic, and the capabilities are principally managed through horizontal scaling, which means changing the number of nodes. However, when there are more resources present on compute nodes, they can handle additional workloads, and therefore there are benefits to vertical scaling.

Two container registries are used to provide the container images needed by the platform: The local secure registry which provides the containers for the core Domino application, and the NVIDIA AI Enterprise NGC private registry which provides the containers for deploying the NVIDIA GPU drivers, CUDA libraries, and data science tools and frameworks optimized for NVIDIA GPUs.

Solution Overview

The Domino / NVIDIA AI Enterprise solution utilizes many of the latest capabilities provided by NVIDIA EGX systems, NVIDIA AI Enterprise software, and VMware vSphere with Tanzu Kubernetes Grid Service.

Systems

The solution is designed to run on NVIDIA NGC-Ready and NVIDIA-Certified Systems. The reference architecture for this guide was deployed on the following certified hardware configuration:

3 x Dell EMC PowerEdge R750 NVIDIA-Certified server
- 2 x Intel Xeon Gold 6338 CPU @ 2.00GHz (32C/64T)
- 8 x 64 GB RAM (512 GB)
- 1 x Dell Ent NVMe P5600 MU U.2 1.6TB as vSAN Cache Disk
- 1 x Dell Ent NVMe CM6 RI 7.68TB as vSAN Capacity Disk
- 1 x Broadcom NetXtreme E-Series Advanced Dual-port 25Gb

1 x BOSS-S2 (Embedded) AHCI controller with 240GB SATA as ESXi booting device
2 x NVIDIA Ampere A100 PCIE 40GB

Virtualization

Virtualization and the Kubernetes cluster are provided by the VMware vSphere 7 Enterprise Plus with Tanzu Kubernetes Grid Service. Our reference solution requires vSphere Hypervisor (ESXi) version: 7.0U3c or later and vCenter: 7.0.3 or later.

For details on preparing vSphere for Tanzu, please refer to the following document:
Deploy an AI-Ready Enterprise Platform on VMware vSphere 7 with VMware Tanzu Kubernetes Grid Service

NVIDIA AI Enterprise Software

The NVIDIA AI Enterprise Software version 1.0 used in this solution is accessible from your NGC Private Registry. You access the software online at: https://ngc.nvidia.com once your organization has obtained a valid license. See your solution provider for details.

NVIDIA Virtual GPU

When adding physical nodes to the cluster it is important to install the NVIDIA Virtual GPU (vGPU) software on the ESXi host before adding it to the vSphere cluster. The NVIDIA vGPU Host Driver is available as a vSphere Installation Bundle (VIB) and can be downloaded from your NVIDIA AI Enterprise NGC Registry.

NVIDIA vGPU VIB version: NVIDIA-AIE_ESXi_7.0.2_Driver_470.82-1

Installation instructions are provided here:
Installing and configuring the NVIDIA VIB on ESXi

NVIDIA GPU Operator

The GPU Operator allows administrators of Kubernetes clusters to manage GPU nodes just like CPU nodes in the cluster. Instead of provisioning a special OS image for GPU nodes, administrators can rely on a standard OS image for both CPU and GPU nodes and then rely on the GPU Operator to provision the required software components for GPUs.

The NVIDIA GPU Operator version 1.9.0-beta or later is required; v1.9.1 or greater is preferred.

Installation of the NVIDIA GPU Operator is done by running the appropriate helm chart. The chart is downloaded with the following command:

```
helm fetch
https://helm.ngc.nvidia.com/ea-cnt/ext_operator_vgpu/charts/gpu-operator-v1.9.
1-rc.3.tgz --username='$oauthtoken' --password=<YOUR API KEY>
```

Complete installation instructions are available here:
Install NVIDIA GPU Operator

Storage Options

The Domino Enterprise MLOps Platform requires access to persistent storage that is attached to the Kubernetes cluster.

In general, automatic provisioning of persistent volumes must be configured.

For importing, exporting, and using Domino Datasets, The Domino Platform requires access to a persistent shared file system or object store.

The recommended storage facility when deploying to VMware vSphere is VMware vSAN. When using vSAN, it is recommended to use vSAN Ready Nodes which are also NGC qualified.

Storage options include:

- vSAN
- NFS
- vCloud Director / Cloudian
- NetApp ONTAP
- Ceph

Creating a Tanzu Kubernetes Cluster for Domino

Kubernetes Version

The Domino Data Science Platform will run on any CNCF compliant Kubernetes cluster of version 1.17 or later. The Domino NVIDIA AI Enterprise solution was validated on vSphere with Tanzu using TKG version v1.20.8---vmware.1-tkg.2.

Ingress and SSL

Domino will need to be configured to serve from a specific FQDN, and DNS for that name should resolve to the address of an SSL-termination load balancer with a valid certificate. The load balancer must target incoming connections on ports 80 and 443 to port 80 on all nodes in the Platform pool. This load balancer must support websocket connections.

Health checks for this load balancer should use HTTP on port 80 and check for 200 responses from a path of `/health` on the nodes.

NTP

In order to support SSO protocols, TLS connections to external services, intra-cluster TLS when using Istio, and to avoid general interoperability issues, the nodes in your Kubernetes cluster should have a valid Network Time Protocol (NTP) configuration. This will allow for successful TLS validation and operation of other time-sensitive protocols.

Storage Classes

Domino requires at least two storage classes, one for dynamic block storage and one for long term shared storage.

Block Storage

Domino requires high-performance block storage for the following types of data:

- Ephemeral volumes attached to user execution
- High-performance databases for Domino application object data

This storage needs to be backed by a storage class with the following properties:

- Supports dynamic provisioning
- Can be mounted on any node in the cluster
- SSD-backed recommended for fast I/O
- Capable of provisioning volumes of at least 100GB
- Underlying storage provider can support ReadWriteOnce semantics
- By default, this storage class is named `domino-disk`.

Shared Storage

Domino needs a separate storage class for long term storage for:

- Project data uploaded or created by users
- Domino Datasets
- Docker images
- Domino backups

This storage needs to be backed by a storage class with the following properties:

- Dynamically provisions Kubernetes PersistentVolume
- Can be accessed in ReadWriteMany mode from all nodes in the cluster
- Uses a `VolumeBindingMode` of `Immediate`

By default, this storage class is named `domino-shared`.

Node Pools

Domino requires a minimum of two node pools, one to host the Domino Platform and one to host Compute workloads. Additional optional pools can be added to provide specialized execution hardware for some Compute workloads.

Platform pool - Nodes hosting Domino platform services should have the label "`dominodatalab.com/node-pool: platform`"

Compute pool - Worker nodes for Domino user executions should have the label "`dominodatalab.com/node-pool: default`"

GPU pool - GPU worker nodes should be segregated into a gpu node pool by using the labels "`dominodatalab.com/node-pool: default-gpu`" and "`nvidia.com/gpu: true`"

Node pool labels are added automatically from the TanzuKubernetesCluster spec you provide for defining the cluster. You can verify node pool labels have been applied properly by running the following command after deploying the cluster:

```
$ kubectl get nodes –show-labels
```

VM Class Requirements

A VM Class can be thought of as a template for VM instances to be created in a Tanzu Kubernetes cluster. VM Classes provide a declarative method of defining your compute resources.

Platform Resources

In the Domino Kubernetes cluster you will need a control plane group for the Kubernetes API and control plane resources. This is the Tanzu Kubernetes cluster control plane.

There must also be a domino-platform node pool with at least 3 replicas to host the Domino platform and web services. These are the Domino MLOps platform application nodes.

The solution reference architecture included Tanzu Kubernetes cluster control plane nodes with the best-effort-medium VM Class:

- 2 vCPU
- 8 GB RAM

The domino-platform application node-pool uses a best-effort-2xlarge VM Class:

- 8 vCPU
- 64 GB RAM

Additionally, domino-platform nodes should be provided with a 100 GB local disk (see yaml).

You can create custom VM Classes for these groups as well as your compute resources. The sizes above should be considered as a minimum configuration.

Compute Resources

Compute VMs (worker nodes) in the cluster are where Domino user workloads can be run. These should have at a minimum the following specifications:

- 8 vCPU
- 32 GB RAM

400 GB local disk should be added to compute nodes to provide for a variety of workspace environment sizes and user work preferences.

GPU Enabled Workers

Compute instances with GPU resources are created using the same virtual machine classes. For more details on how VM Classes are created and used in Tanzu, see the following doc: Virtual Machine Classes for Tanzu Kubernetes Clusters

We specify the storage resources including storage class, number and size of local disks for the domino-platform and domino-compute nodes when we define the Tanzu Kubernetes cluster in the TanzuKubernetesCluster spec YAML file.

For most deployments compute resources will be substantially larger than these minimum specifications. It is recommended that you contact your Domino account representative or sales engineer to help you get a more accurate estimate of your sizing needs. A sizing guide for the Domino Platform is also available online on the Domino documentation website.

For more information on managing GPU and compute resources in Domino see the section: Managing Domino Compute Resources

Tanzu Kubernetes Cluster with vGPU Access

Here is an example TanzuKubernetesCluster spec yaml file that can be applied in the Tanzu supervisor cluster to create a suitable simple Tanzu Kubernetes cluster for Domino.

Example TanzuKubernetesCluster spec:

```
apiVersion: run.tanzu.vmware.com/v1alpha2
kind: TanzuKubernetesCluster
metadata:
  name: tkg-cluster-vgpu-domino
  namespace: domino-ns
spec:
  topology:
    controlPlane:
      replicas: 3
      storageClass: vsan-r1
      tkr:
```

```
      reference:
        name: v1.20.8---vmware.1-tkg.2
    vmClass: best-effort-medium
  nodePools:
  - name: domino-platform
    labels:
      dominodatalab.com/node-pool: "platform"
    replicas: 3
    vmClass: best-effort-2xlarge
    storageClass: vsan-r1
    volumes:
    - name: var-lib
      mountPath: /var/lib
      capacity:
        storage: 128Gi
  - name: mig-20
    labels:
     dominodatalab.com/node-pool: "mig20-gpu"
    replicas: 1
    storageClass: vsan-r1
    tkr:
      reference:
        name: v1.20.8---vmware.1-tkg.2
    vmClass: 16vcpu-64gram-mig-20c-vmxnet3
    volumes:
    - name: var-lib
      mountPath: /var/lib
      capacity:
        storage: 400Gi
  - name: mig-40
    labels:
      dominodatalab.com/node-pool: "default-gpu"
    replicas: 2
    storageClass: vsan-r1
    tkr:
      reference:
        name: v1.20.8---vmware.1-tkg.2
    vmClass: 16vcpu-64gram-mig-40c-vmxnet3
    volumes:
    - name: var-lib
      mountPath: /var/lib
      capacity:
        storage: 400Gi
  - name: nongpuworkers
    labels:
      dominodatalab.com/node-pool: "default"
      domino/build-node: "true"
    replicas: 1
    storageClass: vsan-r1
    tkr:
      reference:
```

```
      name: v1.20.8---vmware.1-tkg.2
    vmClass: best-effort-2xlarge
    volumes:
    - name: var-lib
      mountPath: /var/lib
      capacity:
        storage: 400Gi
```

Installing Domino

The install automation tools are delivered as a Docker image, and need to run on an installation workstation that meets the following requirements:

- Docker installed
- Kubectl service account access to the cluster
- Access to quay.io to download the installer image*
- Access to the NVIDIA image repository at nvcr.io*

*Solutions for air-gapped installation are available, see your solution provider for details.

Accessing the Installer

Additionally, you will need credentials for an installation service account that can access the Domino upstream image repositories in quay.io. Throughout these instructions, these credentials will be referred to as `$QUAY_USERNAME` and `$QUAY_PASSWORD`. Contact your Domino account team if you need new credentials.

Log in to quay.io with the credentials described above

```
$ docker login quay.io
```

Pull the fleetcommand-agent image

```
$ docker pull quay.io/domino/fleetcommand-agent:v47
```

Creating the Domino Configuration File

Here is a template configuration file named `domino.yml`:

```
schema: '1.0'
name: <DESCRIPTIVE_NAME>
version: 4.6.2
hostname: <DOMINO_DEPLOYMENT_FQDN>
pod_cidr: '0.0.0.0/0'
ssl_enabled: false
ssl_redirect: false
request_resources: true
```

```
enable_network_policies: true
enable_pod_security_policies: true
global_node_selectors: {}
create_restricted_pod_security_policy: true
kubernetes_distribution: cncf
istio:
  enabled: false
  install: false
  cni: true
  nginx_annotations: true
namespaces:
  platform:
    name: domino-platform
    annotations: {}
    labels:
      domino-platform: 'true'
  compute:
    name: domino-compute
    annotations: {}
    labels:
      domino-compute: 'true'
  system:
    name: domino-system
    annotations: {}
    labels: {}
  istio:
    name: istio-system
    annotations: {}
    labels: {}
ingress_controller:
  create: true
  gke_cluster_uuid: ''
  class_name: nginx
storage_classes:
  block:
    create: false
    name: vsan-r1
    type: vsphere-volume
    access_modes:
    - ReadWriteOnce
    base_path: ''
    default: false
    parameters: {}
  shared:
    create: true
    name: domino-shared
    type: nfs
    access_modes:
    - ReadWriteMany
    volume_capacity: 2Ti
    efs:
```

```
      region: ''
      filesystem_id: ''
    nfs:
      server: '<NFS_SERVER_NAME>'
      mount_path: '/vsanfs/domino-share'
      mount_options: [ "nfsvers=4.1", ]
    azure_file:
      storage_account: ''
blob_storage:
  projects:
    type: shared
    s3:
      region: ''
      bucket: ''
      sse_kms_key_id: ''
      access_key_id: ''
      secret_access_key: ''
    azure:
      account_name: ''
      account_key: ''
      container: ''
    gcs:
      bucket: ''
      service_account_name: ''
      project_name: ''
  logs:
    type: shared
    s3:
      region: ''
      bucket: ''
      sse_kms_key_id: ''
      access_key_id: ''
      secret_access_key: ''
    azure:
      account_name: ''
      account_key: ''
      container: ''
    gcs:
      bucket: ''
      service_account_name: ''
      project_name: ''
  backups:
    type: shared
    s3:
      region: ''
      bucket: ''
      sse_kms_key_id: ''
      access_key_id: ''
      secret_access_key: ''
    azure:
      account_name: ''
```

```
      account_key: ''
      container: ''
    gcs:
      bucket: ''
      service_account_name: ''
      project_name: ''
  default:
    type: shared
    s3:
      region: ''
      bucket: ''
      sse_kms_key_id: ''
      access_key_id: ''
      secret_access_key: ''
    azure:
      account_name: ''
      account_key: ''
      container: ''
    gcs:
      bucket: ''
      service_account_name: ''
      project_name: ''
    enabled: false
autoscaler:
  enabled: false
  cloud_provider: aws
  auto_discovery:
    cluster_name: domino
    tags: []
  groups:
  - name: ''
    min_size: 0
    max_size: 0
  aws:
    region: ''
  azure:
    resource_group: ''
    subscription_id: ''
spotinst_controller:
  enabled: false
  token: ''
  account: ''
external_dns:
  enabled: false
  provider: aws
  domain_filters: []
  zone_id_filters: []
  txt_owner_id: ''
git:
  storage_class: vsan-r1
email_notifications:
```

```
  enabled: true
  server: <SMTP_SERVER>
  port: 465
  encryption: ssl
  from_address: <FROM_EMAIL>
  authentication:
    username: '<SMTP_USERNAME>'
    password: '<SMTP_PASSWORD>'
monitoring:
  prometheus_metrics: true
  newrelic:
    apm: false
    infrastructure: false
    license_key: ''
helm:
  version: 3
  host: ''
  namespace: ''
  insecure: false
  username: ''
  password: ''
  skip_daemonset_validation: false
  tiller_image: ''
  prefix: ''
  cache_path: '/app/charts'
private_docker_registry:
  server: quay.io
  username: '<QUAY_USERNAME>'
  password: '<QUAY_PASSWORD>'
internal_docker_registry:
  enabled: true
  s3_override:
    region: ''
    bucket: ''
    sse_kms_key_id: ''
    access_key_id: ''
    secret_access_key: ''
  gcs_override:
    bucket: ''
    service_account_name: ''
    project_name: ''
  azure_blobs_override:
    account_name: ''
    account_key: ''
    container: ''
external_docker_registry: null
telemetry:
  intercom:
    enabled: false
  mixpanel:
    enabled: false
```

```
    token: ''
gpu:
  enabled: false
fleetcommand:
  enabled: false
  api_token: ''
certificate_management:
  enabled: true
teleport_kube_agent:
  enabled: false
  proxyAddr: teleport.domino.tech:443
  authToken: eeceeV4sohh8eew0Oa1aexoTahm3Eiha
image_caching:
  enabled: true
workbench:
  enabled: true
modelmonitor:
  enabled: false
services:
  nginx_ingress:
    chart_values:
      controller:
        replicaCount: 2
        kind: Deployment
        hostNetwork: false
        config:
          use-proxy-protocol: 'false'
        service:
          targetPorts:
            http: http
          enabled: true
          type: LoadBalancer
  forge:
    version: 0.20.2
    image:
      tag: v0.4.2-replicator
    chart_values:
      config:
        fullPrivilege: true
```

Edit the configuration file with all necessary details about the target cluster, storage systems, and hosting domain. Read the configuration reference for more information about available keys, and consult the configuration examples for guidance on getting started. You will be provided a quay.io login with access to pull the Domino container images.

Note that you should set the value of `name` to something that identifies the purpose of your installation and contains the name of your organization.

Running the Installer

Execute a dry-run with the following command:

```
$ docker run --rm -v $(pwd):/install -v $(pwd)/logs:/app/logs -v
(pwd)/cache:/app/.appr_chart_cache quay.io/domino/fleetcommand-agent:v47 run
--dry --file /install/domino.yml
```

Run the Domino installer with the following command:

```
$ docker run --rm -v $(pwd):/install quay.io/domino/fleetcommand-agent:v47
run --file /install/domino.yml
```

When the installation completes successfully, you should see a message that says:

```
2019-11-26 21:20:20,214 - INFO - fleetcommand_agent.Application - Deployment
complete.
Domino is accessible at $YOUR_FQDN
```

However, the application will only be accessible via HTTPS at that FQDN if you have configured DNS for the name to point to an ingress load balancer with the appropriate SSL certificate that forwards traffic to your platform nodes.

Post Installation Steps

Check the Cluster

You can check the status of the cluster after the installer has run by executing the following command and verifying the pod status as ‘Running’ or ‘Completed’:

```
$ kubectl get pods --namespace domino-platform
```

Verify Node Labels

Verify that the cluster nodes are labeled appropriately.

```
$ kubectl get nodes --show-labels
```

Add a User Account

See the section on user authentication for instructions on how to add a user account.

Configuring Domino

User Authentication

Domino uses Keycloak, an enterprise-grade open source authentication service to manage users and logins. Keycloak runs in a pod in the Domino Platform. There are three modes you can use for identity management in Domino:

1. Local usernames and passwords
2. Identity federation to LDAP / AD
3. Identity brokering to a SAML provider for SSO

To log in as the default keycloak administrator user, you will need kubectl access to the cluster to retrieve the password from a Kubernetes secret called `keycloak-http`.

```
$ kubectl get secret -n domino-platform keycloak-http
--output=jsonpath="{.data.password}” | base64 -d
```

Use this password and the username “keycloak” to login to the Keycloak UI at:

`https://<domino-domain>/auth/`

Note that the trailing / in the URL is required.

Keycloak will be configured automatically by Domino with a realm named DominoRealm that will be used for Domino authentication. When reviewing or changing settings for Domino authentication, ensure that you have DominoRealm selected in the upper left.

Local Username and Password Configuration

The simplest option for authentication to Domino is to use local usernames and passwords. In this case all user information is stored by Keycloak in the Postgres database, and there is no federation or brokering to other identity providers.

In this mode the key settings are on the Login tab of the DominoRealm settings page.

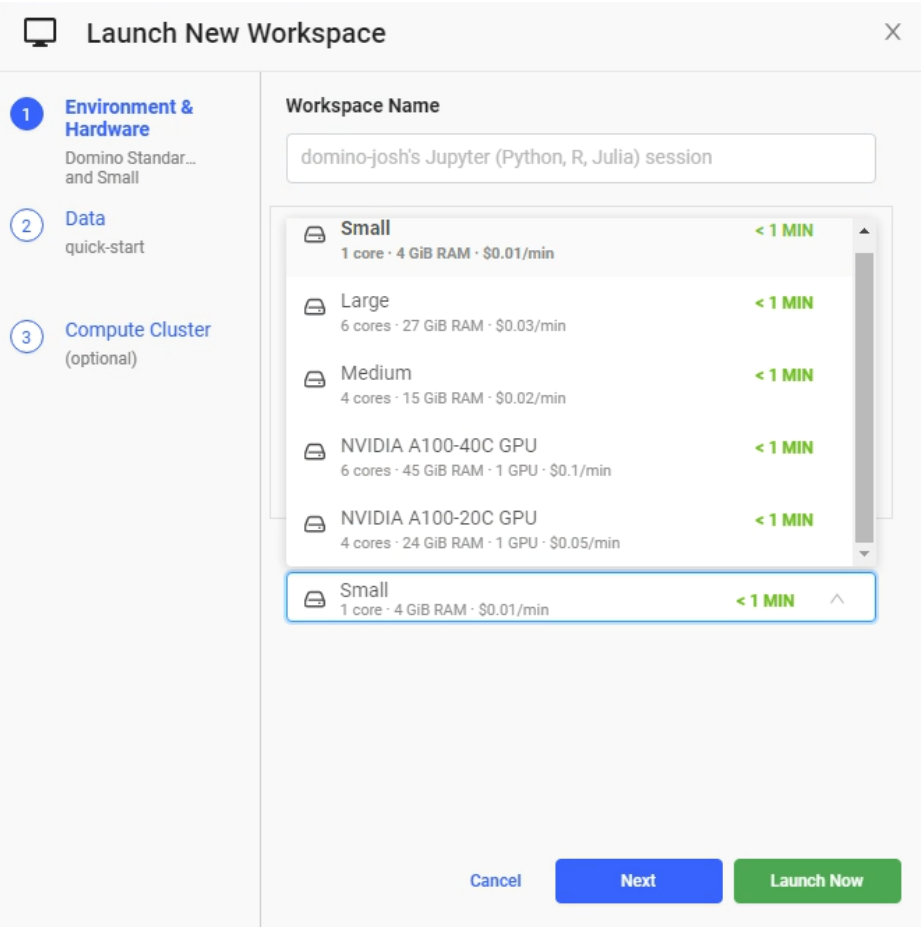
The one setting on this tab that is not supported is Email as username as that would automatically use email as username and Domino currently does not support that as a valid username. Note also that if you want to use the Verify Email option, an SMTP connection must be configured in the Email tab.

You can add, edit, and deactivate local users from the Users menu.

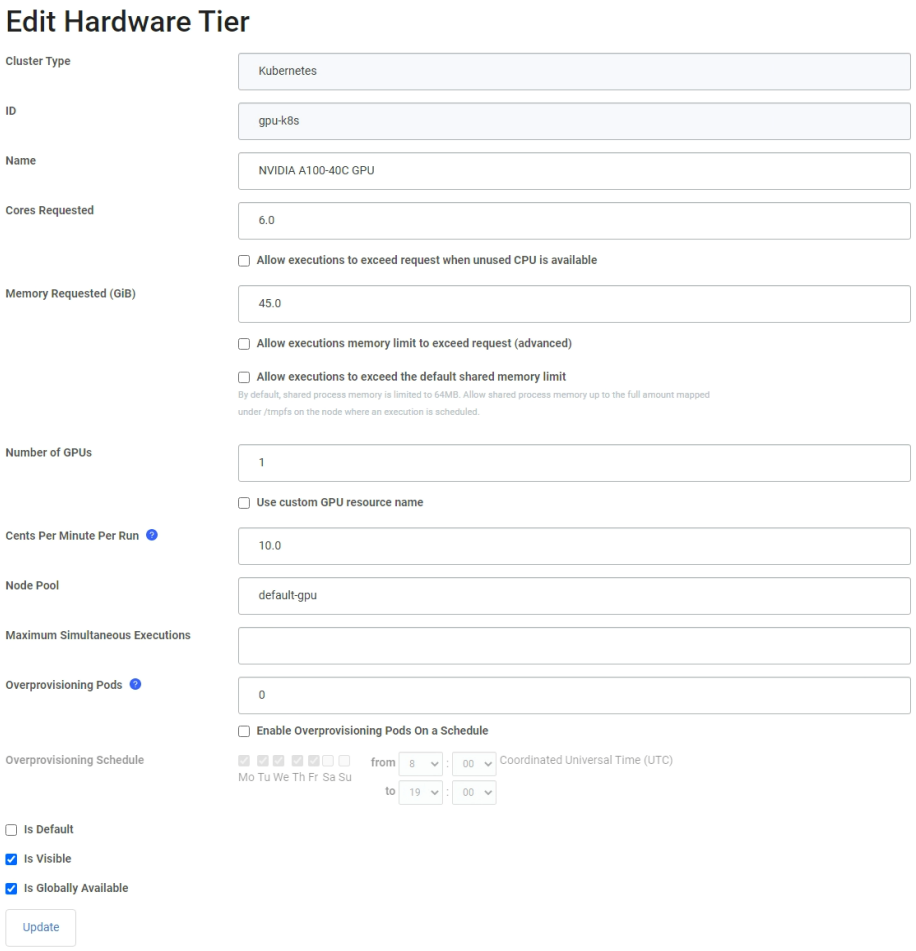
Managing Domino Compute Resources

Hardware Tiers

When launching a Domino execution, users specify a Hardware Tier (HWT) that determines the resources available for their execution. At a Kubernetes level, HWTs specify the desired amount of CPU, Memory, and GPU for the pod as well as the node pool it should run on. Here is a screenshot of the HWT dropdown for starting an interactive Workspace execution in Domino.



Users can see at a glance if a HWT is available and an admin-provided hourly cost of that HWT, allowing them to make complete decisions about what resources to use. Admins of a Domino deployment can create these HWTs as described here. Through the Domino UI, they can set the resource requests for each HWT and the label corresponding with an existing node pool in the Tanzu Kubernetes cluster. Below is an example configuration for a HWT with an NVIDIA A100-40C MIG profile.



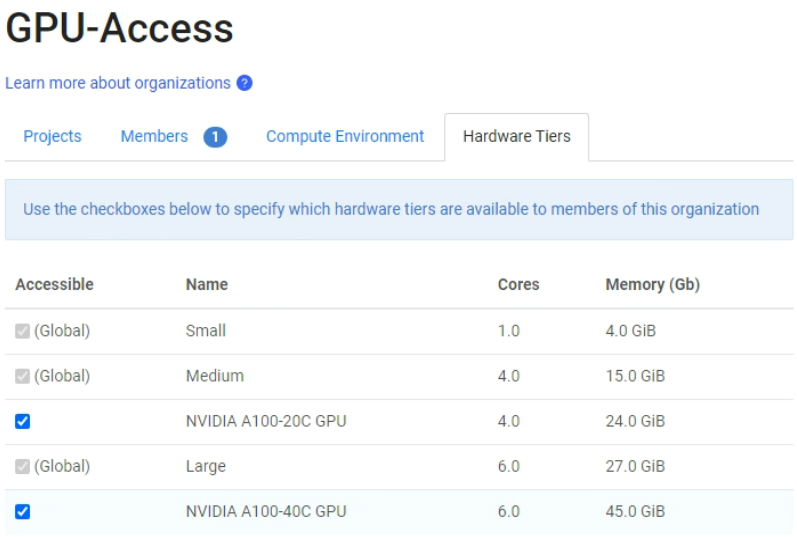
Sizing Hardware Tiers

When designing HWTs, you need to take into account what resources will be available on a given node pool VM when Domino submits your workload for execution. Not all memory and CPU cores will be available due to overhead, which depends on exactly how your infrastructure was configured. As a starting point, per-VM overhead is roughly 0.5 of one vCPU and 0.5 of one GB RAM, and per-workload overhead is roughly 1 vCPU and 1.5 GB RAM. As an example, if you want to create the largest HWT that will fit on an 8 vCPU, 32 GB VM, you should expect to end up with a 6.5 vCPU, 30 GB HWT.

Since workers in the compute node pool are often much larger than that example, you will likely want to create smaller sizes as well. Here we recommend dividing HWTs in half until you get to a size that is too small to be useful (remember to account for per-workload overhead!). If we were to create the next smaller HWT for the example above, we would get 2.75 vCPU, 14.25 GB. Dividing a node pool worker by powers of two creates more opportunities for workloads to be fit in the same VM, increasing utilization efficiency of resources in the cluster.

Controlling Resource Usage

Domino offers a number of mechanisms to control HWT usage such as setting limits on the number of concurrent executions per-HWT (via a field in the HWT configuration UI) or per-user (via an admin settings page). It is also possible to restrict high-cost HWT access to a subset of users, by limiting it to an Organization (Domino user group) where they are a member. Organizations can be either manually created or propagated from a SAML IdP. Once a HWT is created with the **Globally Available** option unchecked, an admin can add it to an existing organization as seen below.



Adding NVIDIA AI Enterprise Containers to Domino

Add NVIDIA AI Enterprise Credentials to Domino

For the Domino deployment to pull the NVIDIA AI Enterprise container images from NGC, you will need to make the credentials of a service account NGC user available to Domino Compute Environment builds via a Kubernetes secret.

First, you will want to back up the existing secret that contains credentials for other images used by Domino and make a copy.

```
$ kubectl -n domino-platform get secret domino-quay-repos -o
jsonpath='{.data.\.dockerconfigjson}' | base64 -d > repos-auths.json
$ cp repos-auths.json repos-auths-new.json
```

In `repos-auths-new.json` add the new (bolded) credential including the comma.

```
{"auths": { "quay.io": {"username": "...", "password": "...", "email": "."},
"nvcr.io": {"username": "$oauthtoken", "password": "<<INSERT_API_TOKEN>>",
"email": "."}}}
```

Finally, replace the Kubernetes secrets with the updated json.

```
$ kubectl -n domino-platform create secret generic domino-quay-repos \
--type=kubernetes.io/dockerconfigjson \
--from-file=.dockerconfigjson=./repos-auths-new.json \
--dry-run -o yaml | kubectl replace -f -
```

```
$ kubectl -n domino-compute create secret generic domino-quay-repos \
--type=kubernetes.io/dockerconfigjson \
--from-file=.dockerconfigjson=./repos-auths-new.json \
--dry-run -o yaml | kubectl replace -f -
```

Create Domino Compute Environments with NVIDIA AI Enterprise Containers

A Compute Environment is a Domino abstraction on top of a Docker image. Each run takes place in an isolated Docker container based on the Environment associated with your project. Container images from the NVIDIA NGC Enterprise Catalog can be used as the base image for one of these environments. See details on how to create Domino Environments here.

Domino has a small set of requirements that need to be added to the images before they can run as an interactive workspace. You can add these dependencies by adding the following to the start of the **Dockerfile Instructions**.

```
#vvvDOMINO DEPENDENCIESvvv#
USER root:root
RUN cd / && wget 'http://nvaie-domino.s3.amazonaws.com/workspace-utils.tgz'
&& tar -xf workspace-utils.tgz

# Validate image has documented dependencies
https://docs.dominodatalab.com/<TBD>
RUN /opt/domino/bin/pre-check.sh

# Configure user for Domino executions
RUN /opt/domino/bin/init.sh

# Validate all injected dependencies are functional
RUN /opt/domino/bin/validate.sh
#^^^DOMINO DEPENDENCIES^^^#
```

The above code pulls a tar file from a public S3 bucket with the dependencies required by Domino for a workspace session and packages them into the image.

You will also need to add the following to the **Pluggables** section, which tells Domino how to launch an IDE when the environment is used as a workspace.

```
jupyterlab:
  title: "JupyterLab"
  iconUrl: "/assets/images/workspace-logos/jupyterlab.svg"
  start: ["/opt/domino/bin/jupyterlab-start.sh"]
  httpProxy:
    internalPath:
"/{{ownerUsername}}/{{projectName}}/{{sessionPathComponent}}/{{runId}}/{{#if
pathToOpen}}tree/{{pathToOpen}}{{/if}}"
    port: 8888
    rewrite: false
    requireSubdomain: false
vscode:
  title: "vscode"
  iconUrl: "/assets/images/workspace-logos/vscode.svg"
  start: [ "/opt/domino/bin/vscode-start.sh" ]
  httpProxy:
    port: 8888
    requireSubdomain: false
```

Additional Resources

- Domino Data Lab admin documentation
- Domino Data Lab user documentation
- NVIDIA AI Enterprise documentation
- NVIDIA NGC Enterprise Catalog
- VMware Tanzu Kubernetes Grid Service with vGPU Deployment Guide

Revisions

Version	Release Date	Notes
1.1 Rev A	TBD	Updated storage requirements in TanzuKubernetesCluster yaml
1.1	January 19th, 2022	First version

Notices

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. Domino Data Lab, Inc. (“Domino”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. Domino shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality. Domino reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice. Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete. Domino products are sold subject to the Domino standard terms and conditions of sale supplied at the time of the order between Customer and Domino, unless otherwise agreed in an individual sales agreement signed by authorized representatives of Domino and customer (“Terms of Sale”). Domino hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the Domino product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

Trademarks

Domino® and the Domino logo, are trademarks and/or registered trademarks of Domino Data Lab, Inc. in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright