DEPLOYMENT GUIDE

Run:ai & NVIDIA AI Enterprise Deployment Guide on VMware vSphere with Tanzu

Version 3.0

Table of Contents

Introduction

Run:ai provides AI Infrastructure made simple

Run:ai' Atlas Platform enables IT organizations to build their AI infrastructure with cloud-like resource accessibility and management, on any infrastructure, and enable researchers to use any machine learning and data science tools they choose. Run:AI's platform builds off of powerful distributed computing and scheduling concepts from High Performance Computing (HPC), but is implemented as a simple Kubernetes plugin. Our AI Cloud Platform speeds up data science workflows and creates visibility for IT teams who can now manage valuable resources more efficiently, and ultimately reduce idle GPU time.

- **Gain Visibility & Control** Simplify management of GPU allocation and ensure sharing between users, teams, and projects according to business policies & goals. Run:ai brings a cloud-like experience to resource management wherever your resources are: on-premises, cloud or hybrid cloud.
- **Built for Cloud-Native** Super-powerful scheduling built as a simple Kubernetes plug-in, built from the ground up to work with containers and cloud-native architectures. Make use of fractional GPUs, integer GPUs, and multiple-nodes of GPUs, for distributed training on Kubernetes. Easily leverage NVIDIA's NGC catalog to deploy various GPU optimized accelerators for your relevant use cases.
- **Improve ROI >2x** Improve utilization by more than 2X, and significantly increase ROI on existing GPU and CPU infrastructure. Run workloads on fractions of GPUs, converting spare capacity to speed and increasing infrastructure efficiency.

Run:ai allows IT organizations to remove static resource allocations. Static compute allocations dramatically reduce researcher productivity and slow down their ability to bring AI solutions to market. Run:ai's Atlas Platform for AI Clouds speeds up data science initiatives by pooling all available resources and then dynamically allocating resources based on need - to maximize accessible compute, regardless where these resources reside (on-premises, cloud or hybrid).

- **Self Serve** No need to go to IT for access to compute, researchers gain self-service availability of all resources. Accelerates compute-intensive AI projects by more than 2x.
- **Compute accessibility** Automates provisioning of pooled GPU across teams, users, clusters and nodes. Access the compute you need, where and when you need it.
- **One-click distributed computing** Thousands of experiments are launched, managed, queued and provisioned on GPUs automatically. One customer ran 6790 concurrent HPO jobs, the system simply queued and ran them as resources became available.

Ensure AI Models are delivered from the Lab to Production, It's common knowledge that most AI models don't make it into the wild. And many companies simply don't have the tools for managing AI infrastructure in a way that maps AI initiatives to business goals. As a result, data science teams are not able to work to their full potential, seriously limiting the organization's ability to innovate. Align your AI initiatives to business goals, by solving significant compute infrastructure inefficiencies that slow down AI adoption and growth.

- **Faster modeling** Make data science more productive with the compute power to run more experiments and deploy more AI initiatives.
- **A new stack, new tools** Run:ai enables the evolution to modern, cloud-native AI infrastructure. Run:AI's Kubernetes-based platform works with all popular data science tools.
- **Maximize ROI** Extract more than 2X efficiency from existing infrastructure. Ensures value from hardware resources.

This document serves to provide a validated deployment guide for deploying Run:ai Atlas Platform on NVIDIA AI Enterprise leveraging a VMware vSphere Tanzu cluster.

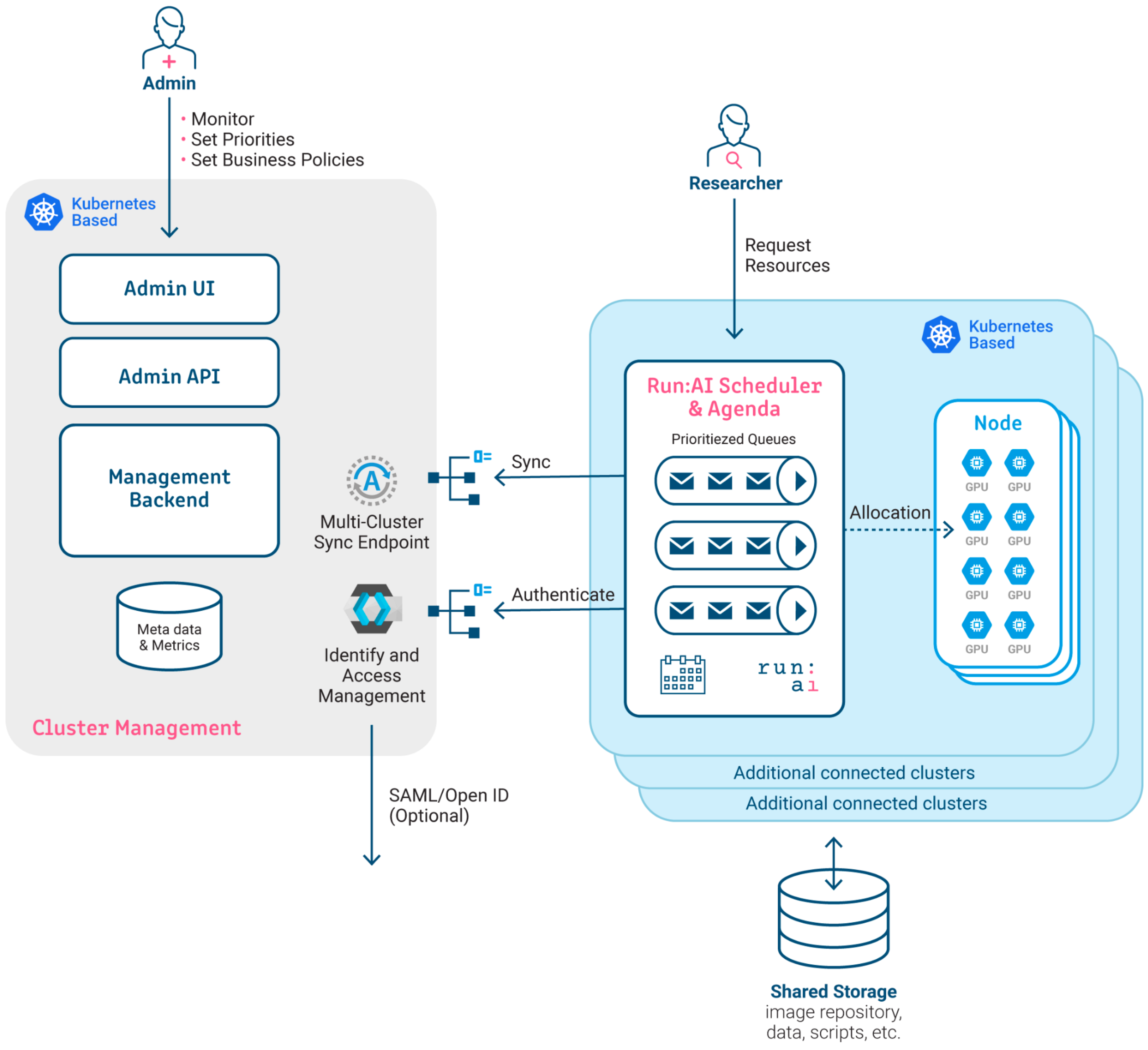
Run:ai collaborated with NVIDIA and VMware to deliver a platform that delivers on demand GPU and compute resources to researchers and data scientists in an on-demand fashion while also providing visibility into compute resources

Run:ai Atlas platform gives IT the control and visibility they need into their AI and ML infrastructure while also abstracting away all of the complex underlying infrastructure, allowing researchers and data scientists to leverage the tools of their choosing to drive innovation within the enterprise.

Architecture

Solution Overview

Run:ai is installed over a Kubernetes Cluster. This serves as the infrastructure to host Run:ai and the backbone to pool GPU resources from which researchers and data scientists will be granted guaranteed access to. Run:ai can be installed on various container orchestrators from Tanzu, OpenShift, Rancher, and Vanilla Kubernetes to cloud service based orchestrators like Google Kubernetes Engine (GKE), Azure Kubernetes Service (AKS), and Amazon's Elastic Kubernetes Service (EKS). The Atlas platform provides multi cluster support to allow IT administrators to manage multiple distributed clusters from a single pane of glass in addition to giving researchers and data scientists access to additional compute capacity.



Run:ai consists of two components. The Run:ai cluster (pictured above, right) and the Run:ai control plane or backend (left). Each of these components include options for how they can be deployed with the options being explained below.

Run:ai cluster

The Run:ai cluster includes components that are installed in the Kubernetes cluster and orchestrate the workloads local to the deployed infrastructure. The Run:ai scheduler extends the capabilities of the default Kubernetes scheduler while not requiring the replacement of the default scheduler. It uses business rules based on project quotas to schedule workloads sent by researchers and data scientists. Fractional GPU, a Run:ai technology, allows Researchers to allocate subsets of a GPU rather than a whole GPU, enhancing the underlying resource utilization. Additionally, the Run:ai agent and other monitoring tools are responsible for sending monitoring data to the Run:ai control plane.

The cluster components allow researchers and data scientists to directly submit and interact with workloads that utilize the compute infrastructure. The cluster includes components that provide the scheduling and orchestration capabilities to ensure fair sharing of GPU resources while also allowing bursting of capacity to eliminate idle resources. The cluster components also ensure that any policies set at an administrative level are enforced to ensure the sharing of GPU resources based on business priorities while also maintaining secure operation of the cluster.

Researchers and data scientists have various methods for submitting and interacting with their workloads. They have the ability to submit machine learning workloads via the Run:ai Command-Line Interface (CLI), directly by sending YAML files to Kubernetes, via the Kubernetes or Run:ai API, or via the Run:ai researcher user interface (GUI). The researcher user interface allows for the simple submission of workloads based on templates that prefill necessary fields for workload submission. This allows administrators to provide a streamlined process for onboarding new users to the platform. Templates delivered through the Run:ai web UI allow for a streamlined approach to deploying Jupyter notebooks while the UI also allows for researchers and data scientists to connect directly to their notebook once it is running.

In addition to all of the tooling that Run:ai provides out of the box, it is an open platform in terms of the tools that your data scientists and researchers will be able to utilize. The advantage of having Run:ai is it can provide various scheduling and orchestration enhancements to any containerized workload. This allows researchers to continue to leverage the tools they are familiar with while gaining the scheduling and orchestration benefits of Run:ai. Validated integrations include: Kubeflow, MLflow, Airflow, JupyterHub and Argo Workflows while also providing straightforward resource allocation for Visual Studio Code, PyCharm, Jupyter notebooks and Tensorboard.

Run:ai control plane (backend)

The Run:ai control plane aggregates monitoring and performance information related to compute infrastructure and running workloads. The control plane allows for the joining of multiple clusters to a single backend so you can manage and monitor multiple clusters from a single pane of glass. The control plane component for Saas customers is fully maintained by Run:ai. The control plane is also the location where administrative changes can be made to the platform. Some of these administrative changes include integration with SSO, managing of projects and resource quotas, toggling of features and managing end users of the platform.

The cluster sends information to the control plane for the purpose of control as well as monitoring through dashboards. Run:ai maintains strict policies around data retention on its Saas platform. Run:ai does not send deep-learning artifacts to the cloud. As such any code, images, container logs, training data, models, checkpoints and the like, stay behind corporate firewalls. More information on the Run:ai data privacy and policy can be found at the following link: https://docs.run.ai/home/data-privacy-details/?h=data+pri.

The directions below include installation for the SaaS version of Run:ai. If you are interested in installing the self hosted version of Run:ai where the control plane is hosted in your Kubernetes cluster, please follow the instructions here:
https://docs.run.ai/admin/runai-setup/self-hosted/overview/

Prerequisites

General Run:ai prerequisites

The latest Run:ai version supports Kubernetes versions 1.21 through 1.24. For RedHat OpenShift, Run:ai supports OpenShift versions 4.8 to 4.10.

NVIDIA's GPU Operator is also a prerequisite for using Run:ai. Installation instructions for the GPU Operator are included below.

Run:ai also leverages Prometheus as its time series database. The Run:ai cluster installation will, by default, install Prometheus, but it can also connect to an existing Prometheus instance installed by an organization. In the latter case, it's important to: verify that both Prometheus Node Exporter and kube-state-metrics are installed. It is also important to understand how Prometheus has been installed, whether directly or with the Prometheus Operator. The distinction is important during the Run:ai cluster installation.

Run:ai provides unique model serving capabilities as well. This allows you to serve and scale Triton and other inference processes on the exact GPU memory footprint required by the process. More information on the requirements for enabling Run:ai's model serving capabilities can be found at the following link:
https://docs.run.ai/admin/runai-setup/cluster-setup/cluster-prerequisites/#inference

Distributed training is the ability to run workloads on multiple nodes (not just multiple GPUs on the same node, rather multiple GPUs on multiple nodes). Run:ai provides this capability via Kubeflow MPI. If you need to leverage this functionality, you will need to install the Kubeflow MPI Operator via the application of the command below:

- `kubectl apply -f https://raw.githubusercontent.com/kubeflow/mpi-operator/master/deploy/v2beta1/mpi-operator.yaml`

For production installs, it is recommended to leverage Run:ai system nodes to reduce downtime and save CPU cycles on GPU Machines. Run:ai recommends that clusters contain two or more CPU worker nodes, designated for Run:ai Software. These nodes do not have to be dedicated to Run:ai however from a resource perspective; the Run:ai components require: 4 CPUs, 8GB of RAM and 50GB of Disk space.

Run:ai also a few network requirements for pulling container images as well as pushing data to the Saas control plane. These network requirements are defined in the following link: https://docs.run.ai/admin/runai-setup/cluster-setup/cluster-prerequisites/#network-requirements

Helm is another prerequisite for installing the Run:ai cluster as well as the NVIDIA GPU Operator. Helm is a package manager that helps you manage Kubernetes applications using charts that define, install, and upgrade even the most complex Kubernetes applications. Installation instructions for Helm are provided below in addition to full documentation at the link provided: https://helm.sh/docs/intro/install/

- `curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3`
- `chmod 700 get_helm.sh`
- `./get_helm.sh`

For the most up to date prerequisites please refer to the following link: https://docs.run.ai/admin/runai-setup/cluster-setup/cluster-prerequisites/

Running kubectl and CLI commands

The commands in the deployment guide are expected to be executed from one of the control nodes of the cluster or from a node that has the kubeconfig file of the Tanzu cluster exported locally to it.

For more information on kubeconfig files and how to use them, please review the official Kubernetes documentation:
https://kubernetes.io/docs/concepts/configuration/organize-cluster-access-kubeconfig/

Deploy a Tanzu Kubernetes cluster

VMware vSphere with Tanzu overview

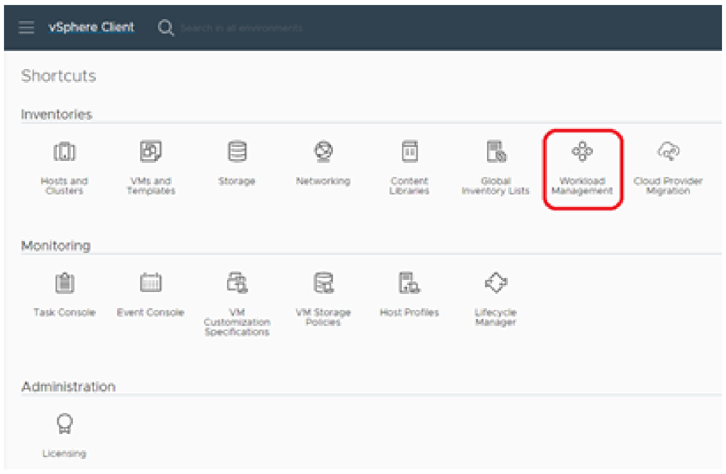
The goal of this portion of the deployment guide is to provide a blueprint for deploying a Tanzu Kubernetes cluster with GPU nodes. VMware vSphere with Tanzu directly integrates with vSphere, which provides a simplified orchestration solution. Tanzu is declarative, so creating and interacting with a GPU-enabled cluster often requires fewer steps than upstream Kubernetes, and it can be done on-demand. Tanzu lets you create and operate Tanzu Kubernetes clusters natively in vSphere. These instructions are not designed to be leveraged for a production environment, rather to provide a bare minimum setup to demonstrate the abilities of Run:ai in conjunction with VMware vSphere with Tanzu.

The instructions assume you have vSphere installed and you have administrative access to the infrastructure. The first step will be to ensure you can connect to vCenter to begin managing your infrastructure.

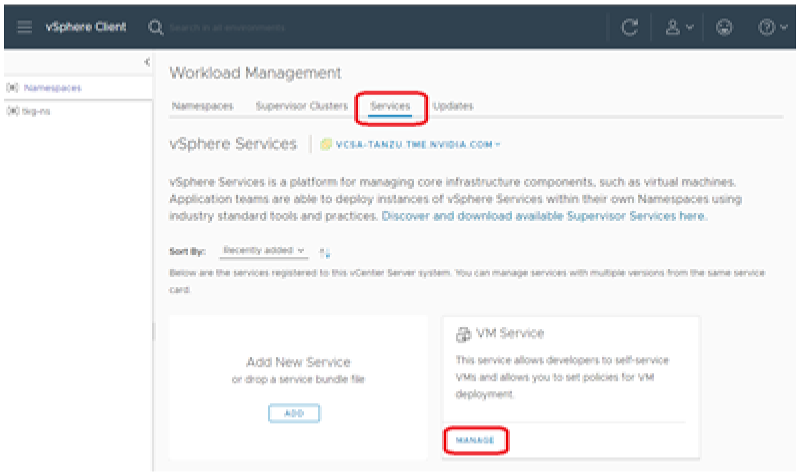
Create a VM Class with GPUs

To leverage a Tanzu Kubernetes cluster with GPUs, you will need to create a virtual machine class. This virtual machine class specifies the underlying compute resources both CPU and GPU that will be dedicated to your Kubernetes worker. VMware vSphere with Tanzu provides default classes for compute or you can create your own VM classes as described below.

To create your own VM classes for your GPU nodes, first login to vCenter. Navigate to Shortcuts and then to Workload Management.



On the Services tab of Workload Management, select Manage under the VM Service tile.

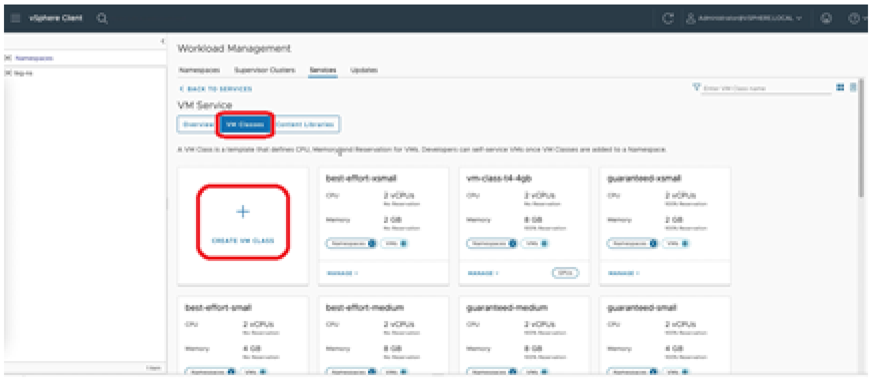


vSphere Services is a platform for managing core infrastructure components, such as virtual machines. Application teams are able to deploy instances of vSphere Services within their own Namespaces using industry standard tools and practices. Discover and download available Supervisor Services here.

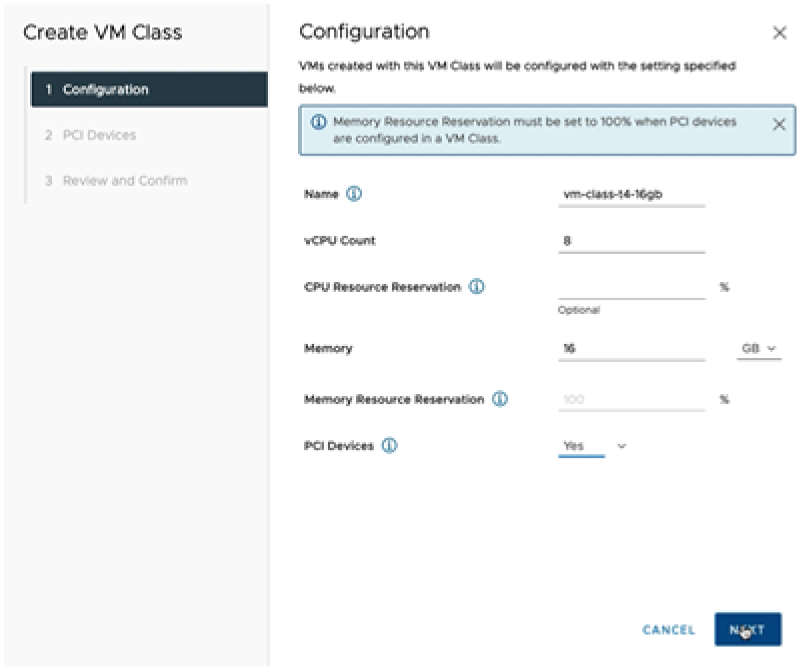
Below are the services registered to this vCenter Server system. You can manage services with multiple versions from the same service card.

This service allows developers to self-service VMs and allows you to set policies for VM deployment.

Select VM Classes and then select the tile to Create VM Class.



Provide the configuration for the VM class. It is recommended to name the VM class relative to the resources on the node itself such as GPU type or you can leverage your organizations naming convention to name and identify the nodes. Provide a vCPU count, Memory allotment and ensure that you select Yes for the PCI devices from the drop down.


Create VM Class

1 Configuration
2 PCI Devices
3 Review and Confirm

Configuration

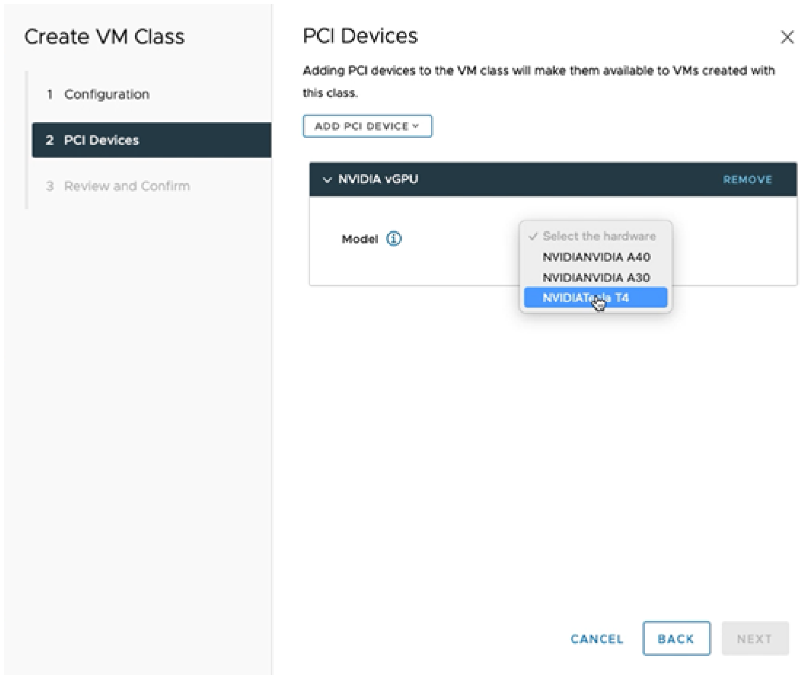
VMs created with this VM Class will be configured with the setting specified below.

Memory Resource Reservation must be set to 100% when PCI devices are configured in a VM Class.

Name	vm-class-t4-16gb
vCPU Count	8
CPU Resource Reservation	% Optional
Memory	16 GB
Memory Resource Reservation	100 %
PCI Devices	Yes

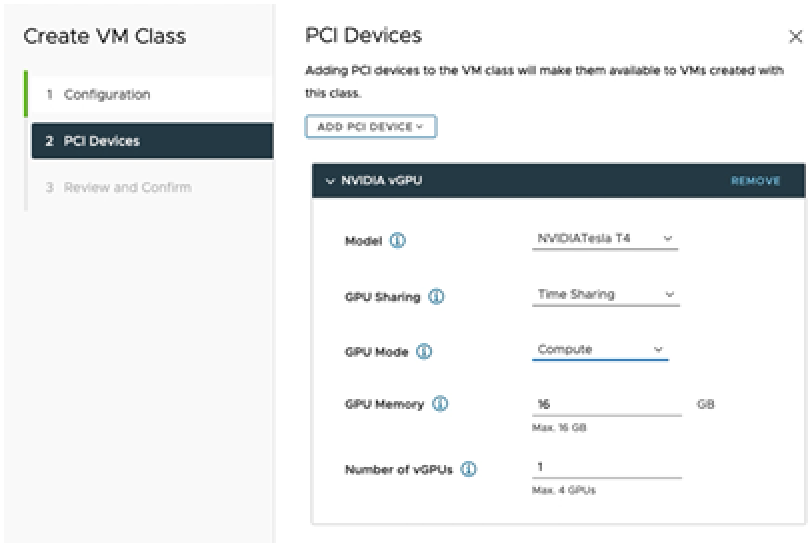
CANCEL NEXT

Next on the PCI devices tab, select NVIDIA vGPU from the dropdown and then select the GPU model that you want to add to your VM class.



Adding PCI devices to the VM class will make them available to VMs created with this class.

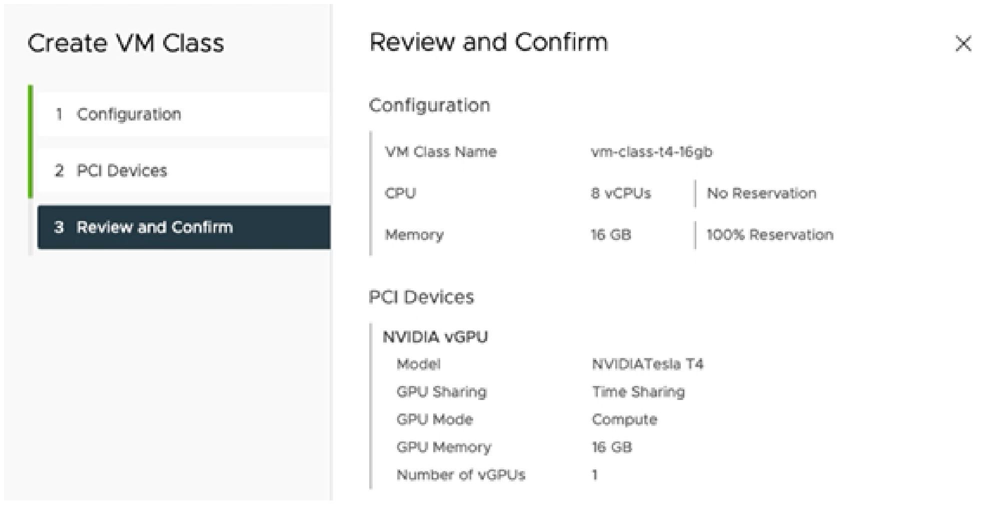
After selecting the GPU model, additional details will need to be provided such as the GPU sharing mode (Time Sharing), GPU Mode (Compute), GPU Memory (typically the maximum memory is allotted) in addition to the total number of vGPUs per node.



Adding PCI devices to the VM class will make them available to VMs created with this class.

Model: NVIDIATesla T4
GPU Sharing: Time Sharing
GPU Mode: Compute
GPU Memory: 16 GB
Max: 16 GB
Number of vGPUs: 1
Max: 4 GPUs

On the final tab, review the details of the VM class and click on Finish to finalize the creation of the VM class.



Create VM Class

1 Configuration
2 PCI Devices
3 Review and Confirm

Review and Confirm

Configuration

VM Class Name	vm-class-t4-16gb	
CPU	8 vCPUs	No Reservation
Memory	16 GB	100% Reservation

PCI Devices

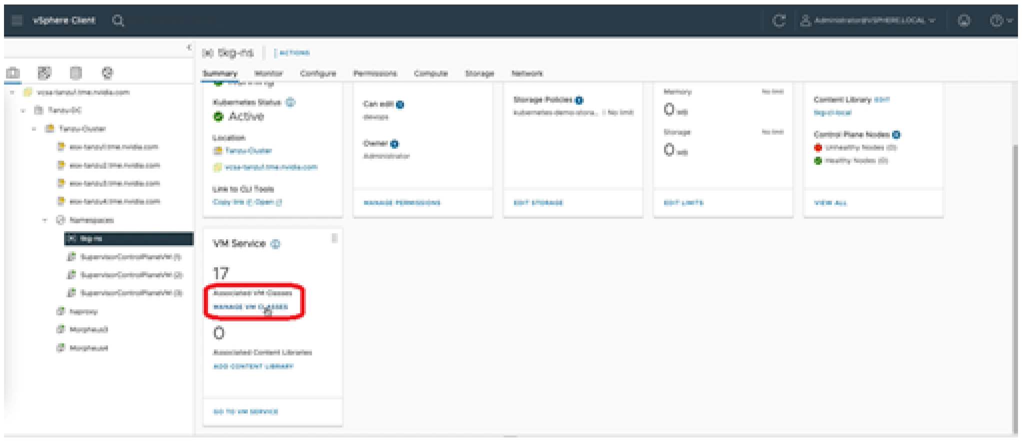
NVIDIA vGPU

Model	NVIDIATesla T4
GPU Sharing	Time Sharing
GPU Mode	Compute
GPU Memory	16 GB
Number of vGPUs	1

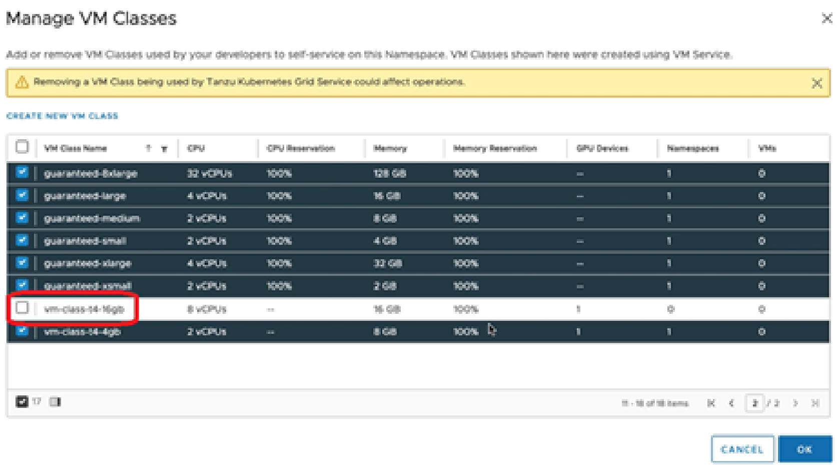
Assign the GPU accelerated VM Class to the Supervisor Namespace

After creating a GPU accelerated VM class it will need to be associated with the Supervisor Namespace. VM classes can reside in one or more namespaces within a Supervisor Cluster. Additionally Supervisor clusters can include multiple different types of VM classes.

Within vCenter, navigate to inventory and expand your Tanzu cluster and associated namespace. Select the appropriate namespace and click on Add VM Class or Manage VM Classes.



Upon entering Add VM Class or Manage CM Classes, you can select the check box next to the VM class that was created in the prior steps as well as any other relevant VM classes to make it available within the namespace.



Manage VM Classes

Add or remove VM Classes used by your developers to self-service on this Namespace. VM Classes shown here were created using VM Service.

Removing a VM Class being used by Tanzu Kubernetes Grid Service could affect operations.

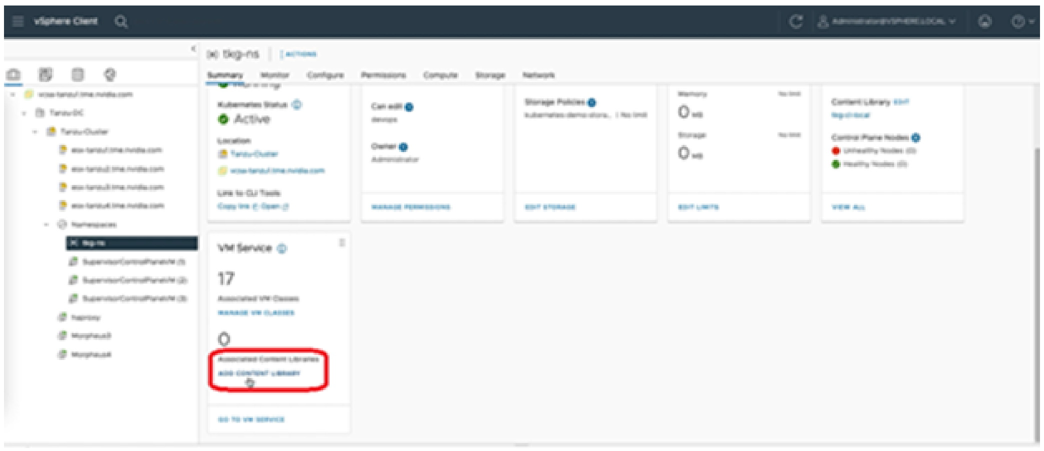
CREATE NEW VM CLASS

	VM Class Name	CPU	CPU Reservation	Memory	Memory Reservation	GPU Devices	Namespaces	VMs
✓	guaranteed-8xlarge	32 vCPUs	100%	128 GB	100%	--	1	0
✓	guaranteed-large	4 vCPUs	100%	16 GB	100%	--	1	0
✓	guaranteed-medium	2 vCPUs	100%	8 GB	100%	--	1	0
✓	guaranteed-small	2 vCPUs	100%	4 GB	100%	--	1	0
✓	guaranteed-xlarge	4 vCPUs	100%	32 GB	100%	--	1	0
✓	guaranteed-xsmall	2 vCPUs	100%	2 GB	100%	--	1	0
	vm-class-t4-16gb	8 vCPUs	--	16 GB	100%	1	0	0
✓	vm-class-t4-4gb	2 vCPUs	--	8 GB	100%	1	1	0

17

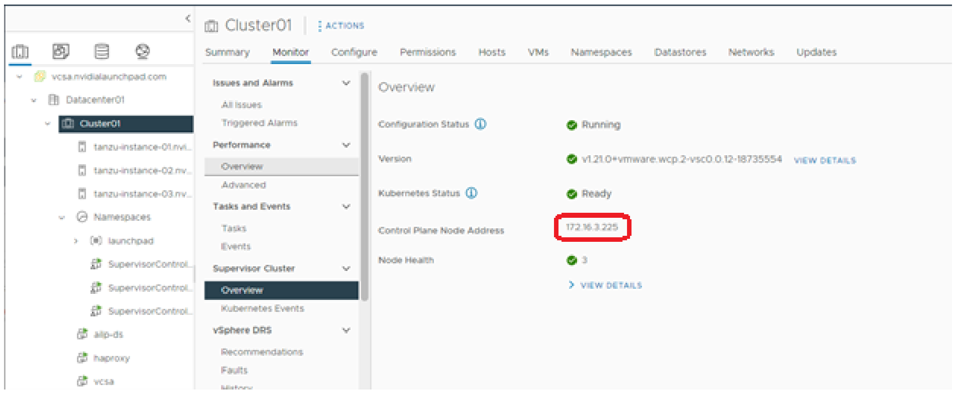
11 - 18 of 18 items 2 / 2

Validate that the Content Library is associated with the Supervisor Namespace by clicking on the Add Content Library button in the VM Service card. Ensure that the VM template is present in the Subscribed Content Library so it can be used by NVIDIA AI Enterprise.



Connect to the Supervisor Cluster

Identify the IP address of the control plane node (<KUBERNETES-CONTROL-PLANE-IP-ADDRESS>) and leverage the IP address to connect to the Supervisor Cluster VM in vCenter.



Overview

Configuration Status: Running

Version: v1.21.0+vmware.wcp.2-vsc0.0.12-18735554

Kubernetes Status: Ready

Control Plane Node Address: 172.16.3.225

Node Health: 3

Use the following command to connect to the Supervisor Cluster VM:

- **`kubectl vsphere login --server=<KUBERNETES-CONTROL-PLANE-IP-ADDRESS> --vsphere-username administrator@vsphere.local --insecure-skip-tls-verify --tanzu-kubernetes-cluster-namespace <namespace>`**

Run the following commands on the CLI to ensure you are connected and submitting commands to the correct cluster:

- **`kubectl config get-contexts`**
- **`kubectl config use-context <context name>`**

Validate your connection to the cluster by running the following commands to view the outputs of the nodes that are in the cluster in addition to all of the running pods.

- **`kubectl get nodes`**
- **`kubectl get pods -A`**

Create a GPU Accelerated Tanzu Cluster

The next step to enabling the groundwork for Run:ai is to provision the GPU accelerated TKG cluster. To ensure that your VM class created in the previous steps is available, run the following kubectl command:

- **`kubectl get virtualmachineclasses`**

To validate that the virtual machine class is configured with GPUs, you can describe the VM class to gather more information.

- **`kubectl describe virtualmachineclass <vmclass-name>`**

The final step for creating the cluster will be to use a text editor to create a yaml file that defines the cluster and includes the VM classes with the GPUs we intend to leverage. An example yaml file below (tanzucluster.yaml) is provided below with some default parameters.

```
apiVersion: run.tanzu.vmware.com/v1alpha2
kind: TanzuKubernetesCluster
metadata:
 name: tkg-cluster
 namespace: tkg-runai
spec:
  distribution:
    fullVersion: 1.20.8+vmware.1-tkg.2
  settings:
    network:
      cni:
        name: antrea
      pods:
       cidrBlocks:
         - 192.0.2.0/16
     serviceDomain: local
      services:
        cidrBlocks:
          - 198.51.100.0/12
    storage:
      defaultClass: runai-kubernetes
  topology:
    controlPlane:
      replicas: 1
      storageClass: runai-kubernetes
      vmClass: guaranteed-medium
    nodePools:
      -
        name: nodepool-t4
        replicas: 2
        storageClass: runai-kubernetes
        vmClass: vm-class-t4-16gb
        volumes:
          -
            capacity:
              storage: 100Gi
            mountPath: /var/lib/containerd
            name: containerd
```

Once the yaml file is edited, it can be applied to deploy the cluster.
- **`kubectl apply -f tanzucluster.yaml`**

Check the status of the cluster and wait until the cluster shows ready.
- **`kubectl get tkc`**

Once the cluster shows as Ready, you can use the following command to connect to the GPU accelerated cluster to deploy NVIDIA's GPU Operator as well as proceed with the installation of Run:ai.

- `kubectl vsphere login --server=<KUBERNETES-CONTROL-PLANE-IP-ADDRESS> --vsphere-username administrator@vsphere.local --insecure-skip-tls-verify --tanzu-kubernetes-cluster-name tkg-cluster --tanzu-kubernetes-cluster-namespace tkg-runai`

NVIDIA GPU Operator

Install the NVIDIA GPU Operator

Run:ai requires the NVIDIA GPU Operator to be installed on the Kubernetes cluster in order to expose the GPUs to the Run:ai platform. The NVIDIA GPU Operator uses the operator framework within Kubernetes to automate the management of all NVIDIA software components needed to provision GPUs. These components include the NVIDIA drivers (to enable CUDA), Kubernetes device plugin for GPUs, the NVIDIA Container Runtime, Node Feature discovery for automatic node labeling and DCGM based monitoring.

The recommended installation steps for installing the NVIDIA GPU Operator are included below and the full installation instructions including potential customization options are provided by NVIDIA at the following link:
https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/getting-started.html

Add the NVIDIA helm repository:

- `helm repo add nvidia https://helm.ngc.nvidia.com/nvidia && helm repo update`

Install the GPU Operator

- `helm install --wait --generate-name -n gpu-operator --create-namespace nvidia/gpu-operator`

Deploy the Run:ai stack

Install the Run:ai cluster

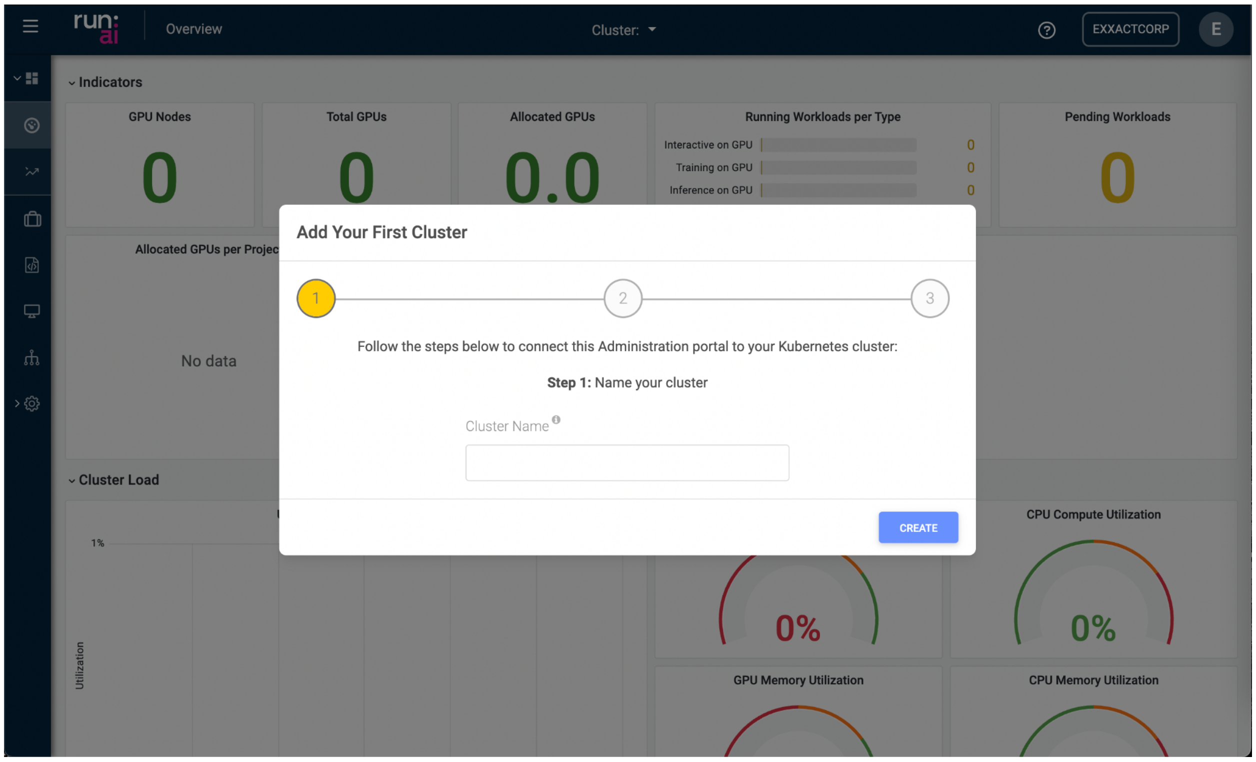
The installation instructions for Run:ai below are for the Saas version however the cluster installation will be similar for both self hosted and Saas installations. For the most up to date

installation instructions please refer to the following link:
https://docs.run.ai/admin/runai-setup/cluster-setup/cluster-install/

Once all of the prerequisites have been completed including the provisioning of a vSphere Tanzu cluster and installation of the NVIDIA GPU Operator, proceed to the installation of Run:ai.

Begin the installation of Run:ai by navigating to your Saas tenant URL. Typically the format for a tenant will be **https://<customername>.run.ai/.** The credentials to authenticate to this page will be provided by Run:ai customer support. Upon first login, if no other clusters are configured you will be prompted to install a cluster. If a cluster is already configured, you can navigate to clusters on the left side hamburger menu to add an additional cluster.

The first step is to name the cluster, the cluster name is nonfunctional, its primary use is for identification purposes. After you have named the cluster, click on the blue "**Create**" button to proceed.


Add Your First Cluster

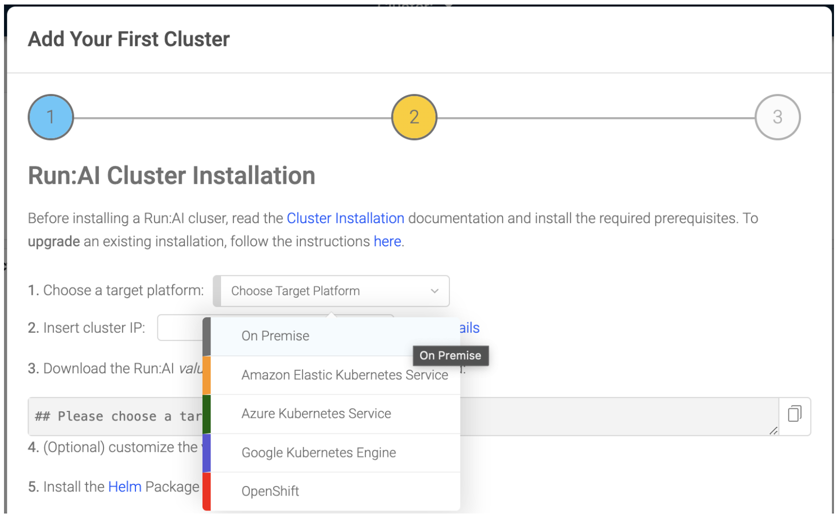
Follow the steps below to connect this Administration portal to your Kubernetes cluster:

Step 1: Name your cluster

Cluster Name

On the "Add Your First Cluster" page, there are a few parameters that will need to be filled in. The first option will be to choose the target platform for the installation. Run:ai supports various container orchestrators from cloud based Kubernetes services to OpenShift and Rancher. For this particular installation we can choose On Premises from the dropdown menu. The next item that needs to be selected is the cluster IP. We can use any of the node IPs of the cluster here for testing purposes and for production deployments Run:ai can integrate with existing ingress controllers or external load balancers to grant access. Upon filling in the cluster IP, the field below will populate with a curl command to pull the values file used for the Run:ai installation.

This value file contains the configurations needed to link the Tanzu cluster to the Run:ai Saas tenant. If you need to make any modifications to the default configuration, the options for modification are documented here:
https://docs.run.ai/admin/runai-setup/cluster-setup/customize-cluster-install/?h=customize


Add Your First Cluster

1 2 3

Run:AI Cluster Installation

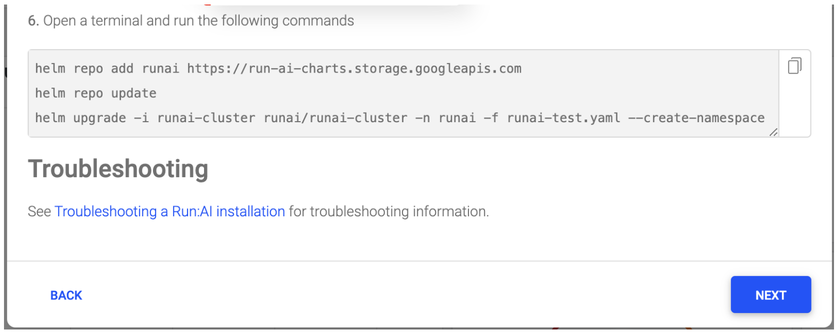
Before installing a Run:AI cluser, read the Cluster Installation documentation and install the required prerequisites. To **upgrade** an existing installation, follow the instructions here.

1. Choose a target platform: Choose Target Platform
2. Insert cluster IP:
3. Download the Run:AI *valu*

```
## Please choose a tar
```

4. (Optional) customize the
5. Install the Helm Package

From there the installation proceeds with 3 helm commands. First add the Run:ai repo. Next update the helm repo and then finally perform a helm upgrade command similar to the one below. This deploys the full Run:ai stack on the cluster in the runai namespace.


6. Open a terminal and run the following commands

```
helm repo add runai https://run-ai-charts.storage.googleapis.com
helm repo update
helm upgrade -i runai-cluster runai/runai-cluster -n runai -f runai-test.yaml --create-namespace
```

Troubleshooting

See Troubleshooting a Run:AI installation for troubleshooting information.

Validate the installation

To validate that the install was successful, run the following command to validate that all of the pods in the runai namespace are running.

- **`kubectl get pods -n runai`**

Once the installation is complete, you can return to the overview page of the Saas tenant where the GPU and cluster details should begin to appear after a few minutes. For additional troubleshooting methods please see the troubleshooting section of the documentation:
https://docs.run.ai/admin/troubleshooting/troubleshooting/?h=tro

Configure the Run:ai installation

Modify the kube-apiserver to Enable Researcher Authentication

For full Run:ai product functionality, it's required to have the Kubernetes API server authenticate via Run:ai. This requires adding flags to the Kubernetes API Server. Modifying the API Server configuration differs between Kubernetes distributions; the most up to date instructions for modifying the api-server are included for multiple distributions at the following link: https://docs.run.ai/admin/runai-setup/authentication/researcher-authentication/?h=kube+apiserver#mandatory-kubernetes-configuration

Install the Researcher Command Line Interface (CLI)

For any users who choose to use the CLI to interact with Run:ai, full instructions for enabling CLI access including installation of the runai binary are provided at the following link: https://docs.run.ai/admin/researcher-setup/cli-install/?h=researcher+cli

Configure Projects

Projects are the basic building block to drive the scheduling and orchestration concepts of Run:ai. To streamline resource allocation and prioritize work, Run:ai introduces the concept of Projects. Projects are the tool to implement resource allocation policies as well as create segregation between different initiatives. A project in most cases represents a team, an individual, or an initiative that shares resources or has a specific resources budget (quota). A Researcher submitting a Job needs to associate a Project name with the request. The Run:ai scheduler will compare the request against the current allocations and the Project and determine whether the workload can be allocated resources or whether it should remain in the queue for future allocation. For more information on projects and how they can be configured, see the following link: https://docs.run.ai/admin/admin-ui-setup/project-setup/?h=projects

Getting Started

Run:ai provides a multitude of guides for enabling data scientists and researchers to leverage GPUs. These resources are available at the following links:

- Quickstart Guides: https://docs.run.ai/Researcher/Walkthroughs/quickstart-overview/?h=quickstart
- Use Cases: https://docs.run.ai/Researcher/use-cases/?h=use+case#use-cases
- Integrations: https://docs.run.ai/admin/integration/kubeflow/