Orion Documentation

genesis4.0.1-orion4.0.1

Juno Innovations

Table of contents

1. Transforming High-Performance Workflows

1.1 The Future of Digital Innovation

Orion is Juno Innovations' groundbreaking Unified Compute Plane revolutionizing how enterprises transform their biggest expense into their greatest competitive advantage. As the industry's first Supervisor Layer, Orion orchestrates everything from bare metal to Kubernetes, delivering unprecedented resource optimization across multiple high-performance computing sectors including media & entertainment, life sciences, aerospace, defense, financial services, and government.

With deployment options spanning on-premises, cloud, and hybrid environments, Orion enables organizations of any size to achieve **92% GPU** utilization, **87% CPU** utilization, and **85% RAM** utilization—compared to industry standards of 35-65% — while reducing deployment time from 72+ hours to just **15 minutes**.

1.1.1 Why Industry Leaders Choose Orion

- **Unified Compute Plane Architecture** — The Supervisor Layer that orchestrates all infrastructure types without vendor lock-in. Deploy on-premises, in the cloud, or create hybrid environments with burst capabilities. Your infrastructure, your control.
- **Proven Resource Optimization** — Achieve **92% GPU**, **87% CPU**, and **85% RAM** utilization in production environments. Transform resource waste into competitive advantage with measurable **56% cost reduction**.
- **Container-Native Foundation** — Built from the ground up for modern workloads, not retrofitted from legacy architecture. Deploy workloads in **60 seconds** instead of hours, enabling true distributed teams without performance compromise.
- **Production-Proven Reliability** — Validated in demanding environments achieving **99.99998% uptime**. Self-healing architecture with fault tolerance ensures continuous operation protecting critical workflows.
- **Universal Infrastructure Freedom** — Works with any infrastructure— **bare metal, cloud, hybrid, edge** —enabling optimal deployment models without architectural constraints. True multi-cloud freedom through intelligent orchestration.
- **Dynamic Resource Allocation** — Real-time resource optimization that responds to workload demands instantly. Scale up for peak performance, scale down for cost efficiency, all automatically.
- **Comprehensive API Integration** — Every component exposes RESTful API endpoints with complete OpenAPI documentation, enabling seamless pipeline integration and custom workflow development.

1.1.2 Compute Orchestration Focus

Orion's Unified Compute Plane specializes in optimizing compute resources while customers maintain complete control over their data and storage:

1.1.3 Orion Platform Use Cases

The Orion platform addresses critical challenges across multiple industries, transforming fixed infrastructure costs into dynamic competitive advantages:

Media Production Compute Optimization

- **Render Farm Orchestration**: Optimize GPU, CPU, and RAM utilization across hybrid infrastructure
- **Workload Resource Management**: 92% GPU utilization for creative workloads
- **Collaborative Computing**: Enable remote teams with optimized compute access
- **Transcoding Acceleration**: Intelligent resource allocation for media processing

Enterprise Compute Infrastructure

- **Multi-Cloud Resource Optimization**: Unified compute management across AWS, on-premises, and hybrid
- **Container Orchestration**: Kubernetes-based compute deployment with 87% CPU utilization
- **DevOps Compute Automation**: Resource optimization for CI/CD pipelines
- **Cost Optimization**: 56% reduction in compute infrastructure costs

Life Sciences & Pharmaceuticals

- **Drug Discovery Acceleration** — Reduce computational costs by 56% while improving research throughput by 200%
- **Genomic Analysis Optimization** — Dynamic resource allocation for batch processing with peak utilization when needed
- **Regulatory Compliance** — Built-in audit trails and data sovereignty for FDA and international requirements
- **Research Collaboration** — Secure multi-site research environments with controlled data access

Aerospace & Defense

- **Mission-Critical Simulations** — Air-gapped infrastructure delivering 92% resource utilization for classified workloads
- **Computational Fluid Dynamics** — Optimize aircraft design cycles with intelligent resource allocation
- **Secure Collaboration** — Multi-level security clearance environments with complete data isolation
- **Legacy System Integration** — Bridge existing defense infrastructure with modern cloud-native capabilities

Financial Services

- **High-Frequency Trading** — Guaranteed resource availability with real-time trading model performance
- **Risk Analysis Acceleration** — Dynamic scaling for monte carlo simulations and stress testing
- **Regulatory Reporting** — Automated compliance frameworks with comprehensive audit capabilities
- **Hybrid Cloud Strategy** — Balance performance requirements with cost optimization

Government & Public Sector

- **Budget Optimization** — 56% infrastructure cost reduction while maintaining security and compliance
- **Multi-Agency Collaboration** — Secure resource sharing across departments and jurisdictions
- **Citizen Services** — Scalable platforms for public-facing applications with predictable costs
- **Emergency Response** — Rapid deployment capabilities for crisis management and disaster recovery

Oil & Gas

- **Seismic Data Processing** — Process exploration data 50% faster with dynamic resource allocation
- **Reservoir Modeling** — Optimize simulation workloads across hybrid infrastructure
- **Field Operations** — Edge computing capabilities for remote monitoring and analysis
- **Environmental Compliance** — Automated reporting and monitoring for regulatory requirements

1.1.4 Infrastructure Deployment Scenarios

Multi-Cloud Orchestration

- **Hybrid Deployments** — Seamlessly span AWS, CoreWeave, and on-premises infrastructure (Azure, GCP, private data centers are untested but should work)
- **Workload Portability** — Move applications between environments without architectural changes
- **Cost Optimization** — Intelligent workload placement based on performance requirements and costs

- **Disaster Recovery** — Automated failover and recovery across multiple geographic regions

Enterprise Integration

- **Legacy Modernization** — Bridge existing infrastructure with cloud-native applications
- **Identity Management** — Integrate with enterprise authentication systems (LDAP, Active Directory, SAML)
- **Compliance Frameworks** — Meet industry regulations with automated governance and audit trails
- **DevOps Acceleration** — GitOps-driven CI/CD pipelines with container orchestration

High-Performance Computing

- **Research Computing** — Scientific modeling and simulation with optimal resource utilization
- **AI/ML Workloads** — Model training and inference with dynamic GPU allocation
- **Big Data Analytics** — Distributed processing with automatic scaling based on data volume
- **Real-Time Processing** — Stream processing and edge computing for time-sensitive applications

1.1.5 Complete Integrated Platform

Orion integrates seamlessly with Juno Innovations' ecosystem:

- **Unified Compute Plane** — The Supervisor Layer for all your infrastructure
- **Terra Marketplace** — One-click deployment for complex workloads and applications
- **Professional Services** — Migration assistance, training, and custom implementations

1.1.6 Ready for the Future

Developed by a team with proven track record in infrastructure transformation, Orion represents the post-hypervisor evolution of enterprise computing. Join organizations already transforming their workflows with the industry's first Unified Compute Plane—where infrastructure amplifies innovation instead of constraining it.

Beyond the hypervisor. Above the stack.

Contact our sales team today to schedule a demonstration or discuss your specific requirements.

2. Installation

2.1 Quick Start

Installing Orion is meant to be as easy as possible. We provide a "OneClick" installer which will allow you to install Orion as fast as possible for both local testing and enterprise deployments.

2.1.1 Pre-requisites (Test/Dev)

1. Our deployment can run on any relatively modern Linux distribution with systemd support. You can find our officially supported Linux distribution list here

> **Details**
>
> In case you'd like to validate the specifics, check out:
>
> - k3s System Requirements
> - Ensure `modprobe wireguard` runs correctly - this is used to secure the cross-node traffic and comes by default on modern Linux kernels
> - Ensure `cgroupv2` support (already there on all modern Linux releases)

2. The DNS name for your Orion installation points at the IP of the first node in the cluster.
   - For community contributors or small test deployments, this can be just the IP.
   - Large production deployments, we recommend you make this a CNAME record, such as `orion.yourdomain.com`.
3. As a bare minimum, you will need a single node meeting the "service node" requirements listed in our System Requirements
4. (Optional) When performing an air-gapped installation, ensure that:
   - you meet all prerequisites listed in our air-gapped installation guide
   - you have downloaded the latest Installer Archive tar.gz archive and placed it on the first node of your Orion cluster.
   - you will be prompted for the filepath to the installer
   - once that node is up, you'll be able to join the remaining ones using the Genesis web UI.

2.1.2 Installation

Option 1: OneClick - bootstrapping the cluster (Quickest)

1. Connect to one of the servers you allocated for the service/control-plane node via SSH.

```
ssh user@your-server-ip
```

2. Run our helper script to set up the environment and download the latest OneClick Installer.

```
curl -sL "$(curl -s https://api.github.com/repos/juno-fx/Juno-Bootstrap/releases/latest | grep browser_download_url | grep orion-install-helper | cut -d '"' -f 4)" | bash -
```

3. Fill out the form and select "On Prem K3s" as the Deployment Target and follow the prompts.

> **Production Deployments**
>
> While this is the quickest way to get started, for production deployments we recommend you configure the created cluster using our System Requirements as a guide. The generated cluster is a great starting point and already handles quite a few of the labeling steps for you.

Option 2: Helm Chart Install

If you prefer to install Orion via a helm chart install, you can do this by following the ReadMe documentation via our Juno-Bootstrap repo

2.1.3 Joining more nodes to the cluster

Once your cluster is up, you can easily expand it through the web UI. Check out the documentation for it here

2.1.4 Uninstall

1. Connect to your server via SSH.

```
ssh user@your-server-ip
```

2. To uninstall k3s (including Orion), run the following command on the server where it was installed:

```
# control nodes
sudo k3s-uninstall.sh

# worker nodes
sudo k3s-agent-uninstall.sh
```

2.1.5 Next Steps

Now that the cluster is running, we recommend you explore the following steps.

1. Authentication Guide
2. License Management Guide
3. Expand your Orion Cluster
4. Explore Genesis - the management platform for Orion
5. Explore Hubble - the namespace portal for deployed Orion projects
6. Explore Terra - the plugin driven deployment platform for Orion

2.1.6 Additional Deployment Options

The OneClick installer supports multiple deployment targets:

- On-Premises using k3s - a lightweight Kubernetes distribution perfect for small clusters and edge deployments.
- AWS EKS (Coming Soon) - EKS deployment target for enterprise AWS customers.
- CoreWeave (Coming Soon) - CoreWeave deployment target for enterprise CoreWeave customers.

We've successfully tested Orion on multiple Kubernetes distributions:

- **Local Development**: Development teams use kind clusters to run Juno.
- **Cloud**: Our internal production Juno Cloud runs on Amazon EKS
- **On-Premises**: Test environments operate on k3s with hybrid architectures

> **Multi-Architecture Support**
>
> All deployments support both Arm64 and x86_64 architectures, allowing you to choose the best hardware for your needs.

2.2 System Requirements

This guide will walk you through the necessary steps to prepare your Kubernetes cluster for a production-ready Orion environment. Proper cluster preparation ensures optimal performance and reliability.

2.2.1 Prerequisites

Hardware Requirements

The following table outlines the minimum hardware requirements for a standard Orion installation:

Server Role	Count	CPU	RAM	Purpose
`Service`	1	4 Core	16GB	Runs core services required by Orion
`Workstation`	1	4 Core	16GB	Handles workload tasks
`Headless`	1	4 Core	16GB	Processes headless workloads

Minimum Requirements

These are minimum requirements. For production environments or larger workloads, we recommend scaling up resources accordingly. We highly recommend having at least 2 nodes for the `Service` role to ensure high availability.

Multi-Role Nodes

For small test deployments or proof-of-concept setups, you can run all roles on a single server. However, this is not recommended for production use.

Kubernetes Requirements

Orion is agnostic to the specific Kubernetes distribution it runs on.

We release modeling our cadence on standard Cloud Native Computing Foundation practices, testing against the last two minor versions of Kubernetes and their latest bugfix/patch releases. We attempt, but not guarantee compatibility with 3 minor versions of the Kubernetes API.

If Orion makes use of newest Kubernetes features ahead of this window, this will be explicitly stated in the Changelog.

You can see the Kubernetes version the final QA for a release was performed on in our Technical Changelog.

2.2.2 JWKS endpoint

Modern Kubernetes clusters ship with a JWKS endpoint.

In most clusters, it is in-cluster. On some distributions, such as EKS, it is outside the cluster, but still discoverable and reachable by default. It is a hard requirment for Orion services to be able to reach this service.

Unless your setup is highly and intentionally customized, you can expect this to work out of the box.

2.2.3 Services

The following services are hard requirements for Orion to function properly:

Service	Required Version	Notes
Nginx Ingress Controller	>=4.12.1 (chart)	We recommend the helm installation method over raw manifests
ArgoCD	> 3.0.0	We upgrade to latest stable Argo quarterly in our QA cycle. That is what official releases test against

2.2.4 Server Role Configuration

Juno products deploy components to servers based on assigned roles. These roles are implemented through Kubernetes labels and taints.

Core Roles

Role	Label	Purpose
Service	`juno-innovations.com/service: true`	Runs infrastructure services (Genesis, Kuiper, Terra, Titan, etc.)
Workstation	`juno-innovations.com/workstation: true`	Runs interactive workload environments
Headless	`juno-innovations.com/headless: true`	Runs generic workloads

> ⚠ **Multi-Role Nodes**
>
> Servers can have multiple roles if needed. For example, in smaller deployments, a server might handle both interactive and headless workloads.

2.2.5 Node Configuration

Cloud Provider Configuration

For cloud environments (AWS, GCP, Azure), refer to your provider's documentation for implementing node groups with the appropriate labels and taints:

- AWS EKS Node Groups
- GCP GKE Node Pools
- Azure AKS Node Pools

On-Premises Configuration

Follow these steps to label and taint your on-premises Kubernetes nodes:

LABELING NODES

> **K8s Users**
>
> If you used our OneClick installer, your nodes may already be labeled correctly. You can verify this by opening the Genesis panel and navigating to the Network section.
>
> You can also use our integrated node provisioning feature to add and label nodes automatically. You can follow the guide in the Expand your Orion cluster section.

Apply the appropriate labels to designate node roles:

```
# Label service nodes
kubectl label nodes <node-name> juno-innovations.com/service=true

# Label workload nodes
kubectl label nodes <node-name> juno-innovations.com/workstation=true

# Label headless nodes
kubectl label nodes <node-name> juno-innovations.com/headless=true
```

2.2.6 Verifying Your Configuration

To verify that your nodes are correctly labeled:

```
# List all nodes with their labels
kubectl get nodes --show-labels | grep juno
```

2.2.7 On-Premise Recommendations - LoadBalancer and DNS records

LoadBalancer

When deploying the Ingress, which acts as the entry point for all the traffic, Kubernetes needs to assign it an IP address. To do that, it asks a "LoadBalancer". How this is provided can be very specific - for example all cloud providers implement their own load balancers. Kubernetes simply asks this component what it should use - however it relies on the Load Balancer to handle all the work in the background.

While there's many options out there, we recommend two:

- K3s built-in Load Balancer, ServiceLB. This is what our pre-packaged deployment uses as the default. It is a great choice for:
- small-to-medium deployments
- cases where you'd rather avoid any extra complexity
- cases where you can perform manual failover

When using our pre-provided installation methods, this will be part of your setup by default. All you need to do on your end is to create appropriate DNS records.

To see how you would handle upgrading your underlying OS with zero-downtime, refer to our Failover & Node Maintenance Guide

- For bigger deployments, we recommend Cilium instead. It brings additional complexity, however we believe larger environments will benefit from its automated failover, as well as observability capabilities. It provides:
- Automatic L2 failover. That means if the node goes down and it was holding the IP handling the traffic, it will be automatically "taken over" by a healthy node.
- Enhanced visibility, rich set of network-level metrics.
- Rich FeatureSet - Cilium is in many ways the gold standard and is often used as a reference for the rest of the K8s ecosystem.

We only recommend Cilium for large, custom K8s deployments - we don't currently pre-package it in our standard installation methods.

You can configure Cilium in 2 modes:

- BGP load-balancing - the most complex to manage, as well as most performant.

> **Advice**
>
> It's unnecessary for most customers. We recommend evaluating & testing the L2 mode practically first and upgrading only if necessary.

- L2 Announcements - despite the official beta status, this is a very mature and capable configuration, providing a single, failover-capable IP that acts as an entry point to your cluster.

> **Advice**
>
> If automated failover is not required, you can stick with the pre-packaged ServiceLB and manual DNS-based failover.

For configuration details on both, refer to upstream Cilium documentation.

For a deeper dive into the underlying concepts, you can find relevant upstream documentation here:

- Kubernetes networking overview
- Load balancers

DNS records

When you deploy Juno, you will need to set up DNS records pointing at your Ingress Controller. The Ingress Controller acts as a reverse proxy and is the entry point for all traffic to your cluster. You can do this before setting up. The DNS records you'll need to create will be:

- an IP of any of your control plane nodes, when using the default k3s Load Balancer, ServiceLB.
- Failover is manual in this case. It relies on you switching the DNS record to another node's IP. Each node is capable of serving traffic.
- You can also use Round-Robin DNS to provide simple load balancing. If a node fails, you would remove its record to avoid its selection.
- the IP you designated for L2 (or BGP) failover/load balancing when using Cilium or bringing your own Load Balancer of choice.

2.2.8 Troubleshooting

If you encounter issues during cluster preparation:

- Ensure all nodes meet the minimum hardware requirements
- Verify that Kubernetes version is 1.27 or newer
- Check that all required services are properly installed
- Confirm node labels and taints are correctly applied

For further assistance, contact Juno Support.

2.3 Authentication Setup

2.3.1 Overview

The Genesis services serve as the primary access point for all Juno products. Genesis implements authentication through the industry-standard NextAuth.js library, providing secure and reliable authentication services. Currently, Genesis supports **Google** and **AWS Cognito** as authentication providers, with more options coming soon.

This comprehensive guide walks you through the process of setting up authentication for your Genesis deployments.

> **Genesis Propagation**
>
> When using Genesis, it automatically detects your configured authentication services and can propagate these settings to your managed Orion deployments if you choose.

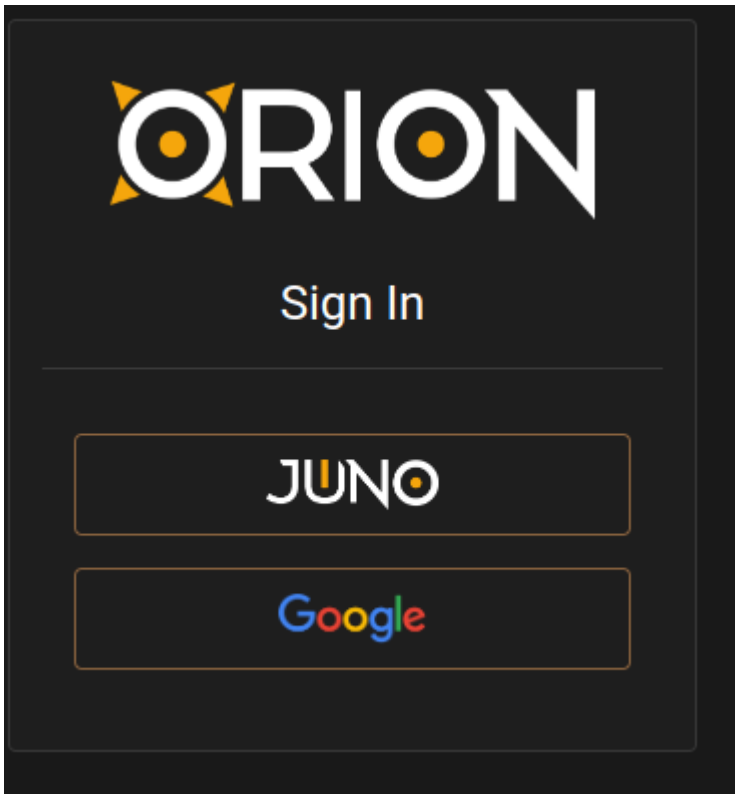
2.3.2 Quick Start

For production deployments, we support **AD/LDAP**, **Google** and **AWS Cognito** authentication as well as many others coming soon. Follow the provider-specific instructions below to get started quickly.

2.3.3 Google Authentication

Follow these steps to set up Google OAuth for your Juno application:

1. Navigate to the Google Cloud Console
2. Create a new project or select an existing one
3. Go to **APIs & Services > Credentials**
4. Click **Create Credentials** and select **OAuth Client ID**
5. Select **Web Application** as the application type
6. Configure the application:
7. Name: [Your Application Name]
8. Authorized JavaScript origins: `https://your-genesis-domain.com`
9. Authorized redirect URIs: `https://your-genesis-domain.com/api/auth/callback/google`
10. Click **Create**
11. Save the generated **Client ID** and **Client Secret** for the environment configuration

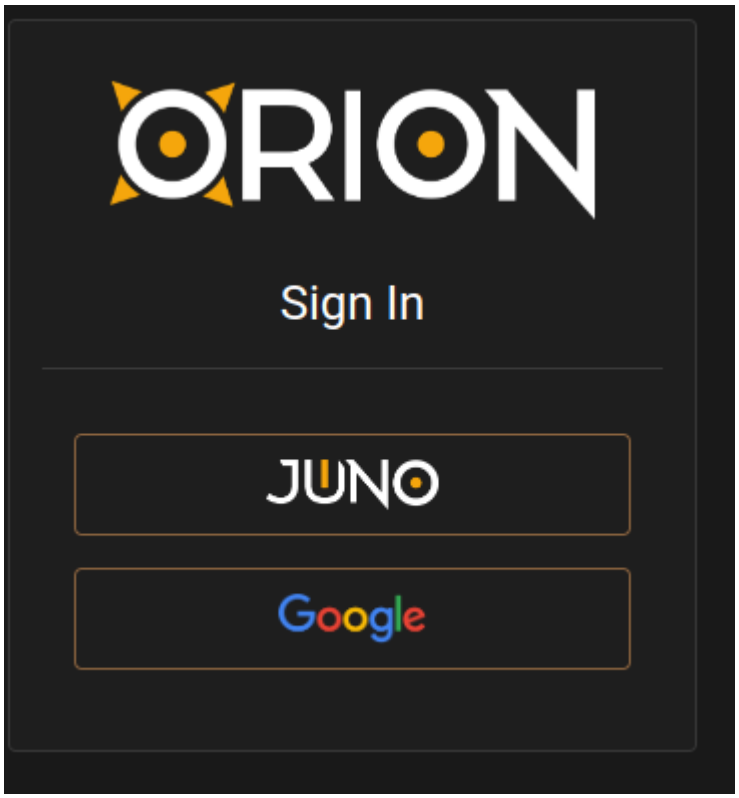


2.3.4 AWS Cognito Authentication

AWS Cognito provides a robust user directory and authentication service. Follow these steps to configure Cognito for your Juno deployment:

1. Log in to the AWS Management Console
2. Navigate to the **Cognito** service
3. Click **Manage User Pools**
4. Create a new user pool or select an existing one
5. Go to the **App Integration** tab
6. Click **Create app client**
7. Configure the app client:
8. App client name: [Your App Client Name]
9. Generate client secret: **Yes** (recommended)
10. Auth Flows Configuration:
    - Enable `ALLOW_REFRESH_TOKEN_AUTH`
    - Enable `ALLOW_USER_PASSWORD_AUTH`
    - Enable `ALLOW_USER_SRP_AUTH`
11. Click **Create app client**
12. In the **App Integration** tab, find the **Hosted UI** section and click **Edit**
13. Configure the Hosted UI:
    - Callback URL(s): `https://your-genesis-domain.com`
    - Sign out URL(s): `https://your-genesis-domain.com/api/auth/callback/cognito`
    - OAuth grant type: Select both **Authorization code grant** and **Implicit grant**
    - OpenID Connect Scopes: Select `openid`, `email`, `phone`, and `profile`
14. Click **Save changes**
15. Note the following values for environment configuration:
    - User Pool ID (from the user pool's General settings)
    - App client ID (from the App clients section)
    - App client secret (from the App clients section)

> **Cognito Button**
>
> Juno internally uses AWS Cognito, so the Cognito login button will display the Juno logo in the authentication interface.



2.3.5 Basic Authentication

We also provide a basic authentication setup, which does not require you to have any enterprise authentication available. It is built for local development & testing of the Juno.

It is intended for local testing and insecure by design.

> **Basic Authentication Warning**
>
> Not appropriate for production use.

To use it:

1. Set the `BASIC_AUTH_EMAIL` and `BASIC_AUTH_PASSWORD` environment variables, as in the example below.
2. Make sure the `owner` user email you configure in Juno-Bootstrap matches `BASIC_AUTH_EMAIL`.

That's it! Once you deploy, you will be able to sign-in with the credentials you provided.

2.3.6 AD/LDAP Authentication

To set up LDAP authentication, follow these steps:

1. Ensure you have access to an LDAP server (e.g., Microsoft Active Directory)
2. Configure your LDAP server with the necessary user directory and permissions
3. You must provide a bind user DN and password - a bind user is a standard account able to authenticate against your directory and query user attributes. We recommend it to be a specific service account used only for Juno, with no extra privileges

When using Active Directory, we also support LDAPs.

You can pass in the connection string (`LDAP_URI`) in the format of: `ldaps://my-server:<port>` eg. `ldaps://dc1.ad.example.com:689`
When using your own Certificate Authority you can optionally pass it in, to enable self-signed CAs.

For this to work, take your CA cert in the PEM format and then base64 encode it:

```
cat ca_cert.pem | base64 -w0
```

You can pass the resulting string to the option outlined below.

2.3.7 Environment Configuration

After setting up your authentication provider, update your Juno Cluster Bootstrap configuration file with the appropriate environment variables. The configuration should be added to the `env` section as shown below:

```
# Juno Cluster Bootstrap configuration
config:
  # Other configuration settings...

  # Environment Variables
  env:
    # You must have either Google or AWS Cognito configured.
    # Uncomment one of the following (REQUIRED)
    ### Google OAuth (Uncomment if using)
    # GOOGLE_CLIENT_ID: "your-google-client-id"
    # GOOGLE_CLIENT_SECRET: "your-google-client-secret"
    ### AWS Cognito (Uncomment if using)
    # COGNITO_CLIENT_ID: "your-cognito-app-client-id"
    # COGNITO_CLIENT_SECRET: "your-cognito-app-client-secret"
    # COGNITO_ISSUER: "https://cognito-idp.{region}.amazonaws.com/{userPoolId}"
    ### LDAP (Uncomment if using)
    #  The CA cert is optional.
    #  It is to support LDAPS certs signed with your own CA.
    #  It expects a PEM-formatted CA cert exported as base64 string.
    # LDAP_URI: "ldap://your-ldap-server:389"
    # LDAP_BIND_DN: "cn=admin,dc=example,dc=com"
    # LDAP_BIND_PASSWORD: "password"
    # LDAP_SEARCH_BASE: "dc=example,dc=com"
    # LDAP_EMAIL_ATTRIBUTE: "mail"
    # LDAP_USERNAME_ATTRIBUTE: "samaccountname"
    # LDAPS_CA_CERT_BASE64:
    #   This is the option we provide for local development.
    #   It is insecure and not recommended for production use.
    # BASIC_AUTH_EMAIL: "your-email@example.com"
    # BASIC_AUTH_PASSWORD: "your-password"
    # BASIC_AUTH_EMAIL_2: "another-email@example.com"
    # BASIC_AUTH_PASSWORD_2: "another-password"
```

2.3.8 Troubleshooting

Common Issues

Problem	Possible Solution
"Redirect URI Mismatch" error	Ensure the redirect URI in your provider configuration exactly matches your application's callback URL
Authentication timeout	Check network connectivity between your application and the authentication provider
"Invalid Client" error	Verify that your client ID and secret are correct and properly configured
Users cannot log in	Ensure the user has been added to the user directory and has the correct permissions

2.3.9 Security Best Practices

To maintain a secure authentication system:

1. **Never use basic authentication in production**
2. **Use HTTPS** for all authentication endpoints and redirects
3. **Monitor authentication logs** for suspicious activity

For further assistance, contact Juno Support.

2.4 License Management

2.4.1 Activating Your License

Upon purchasing from Juno, you will receive a license key via email from our team. This key is added to Genesis under the "Licensing" tab after the deployment process to validate your subscription and activate all licensed services. You can also update your license at any time via the Genesis web UI, under the "Licensing" tab following the same process as initial setup.

If you haven't received your license key or need assistance, please contact our sales team at sales@juno-innovations.com. We aim to respond to all inquiries within one business day.

2.4.2 Community Edition

We provide a Community Edition to enable you to evaluate, as well to support open-source developers that contribute to Helios, our containerized workload platform.

The Community Edition enables you to run up to 2 workloads for free, indefinitely and without any other restrictions on workload functionality.

2.4.3 User & Workload Allowance

If your license is issued for a set amount of users, each user is allowed to spawn as many workloads as they wish. This means you can have up to 20 unique users spawn multiple machines each. The license cap counts unique users across all your workloads.

There is no limit on the amount of users allowed to log into the platform. Only users with active workloads count against the allowance.

For workload-bound licenses, the amount of users is not considered and only the amount of running workloads is evaluated.

2.5 Expand your Orion cluster

2.5.1 Cloud

If you are using a cloud deployment, we recommend you install an Autoscaler for your cluster. With an autoscaler, whenever you request a workload, Kubernetes will check for available compute resources.

If your current cluster cannot run the workload you had requested, it will automatically bring online another node. Once you configure your autoscaler, it will also bring down any unused nodes, making sure you are not being charged for them and are utilizing only the necessary resources.

For AWS deployments, we highly recommend:

- Karpenter
- Cluster Autoscaler

Offline/air-gapped considerations

For offline installations, make sure that:

- the `juno-fx/ansible-ee:latest` image can be pulled from your nodes via the configuration in /etc/rancher/k3s/registries.yaml
- this configuration is set up automatically when using our install helper script found in the Quick Start Guide
- if you adjust it, please make sure to do so across all your control plane nodes
- when expanding the cluster, the file will be copied from your control plane node to any newly added nodes
- when you have multiple nodes, this will be performed on one of them. That is why it's important this file is consistent across the board.

2.5.2 On-Premise (k3s)

You can expand your cluster in genesis via the network page. Read more about node provisioning here

2.6 Backup & Disaster Recovery

As part of a production installation, you should make sure to set up regular backups.

This guide will walk you through all the state that Orion stores and relies on. We will suggest a prebuilt plugin we provide to back it up - however if you already have a Kubernetes backup strategy, you can of course instead plug in your existing tooling.

Note that this only walks you through backing up data related to Orion and its core functionality.

It does not cover backing up data you attach to workloads running within the platform. For example, when you mount in a PVC attached to your existing NFS storage, you must ensure it is backed up separately.

2.6.1 What Needs To Be Backed Up

> **Important**
>
> When performing a restore, make sure to do it in the order outlined below, from 1 to 10

The below lists all Kubernetes resources we recommend backing up. We do not currently rely on any state stored outside the Kubernetes API. This list is all you need to get your Orion installation back up & running in a Disaster Recovery scenario:

1. The TLS secret you use for your ingress.
2. The `juno-auth-secret` Secret in the `argocd` namespace.
3. The `applications.argoproj.io` `genesis` resource.
4. All `users.juno-innovations.com` resources.
5. All `groups.juno-innovations.com` resources.
6. All Secrets in the `argocd` namespace suffixed with `-token`.
7. All `applications.argoproj.io` resources annotated with `terra.juno-innovations.com/plugin_project`.
8. All ConfigMaps in the `argocd` namespace labeled with `kuiper.juno-innovations.com/template` =.
9. All `applications.argoproj.io` resources labeled with `juno-innovations.com/provider`.
10. All resources in the per-environment namespace. This will contain your storage configuration.

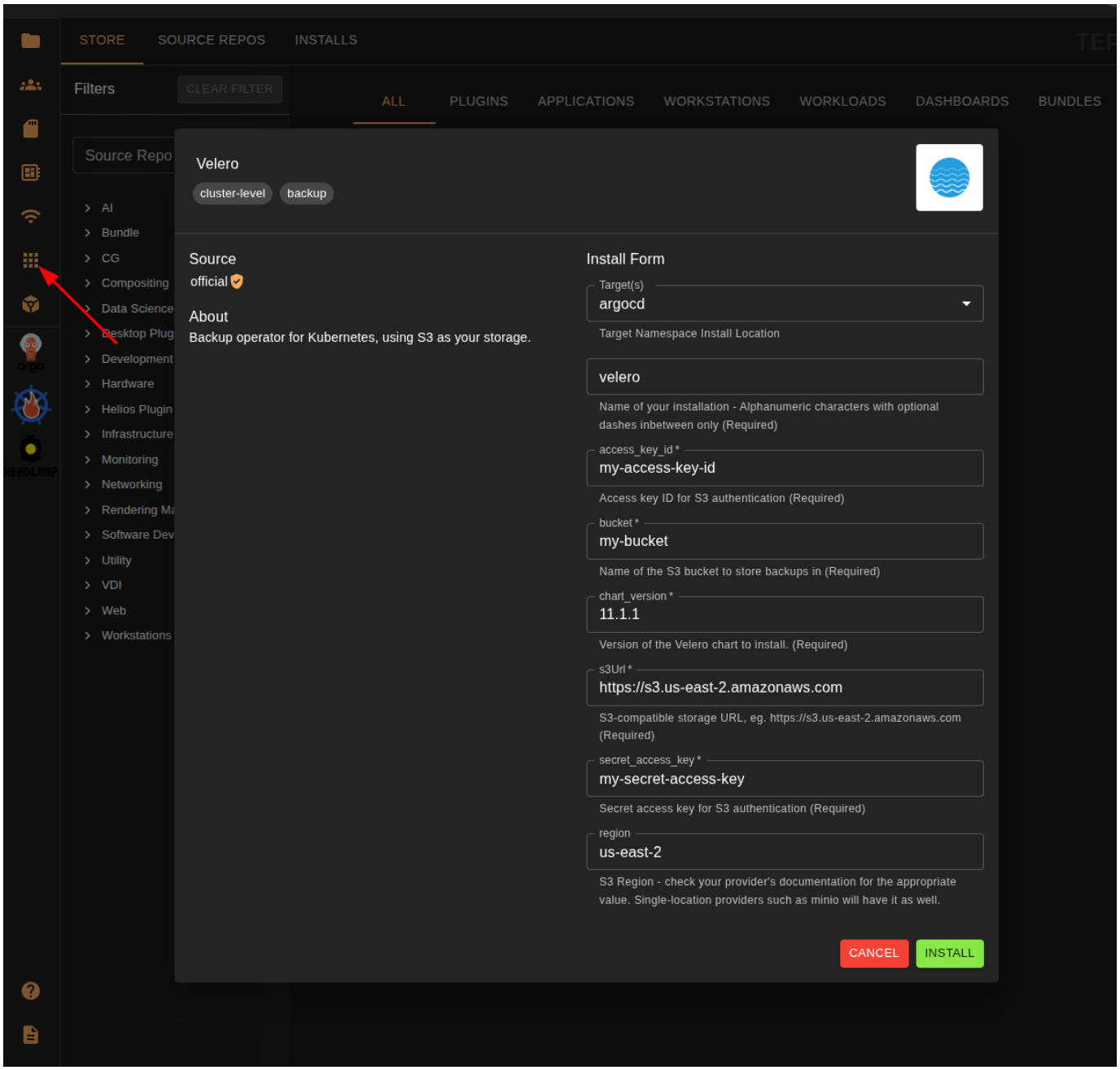
2.6.2 Preconfigured Velero plugin

Within our Terra App Store, you can find a preconfigured set of Velero plugins. Setting them both up will take care of all necessary backups for the Orion platform.

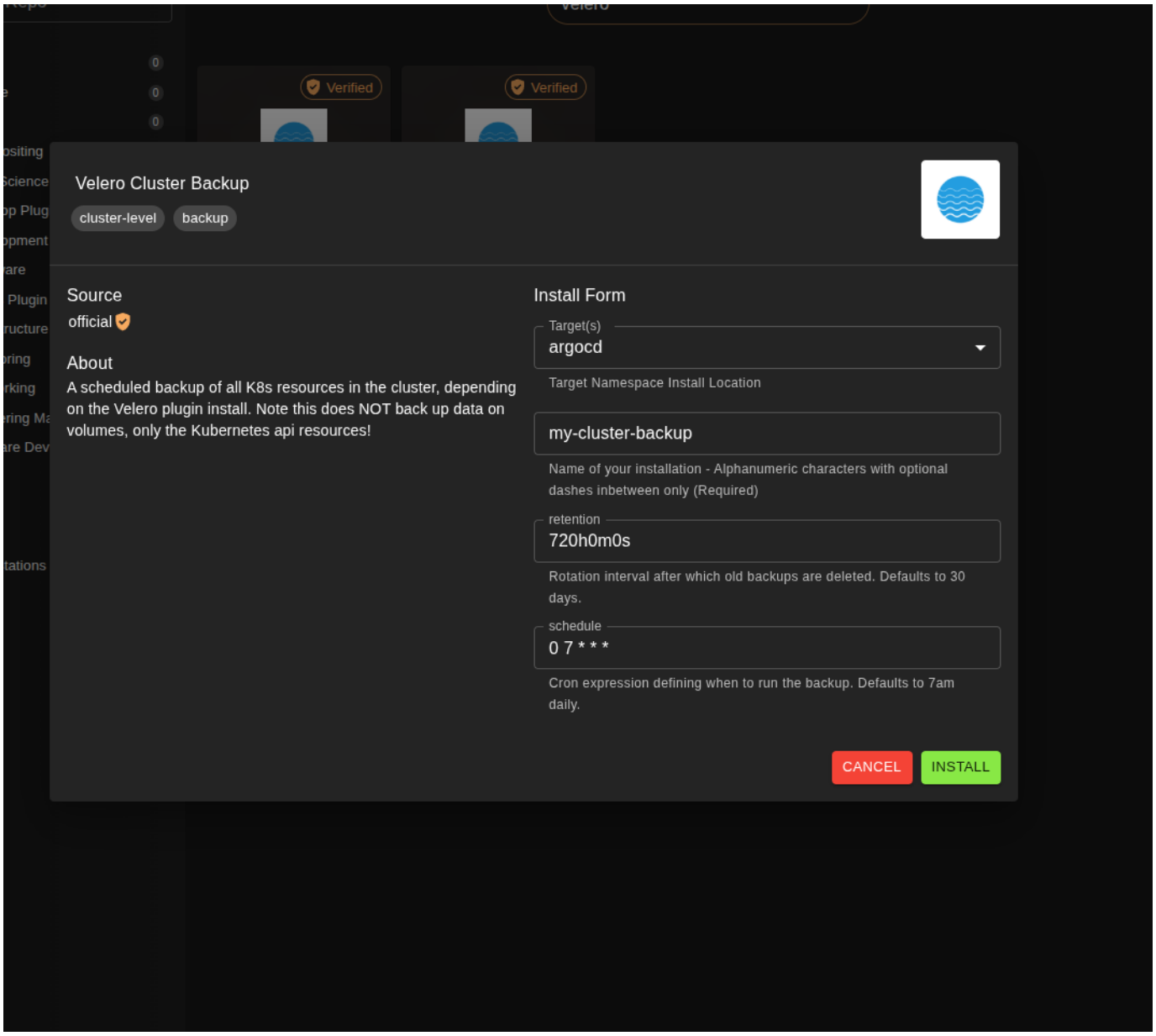
It can upload to any S3-compatible storage, such as MinIO, Wasabi, etc.

We recommend AWS S3, however anything compatible with the protocol will be up to the task.

To install Velero, simply go to the app store and install "Velero". That deploys the operator and points it to your storage backend. It does not yet schedule backups.



Having installed it, you can then install the "Velero Cluster Backup" plugin. This will back up all resources in your Kubernetes cluster to S3. Keep in mind this will also back up non-Orion Kubernetes resources.



It will not back up volumes/PVCs - you can expect the backup size to be very conservative thanks to that. While you can reuse Velero for PVC backups, this is currently outside the scope of the plugin - if you'd like to do that, we currently recommend setting up Velero with a custom configuration.

2.6.3 Performing a restore

Depending on whether you are performing a full Disaster Recovery scenario, migrating some data or recovering a single resource, you might want to filter for resources listed in What Needs To Be Backed Up section.

Once you know the scope of resources you are restoring, you can target them through the Velero CLI. Refer to the upstream documentation here for exact steps on the restore procedure.

2.7 Advanced Deployments

2.7.1 Kubernetes cluster setup

Juno can be installed on any Kubernetes cluster that can meet its requirements - we are vendor-agnostic. You can find the requirements for a functioning cluster under:

System Requirements

To make it easier for you to get up & running, we provide platform-specific guides. For on-premises deployments, you can find the docs here:

- Cluster setup recommendations:
- Our Oneclick installer - this is the recommended installation method. It will both set up a K3s cluster, and install Orion.
- Installing k3s with Ansible - this will guide you through the setup using our Ansible playbooks.
- Distribution-specific recommendations:
- Debian
- Rocky Linux
- Ubuntu

2.7.2 Cloud Providers

Amazon EKS

EKS PREREQUISITES & PLANNING GUIDE

Overview

This guide covers all prerequisites, planning considerations, and preparation steps required before deploying Orion on AWS EKS. Proper preparation ensures a smooth deployment and optimal performance.

Estimated Setup Time: 30-45 minutes

Required Skills & Knowledge

Before beginning your Orion deployment, ensure you have foundational knowledge in these areas:

AWS Foundational Knowledge

- **Amazon EC2**: Instance types, security groups, key pairs
- **Amazon EKS**: Managed Kubernetes concepts and architecture
- **Amazon VPC**: Networking, subnets, route tables, NAT gateways
- **AWS IAM**: Users, roles, policies, and permissions
- **Amazon Route 53**: DNS management and hosted zones
- **EKSCTL**: Command-line tool for EKS cluster management

Kubernetes Fundamentals

- Pod, Service, Deployment, and ConfigMap concepts
- kubectl command-line interface proficiency
- Understanding of namespaces and resource management
- Basic troubleshooting with logs and events

Container Orchestration Concepts

- Container images and registries
- Resource requests and limits
- Health checks and probes
- Persistent volumes and storage classes

Technical Prerequisites

- **Command Line Proficiency**: Comfortable with bash/shell environments
- **YAML Configuration**: Ability to read and edit YAML files
- **Basic Networking**: Understanding of TCP/IP, DNS, and load balancing
- **Text Editor Skills**: Proficiency with vi/vim, nano, or similar editors

AWS Account Setup

AWS Account Requirements Account Access

- AWS account with administrative privileges
- AWS CLI installed and configured with appropriate credentials

Required AWS Service Access

Your AWS role/user must have access to these services:

- **Amazon EKS**: Managed Kubernetes service
- **Amazon EC2**: Virtual machine instances for worker nodes
- **Amazon VPC**: Virtual private cloud networking
- **AWS IAM**: Identity and access management
- **Amazon EBS**: Elastic block storage for persistent volumes
- **Elastic Load Balancing**: Application and network load balancers
- **Amazon ECR**: Elastic container registry (optional but recommended)
- **Amazon Route 53**: DNS management

IAM Permissions Required

Your deployment user/role needs these AWS managed policies:

- `PowerUserAccess` (recommended for simplicity)

Partner Ecosystem Integration

Deployment Partner Strategy

Juno Innovations works with certified deployment partners who provide specialized expertise and integration services:

Partner Responsibilities

- **Custom Integration**: Partners handle integration with customer-specific tools and workflows
- **Monitoring Setup**: Configuration of customer's preferred monitoring and alerting platforms
- **Dashboard Creation**: Custom dashboard development using customer's existing observability stack
- **Operational Training**: Team training on integrated Orion platform within customer environment

Customer Operational Tools Integration

We recommend customers leverage their existing operational ecosystem:

Monitoring Platforms:

You can easily monitor your Orion deployment with any Prometheus or OpenMetric-capable monitoring stack.

For greenfield deployments, we recommend using CloudWatch and its EKS integration for an out-of-the-box setup. It will let you quickly start tracking your resource utilization and mission-critical metrics.

Operational Dashboards:

- Integration with existing NOC/SOC dashboards
- Custom executive reporting platforms
- IT service management systems (ServiceNow, ITSM)

Partner Network Benefits

- **Local Expertise**: Partners provide regional and industry-specific knowledge
- **Faster Implementation**: Leverage partner experience for accelerated deployments
- **Ongoing Support**: Local support augmenting Juno's global support capabilities
- **Custom Solutions**: Tailored implementations meeting specific organizational requirements

Licensing Requirements

Orion Platform Licensing

- **Contact Required**: Reach out to sales@juno-innovations.com
- **License Types**: User-based or node-based licensing available
- **Trial Options**: POC licenses available for evaluation
- **Support Tiers**: Different support levels based on license type

Third-Party Software Licenses Recommended Operational Tools

- **GitOps Platform**: ArgoCD comes pre-packaged with Orion for GitOps-based deployment and configuration management
- **Monitoring Integration**: Customers should leverage existing monitoring platforms (Prometheus, Grafana, Datadog, etc.)
- **Dashboard Solutions**: We rely on customers and deployment partners to use their preferred dashboarding solutions
- **Observability**: Orion can be monitored via any standard Kubernetes monitoring tool.

DNS Requirements

Domain Setup

- **Registered Domain**: You need a domain for ingress access
- **DNS Management**: Route 53 recommended for seamless AWS integration
- **Subdomain Strategy**: Plan subdomain structure (e.g., orion.yourcompany.com)
- **SSL Certificates**: AWS Certificate Manager can provide free SSL certs

DNS Configuration

```
genesis.yourcompany.com     -> Orion Genesis UI
<project>.yourcompany.com   -> Project deployment
```

External Dependencies

Container Registry Access

- **Amazon ECR**: Recommended for private images
- **Docker Hub**: For public images (rate limiting considerations)
- **Private Registry**: If using custom internal registry
- **Network Access**: Ensure nodes can pull images from chosen registry

Deployment and Operations Recommendations

-
- **GitOps Deployment**: We recommend ArgoCD for GitOps-based deployment and configuration management
- **Monitoring Solutions**: Customers should use their existing monitoring and dashboard solutions
- **Partner Integration**: We rely on deployment partners and customers to integrate with their preferred operational tools

Pre-Installation Checklist

AWS Prerequisites

- [x] AWS account with administrative access
- [x] AWS CLI installed and configured
- [x] EKSCTL installed (latest version)
- [x] kubectl installed (compatible with EKS version)
- [x] Service quotas verified and increased if needed

Domain and DNS

- [x] Domain registered and accessible
- [x] Route 53 hosted zone created (if using Route 53)
- [x] DNS propagation verified
- [x] SSL certificate planned (AWS Certificate Manager recommended)

Licensing and Access

- [x] Orion license obtained or trial arranged
- [x] Container registry access confirmed

Infrastructure Planning

- [x] Sizing requirements determined
- [x] Region and availability zones selected
- [x] Backup and recovery strategy planned

Operational Integration Planning

- [x] GitOps platform selected (ArgoCD required)
- [x] Existing monitoring and dashboard platforms identified

Support Resources

- **Pre-sales Support**: sales@juno-innovations.com
- **Technical Support**: support@juno-innovations.com
- **AWS Support**: Your AWS support plan
- **Community**: Kubernetes and EKSCTL community resources

Important: Don't skip the prerequisites! Proper preparation significantly reduces deployment time and prevents common issues.

Disclaimer: All cost estimates and deployment times are based on current AWS pricing and may vary by region and usage patterns. Contact our sales team for precise pricing and customized cost analysis based on your specific requirements.

AWS EKS DEPLOYMENT GUIDE

Amazon EKS

Overview

EKS provides a managed Kubernetes deployment optimized for deploying Orion's Unified Compute Plane. This guide covers the deployment process using EKSCTL for automated cluster provisioning and configuration.

Time Estimates:

- Prerequisites setup: 30-45 minutes
- EKS cluster + Orion deployment: 20-30 minutes
- **Total deployment time: 1.5 hours**

Prerequisites: Before starting, ensure you've completed the EKS Prerequisites & Planning guide.

For comprehensive EKS documentation, see AWS Documentation

Juno EKS Deployment Repo

Juno provides an EKS Deployment repo on GitHub that can be used as a starting point for your Orion deployment:

- **EKS Deployment**

Deployed Resources

The EKS Deployment repo uses EKSCTL to deploy the following resources:

- **Amazon EKS Cluster**: Managed Kubernetes control plane
- **Single EKS Managed Node Groups**: Initial worker node
- **VPC with Public and Private Subnets**: Networking infrastructure
- **Karpenter**: Cluster autoscaler for dynamic node provisioning
- **NGINX Ingress Controller**: Manages inbound traffic to services using a pre-configured NLB

Deployment Architecture Options

Orion supports three primary deployment architectures on AWS EKS, each optimized for different security, performance, and cost requirements:

Architecture Selection Guide

Architecture	Security Level	Cost	Complexity	Use Cases
Private Single AZ	Highest	Medium	Low	Development, high-security environments
Public Single AZ	Medium	Lowest	Lowest	Development, testing, cost-sensitive deployments
Public Multi-AZ	High	Highest	Medium	Production, high availability requirements

Private Single AZ Deployment

Best for: High-security environments, development, air-gapped deployments

Architecture Features:

- All nodes deployed in private subnets
- NAT Gateway provides secure internet access
- Network Load Balancer for internal traffic
- Enhanced security isolation
- Single availability zone for cost optimization
- EKSCTL simplified deployment

Security Benefits:

- No direct internet access to worker nodes
- All traffic routed through NAT Gateway
- Internal load balancing only
- Ideal for compliance requirements

Public Single AZ Deployment

Best for: Development environments, cost-sensitive deployments, proof-of-concept

Architecture Features:

- Nodes in public subnets with direct internet access
- Simplified networking configuration
- Network Load Balancer for external access
- Lower infrastructure costs
- EKSCTL simplified deployment

Cost Benefits:

- No NAT Gateway costs
- Reduced networking complexity
- Fastest deployment option
- Minimal infrastructure overhead

Public Multi-AZ Deployment

Best for: Production environments, high availability requirements, enterprise deployments

Architecture Features:

- High availability across multiple availability zones
- Automatic failover capabilities
- Distributed workload placement
- Production-grade reliability
- Enhanced fault tolerance

Availability Benefits:

- Zone-level fault tolerance
- Automatic node replacement
- Load distribution across AZs
- Mission-critical uptime

Deployment Overview

When deploying Orion on AWS EKS via EKSCTL, the platform creates a comprehensive infrastructure stack designed for security, scalability, and high performance.

Core Infrastructure Components **Amazon EKS Cluster**

- Managed Kubernetes control plane with auto-scaling worker nodes
- High availability across multiple Availability Zones
- Integrated with AWS IAM for role-based access control

Compute Resources

- **EKS Managed Node Groups**: CPU and GPU-optimized instances
- **Auto Scaling Groups**: Dynamic scaling based on workload demands
- **Spot Instance Integration**: Cost optimization for batch workloads

Load Balancing & Traffic Management

- **Network Load Balancer (NLB)**: High-performance TCP/UDP load balancing for specific workloads
- **NGINX Ingress Controller**: Advanced traffic routing and load balancing within the cluster

Storage Resources (Customer-Managed)

Important: Orion provides compute orchestration only. All storage configuration and data management remains under customer control.

Amazon EBS (Customer Configured)

- Customer configures encrypted persistent storage for their applications
- GP3 volumes recommended for optimal price-performance balance
- Customer responsibility: backup, snapshot, and lifecycle management

Amazon EFS (Optional, Customer Managed)

- Customer may configure shared file system for multi-node applications
- Customer responsibility: access control, backup, and cost management

Amazon S3 (Customer Assets)

- Customer configures object storage for their media files and project assets
- Customer responsibility: lifecycle policies, access control, and cost optimization
- Orion provides compute access to customer-configured storage

Network Components **Virtual Private Cloud (VPC)**

- Isolated network environment with dedicated IP address range
- Public and private subnets across multiple Availability Zones
- NAT Gateways for secure internet access from private subnets

Security Groups

- Stateful firewall rules controlling traffic at instance level
- Separate groups for different application tiers (web, application, database)
- Least privilege access principles

Security & Compliance Components **AWS IAM (Identity and Access Management)**

- Service accounts and roles with least privilege principles

EKSCTL Deployment Method

We **highly recommend using EKSCTL** for Orion deployments as it: - Provisions all required AWS resources automatically - Configures proper IAM roles and security groups - Sets up managed node groups with best practices - Handles VPC and networking configuration - Provides declarative cluster configuration

For comprehensive EKSCTL documentation, see EKSCTL Documentation.

Required IAM Roles

EKSCTL automatically creates these essential IAM roles:

EKS Cluster Service Role

- **Purpose**: Allows EKS to manage cluster on your behalf
- **Permissions**: EKS service permissions, VPC management
- **Managed Policies**: `AmazonEKSClusterPolicy`

Node Group Instance Role

- **Purpose**: Allows worker nodes to join cluster and run pods
- **Permissions**: ECR access, CNI management, node registration
- **Managed Policies**:
- `AmazonEKSWorkerNodePolicy`
- `AmazonEKS_CNI_Policy`
- `AmazonEC2ContainerRegistryReadOnly`

Load Balancer Controller Role

- **Purpose**: Manages ALB/NLB resources for ingress
- **Permissions**: EC2 and ELB service permissions
- **Custom Policy**: AWS Load Balancer Controller policy

Cluster Autoscaler Role

- **Purpose**: Automatically scales node groups based on demand
- **Permissions**: Auto Scaling Group management
- **Custom Policy**: Cluster autoscaler permissions

EBS CSI Driver Role

- **Purpose**: Manages EBS volumes for persistent storage
- **Permissions**: EC2 volume operations
- **Managed Policy**: `AmazonEBSCSIDriverPolicy`

Node Configuration Strategy

Recommended Node Groups

We recommend a multi-node group strategy optimized for Orion's workload diversity:

1. CPU Node Group

- **Purpose**: General Orion services and control plane components
- **Instance Types**: `t3.large`, `t3.xlarge`
- **Scaling**: 2-10 nodes based on demand

2. GPU Node Group

- **Purpose**: GPU workloads and AI/ML processing
- **Instance Types**: `g4dn.xlarge`, `g4dn.2xlarge`, `g5.xlarge`
- **Scaling**: 1-20 nodes based on workload demands

3. Spot Instance Node Group

- **Purpose**: Cost optimization for batch processing workloads
- **Instance Types**: Mix of CPU and GPU spot instances
- **Cost Savings**: 50-90% reduction for interruptible workloads
- **Use Cases**: Rendering, batch AI processing, development environments

AWS Cost Considerations

Orion Platform Costs vs AWS Infrastructure Costs

Important: AWS infrastructure costs vary by region and change over time. Please check current AWS pricing for accurate estimates.

Orion Platform Licensing:

- Contact sales@juno-innovations.com for pricing
- User-based or node-based licensing available
- Includes compute optimization platform only

Customer AWS Infrastructure Costs:

- EKS cluster management fees
- EC2 instances: Variable based on customer requirements
- Storage (EBS, EFS, S3): Customer configured and billed
- Networking: Customer responsibility

Cost Optimization Value:

- Up to 56% reduction in compute resource costs through more aggressive auto-scaling and resource density
- In some cases, you may see: 90%+ GPU, 85%+ CPU, 85%+ RAM utilization, allowing for fewer nodes and lower costs
- Savings apply to compute resources, storage costs remain customer-managed

Required AWS Services

These services are required for Orion deployment:

- **EKS Cluster Management**: Per-cluster hourly fees
- **EC2 Instances**: Variable based on node configuration and usage
- **EBS Volumes**: Storage costs based on volume type and size
- **VPC Networking**: Data transfer charges for standard usage

Optional AWS Services

These services enhance functionality but are not required:

- **Amazon EFS**: Shared storage with per-GB monthly costs

Additional Resources

- EKSCTL Documentation - Comprehensive EKSCTL reference
- EKS Best Practices Guide - AWS recommendations

Support

For deployment assistance:

- **Technical Support**: support@juno-innovations.com
- **Sales & Licensing**: sales@juno-innovations.com

Disclaimer: All cost estimates and deployment times are based on current AWS pricing and may vary by region and usage patterns. Contact our sales team for precise pricing and customized cost analysis based on your specific requirements.

AWS SECURITY CONFIGURATION GUIDE

Amazon EKS

Overview

This guide provides comprehensive security configuration requirements and best practices for deploying Orion on AWS infrastructure in compliance with AWS Foundational Technical Review (FTR) standards. These configurations ensure enterprise-grade security while maintaining operational excellence.

Administrative Privileges

Principle of Least Privilege

Orion deployments follow strict least privilege principles, ensuring no component requires unnecessary administrative access:

Orion Platform Privileges

- **No Root Access**: Orion containers run as non-privileged users
- **Service Account Limitations**: Kubernetes service accounts have minimal required permissions
- **API Access**: Limited to specific resource types and namespaces
- **Network Access**: Restricted to essential ports and protocols

EKSCTL Deployment Privileges

During initial deployment, EKSCTL requires:

- **EKS Cluster Management**: Creating and managing EKS clusters
- **EC2 Instance Management**: Managing worker nodes and security groups
- **VPC Management**: Creating subnets, route tables, and NAT gateways
- **IAM Role Management**: Creating service-specific roles with limited scope

Post-Deployment: Administrative privileges can be reduced to operational maintenance level.

Operational Privileges

For day-to-day operations:

- **Platform Updates**: Limited to Orion namespace and resources
- **Monitoring**: Read-only access to system metrics and logs
- **Troubleshooting**: Specific diagnostic permissions without system-wide access

Administrative Access Controls

```
# Example Orion service account with minimal privileges
apiVersion: v1
kind: ServiceAccount
metadata:
  name: orion-platform
  namespace: orion-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: orion-platform-role
  namespace: orion-system
rules:
- apiGroups: [""]
  resources: ["pods", "services", "endpoints"]
  verbs: ["get", "list", "watch", "create", "update", "patch"]
- apiGroups: ["apps"]
  resources: ["deployments", "replicasets"]
  verbs: ["get", "list", "watch", "create", "update", "patch"]
```

IAM Roles and Policies

Recommended EKSCTL

EKSCTL ships with its own system for creating and managing IAM roles and policies. We highly recommend following their recommendations from AWS directly:

- EKSCTL IAM Roles and Policies

Required IAM Roles

Orion deployment creates specific IAM roles with clearly defined boundaries:

EKS Cluster Service Role

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Attached Policies: - `AmazonEKSClusterPolicy`

Purpose: Allows EKS service to manage cluster resources on your behalf

EKS Node Group Role

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Attached Policies:

- `AmazonEKSWorkerNodePolicy`
- `AmazonEKS_CNI_Policy`
- `AmazonEC2ContainerRegistryReadOnly`

Purpose: Allows worker nodes to join cluster and access ECR

AWS Load Balancer Controller Role

Uses OIDC provider for secure service account mapping:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::ACCOUNT-ID:oidc-provider/oidc.eks.REGION.amazonaws.com/id/EXAMPLE"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "oidc.eks.REGION.amazonaws.com/id/EXAMPLE:sub": "system:serviceaccount:kube-system:aws-load-balancer-controller",
          "oidc.eks.REGION.amazonaws.com/id/EXAMPLE:aud": "sts.amazonaws.com"
        }
      }
    }
  ]
}
```

Policy Boundaries and Resource Tagging

All resources created by Orion deployment include consistent tagging:

```
tags:
  Environment: "production"
  Application: "orion-platform"
  ManagedBy: "juno-innovations"
  CostCenter: "it-infrastructure"
  DataClassification: "internal"
```

Encryption Configuration

Encryption at Rest **EKS Cluster Encryption**

```
# EKSCTL configuration for cluster encryption
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: orion-cluster
```

```
  region: us-east-1
encryptionConfig:
  provider:
    keyArn: arn:aws:kms:us-east-1:ACCOUNT:key/KEY-ID
  resources:
    - secrets
```

EBS Volume Encryption

```
# Encrypted storage class
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: orion-encrypted-gp3
provisioner: ebs.csi.aws.com
parameters:
  type: gp3
  encrypted: "true"
  kmsKeyId: arn:aws:kms:us-east-1:ACCOUNT:key/KEY-ID
volumeBindingMode: WaitForFirstConsumer
allowVolumeExpansion: true
```

Encryption in Transit **TLS Configuration**

- All external communications use TLS 1.2 or higher
- Internal service mesh communications encrypted (optional but recommended)
- Database connections encrypted using SSL/TLS

```
# Ingress TLS configuration
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: orion-ingress
  annotations:
    kubernetes.io/ingress.class: "nginx"
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
spec:
  tls:
  - hosts:
    - orion.yourdomain.com
    secretName: orion-tls-secret
  rules:
  - host: orion.yourdomain.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: orion-frontend
            port:
              number: 80
```

Network Security

VPC Configuration **Network Isolation**

```
# EKSCTL VPC configuration
vpc:
  cidr: "10.0.0.0/16"
  subnets:
    public:
      public-subnet-1:
        cidr: "10.0.1.0/24"
        availabilityZone: "us-east-1a"
      public-subnet-2:
        cidr: "10.0.2.0/24"
        availabilityZone: "us-east-1b"
    private:
      private-subnet-1:
        cidr: "10.0.11.0/24"
        availabilityZone: "us-east-1a"
      private-subnet-2:
        cidr: "10.0.12.0/24"
        availabilityZone: "us-east-1b"
```

Security Groups

```
{
  "GroupName": "orion-worker-nodes-sg",
  "Description": "Security group for Orion EKS worker nodes",
  "SecurityGroupRules": [
    {
      "IpProtocol": "tcp",
      "FromPort": 443,
```

```
      "ToPort": 443,
      "CidrIpv4": "0.0.0.0/0",
      "Description": "HTTPS inbound"
    },
    {
      "IpProtocol": "tcp",
      "FromPort": 10250,
      "ToPort": 10250,
      "SourceSecurityGroupId": "sg-cluster-control-plane",
      "Description": "Kubelet API"
    }
  ]
}
```

Network ACLs

Additional network-level protection:

- Default deny all inbound traffic
- Explicit allow rules for required services
- Separate ACLs for public and private subnets

Kubernetes Network Policies

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: orion-network-policy
  namespace: orion-system
spec:
  podSelector:
    matchLabels:
      app: orion-platform
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: ingress-nginx
    ports:
    - protocol: TCP
      port: 8080
  egress:
  - to:
    - namespaceSelector: {}
    ports:
    - protocol: TCP
      port: 443
```

Key Management

AWS KMS Integration **Customer-Managed Keys**

```
# KMS key for Orion resources
ApiVersion: kms.aws.com/v1
Kind: Key
Metadata:
  Name: orion-platform-key
Spec:
  Description: "Encryption key for Orion platform resources"
  KeyUsage: ENCRYPT_DECRYPT
  KeySpec: SYMMETRIC_DEFAULT
  Policy:
    Version: "2012-10-17"
    Statement:
      - Sid: Enable IAM User Permissions
        Effect: Allow
        Principal:
          AWS: "arn:aws:iam::ACCOUNT:root"
        Action: "kms:*"
        Resource: "*"
      - Sid: Allow EKS Service
        Effect: Allow
        Principal:
          Service: eks.amazonaws.com
        Action:
          - kms:Decrypt
          - kms:DescribeKey
        Resource: "*"
```

Key Rotation

We highly recommend enabling automatic key rotation for all customer-managed keys. You can use the secrets-store-csi-driver from the Kubernetes SIGs to manage secrets and keys within your cluster. Please follow the official documentation for installation and follow the core concepts. Once installed, configure the AWS provider as per the AWS provider documentation. You can then follow the usage documentation to setup the rotation of your keys.

- Automatic key rotation enabled for customer-managed keys
- Annual rotation policy with audit trail
- Immediate rotation capability for security incidents

Certificate Management **AWS Certificate Manager Integration**

```
# Certificate for Orion ingress
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: orion-tls-cert
  namespace: orion-system
spec:
  secretName: orion-tls-secret
  issuerRef:
    name: letsencrypt-prod
    kind: ClusterIssuer
  dnsNames:
  - orion.yourdomain.com
  - api.orion.yourdomain.com
```

Internal Service Certificates

- Mutual TLS for internal service communication
- Certificate lifecycle management
- Automatic renewal and distribution

IMDSv1 Disable Support

EC2 Metadata Service Configuration **IMDSv2 Enforcement**

All EKS worker nodes configured to require IMDSv2:

```
# EKSCTL node group configuration
nodeGroups:
  - name: orion-nodes
    instanceType: t3.medium
    desiredCapacity: 3
    ssh:
      publicKeyName: eks-nodes
    iam:
      withAddonPolicies:
        imageBuilder: true
    metadata:
      httpEndpoint: enabled
      httpTokens: required  # Enforces IMDSv2
      httpPutResponseHopLimit: 2
```

Pod-Level IMDS Restrictions

```
# Security context preventing IMDS access
apiVersion: v1
kind: Pod
metadata:
  name: orion-workload
spec:
  securityContext:
    runAsNonRoot: true
    runAsUser: 1000
    fsGroup: 2000
  containers:
  - name: orion-app
    image: juno/orion:latest
    securityContext:
      allowPrivilegeEscalation: false
      readOnlyRootFilesystem: true
      capabilities:
        drop:
        - ALL
```

Security Monitoring and Compliance

CloudTrail Configuration **API Logging**

```
{
  "TrailName": "orion-security-audit",
  "S3BucketName": "orion-cloudtrail-logs",
  "IncludeGlobalServiceEvents": true,
  "IsMultiRegionTrail": true,
  "EnableLogFileValidation": true,
  "EventSelectors": [
    {
      "ReadWriteType": "All",
      "IncludeManagementEvents": true,
      "DataResources": [
        {
          "Type": "AWS::S3::Object",
          "Values": ["arn:aws:s3:::orion-*/*"]
        }
      ]
    }
  ]
}
```

AWS Config Rules **Compliance Monitoring**

- EKS cluster endpoint access configuration
- Security group compliance rules
- Encryption at rest validation
- IMDSv2 enforcement monitoring

```
{
  "ConfigRuleName": "eks-cluster-endpoint-no-public-access",
  "Source": {
    "Owner": "AWS",
    "SourceIdentifier": "EKS_ENDPOINT_NO_PUBLIC_ACCESS"
  },
  "Scope": {
    "ComplianceResourceTypes": [
      "AWS::EKS::Cluster"
    ]
  }
}
```

Security Best Practices Checklist

Pre-Deployment Security

- ✅ Customer-managed KMS keys created and configured
- ✅ IAM roles follow principle of least privilege
- ✅ VPC designed with proper network segmentation
- ✅ Security groups configured with minimal required access
- ✅ IMDSv2 enforced on all EC2 instances

Post-Deployment Security

- ✅ All communications encrypted in transit
- ✅ Storage encrypted at rest with customer-managed keys
- ✅ Network policies implemented for pod-to-pod communication
- ✅ CloudTrail logging enabled and monitored
- ✅ AWS Config rules deployed for compliance monitoring

Ongoing Security Maintenance

- ✅ Regular security group audit and cleanup
- ✅ Certificate renewal monitoring and automation
- ✅ Key rotation compliance verification
- ✅ Security patch management for worker nodes
- ✅ Compliance reporting and audit preparation

Security Incident Response

Detection and Alerting

- CloudWatch alarms for security-relevant events
- AWS Security Hub integration for centralized findings
- VPC Flow Logs analysis for network anomalies

Response Procedures

1. **Incident Classification**: Severity assessment using established criteria
2. **Containment**: Isolate affected resources and limit blast radius
3. **Investigation**: Forensic analysis using CloudTrail and VPC Flow Logs
4. **Recovery**: Restore services from known-good configurations
5. **Lessons Learned**: Post-incident review and security improvements

Compliance Documentation

Audit Requirements

- **Access Logs**: Complete audit trail of all administrative actions
- **Configuration Changes**: Version-controlled infrastructure as code
- **Security Controls**: Evidence of implemented security measures
- **Compliance Reports**: Regular assessment against security frameworks

Documentation Maintenance

- Security configuration baselines
- Incident response playbooks
- Compliance evidence collection
- Regular security assessment reports

Backup and Recovery

Please refer to the official EKS documentation on how to provide safety and recovery for your cluster in this guide.

System Updates and Patch Management

We highly recommend using AWS Managed EKS AMI's for your worker nodes. These AMI's are regularly updated and patched by AWS to ensure you have the latest security updates and patches. You can find more information on how to use these AMI's in the official documentation here. The full documentation for how to build and manually update your system is very well documented here. We recommend using the integrated ami's from AWS to ensure you have the latest security patches and updates. If you need additional libraries, you can build on top of their work to add additional functionality (updated drivers, bleeding edge libraries, etc).

Disclaimer: All cost estimates and deployment times are based on current AWS pricing and may vary by region and usage patterns. Contact our sales team for precise pricing and customized cost analysis based on your specific requirements.

AWS COST AND SIZING GUIDE

Amazon EKS

Overview

This guide provides comprehensive cost analysis and sizing recommendations for deploying Orion's Unified Compute Plane on AWS infrastructure. Understanding both AWS infrastructure costs and Orion platform licensing helps organizations plan budgets and optimize their compute investment.

Cost Responsibility Matrix

AWS Infrastructure Costs (Customer Responsibility) **Mandatory AWS Services**

These services are required for Orion deployment:

Service	Cost Structure
EKS Cluster Management	Standard Managed EKS
EC2 Instances	Per-hour instance pricing
EBS Volumes	Standard EC2 Pricing
VPC Networking	Data transfer charges

Optional AWS Services

These services enhance functionality but are not required:

Service	Estimated Monthly Cost
Amazon EFS	Based on shared storage needs

Orion Platform Licensing

Important: Orion platform licensing is separate from AWS infrastructure costs.

Cost will vary based on customer deployment. Orion runs on top of customer-purchased EC2 instances based on typical AWS pricing. There is an additional cost for the Orion platform built on a per user/month plan, and this pricing will evolve over time.

No Sizing Restrictions: Orion is designed as a lightweight orchestration platform that doesn't require significant compute overhead to run effectively.

Licensing Models

Refer to our Licensing Guide for detailed information on available licensing tiers.

What's Included in Orion License

- Unified Compute Plane orchestration software
- Container-native workload deployment
- Technical support based on license tier

What's NOT Included in Orion License

- AWS infrastructure costs (customer purchases EC2 instances directly)
- Storage costs and management
- Data backup and recovery services
- Custom application development
- Third-party software licenses

Infrastructure Sizing Guide

Disclaimer: All cost estimates are examples based on historical AWS pricing. Please check current AWS pricing for accurate estimates as costs vary by region and change over time.

Service Limits

Please refer to the service limit recommendations from AWS here.

Small Deployment (Development/Testing)

Use Case: Development, testing, small teams (1-10 users)

Node Configuration

- **Control Plane**: Managed by EKS (included with EKS cost)
- **Service Nodes**: 2x (4 vCPU, 8GB RAM each)
- **Workload Nodes**: Autoscaled based on demand (optional)

AWS Infrastructure Components

- EKS cluster management
- EC2 instances (CPU and optional GPU)
- EBS storage
- VPC networking

Orion Platform Value

- 56% cost reduction through resource optimization and autoscaling
- 92% GPU utilization vs industry standard 35-65%
- 87% CPU utilization vs industry standard 40-60%
- On-Demand workload deployment in minutes

Medium Deployment (Production)

Use Case: Production environments, medium teams (10-50 users)

Node Configuration

- **Control Plane**: Managed by EKS
- **CPU Nodes**: 3x (4 vCPU, 8GB RAM each)
- **Workload Nodes**: Autoscaled based on demand (optional)

AWS Infrastructure Components

- EKS cluster management
- EC2 instances (CPU and optional GPU)
- EBS storage
- VPC networking

Cost Optimization Benefits

- 56% cost reduction through Orion's resource optimization
- Significant savings compared to traditional infrastructure deployments
- Improved resource utilization across all compute resources

Large Deployment (High Performance)

Use Case: Large teams, high-performance workloads (50+ users)

Node Configuration

- **Control Plane**: Managed by EKS
- **CPU Nodes**: 5x (4 vCPU, 8GB RAM each)
- **Workload Nodes**: Autoscaled based on demand (optional)

AWS Infrastructure Components

- EKS cluster management
- EC2 instances (CPU and optional GPU)
- EBS storage
- VPC networking

Enterprise Cost Optimization

- Up to 56% cost reduction through Orion's Unified Compute Plane optimization
- Spot instance integration for additional cost savings
- Resource optimization across large-scale deployments

Enterprise Deployment (Custom Scale)

Use Case: Enterprise-wide deployments, specialized workloads (100+ users)

Scalability Considerations

- **Multi-AZ Deployment**: High availability across availability zones
- **Auto Scaling**: Dynamic scaling based on workload demands
- **Disaster Recovery**: Cross-region backup and failover capabilities
- **Compliance**: Additional security and audit requirements

Cost Variables

- Instance types based on specific workload requirements
- Storage requirements vary significantly by use case
- Network architecture complexity impacts costs
- Compliance requirements may add monitoring and logging costs

Contact Sales: For enterprise-scale deployments, contact sales@juno-innovations.com for custom sizing and pricing analysis.

Cost Optimization Strategies

Compute Cost Optimization **Resource Utilization Improvements**

Orion's Unified Compute Plane delivers measurable cost reductions:

- **GPU Utilization**: 92% vs industry standard 35-65%
- **CPU Utilization**: 87% vs industry standard 40-60%
- **RAM Utilization**: 85% vs industry standard 45-70%
- **Overall Cost Reduction**: 56% through resource optimization

Spot Instance Integration

- **Cost Savings**: 50-90% reduction for interruptible workloads
- **Workload Types**: Batch processing, rendering, development environments
- **Auto-Scaling**: Intelligent placement across on-demand and spot instances

Dynamic Resource Allocation

- **Just-in-Time Provisioning**: Resources allocated only when needed
- **Automatic Scaling**: Scale down during idle periods
- **Multi-Cloud Burst**: Overflow to most cost-effective cloud regions

Customer Storage Responsibilities

Important: Storage costs are customer-managed and not included in Orion licensing:

- **Data Backup**: Customer responsibility for backup strategies and costs
- **Storage Scaling**: Customer manages storage capacity and performance requirements
- **Lifecycle Management**: Customer controls data retention and archival policies

Network Cost Optimization **Data Transfer Minimization**

- **Regional Deployment**: Deploy close to users to minimize data transfer
- **CloudFront CDN**: Cache static content for global distribution
- **VPC Endpoints**: Reduce data transfer costs for AWS service access

ROI Calculation **Investment vs Savings**

- **Orion License Investment**: Contact sales for specific pricing
- **AWS Infrastructure Savings**: 56% reduction in compute costs
- **Operational Efficiency**: 98% reduction in deployment time
- **Team Productivity**: 33% increase in productive work time

Break-Even Analysis

Most customers achieve positive ROI within 3-6 months through:

- Reduced infrastructure waste
- Faster deployment cycles
- Improved team productivity
- Lower operational overhead

Support Resources

Cost Planning Assistance

- **Pre-sales Engineering**: Free cost estimation and sizing assistance
- **POC Support**: Guided proof-of-concept deployments with cost tracking
- **Migration Planning**: Professional services for complex migrations

Contact Information

- **Sales & Sizing**: sales@juno-innovations.com
- **Technical Support**: support@juno-innovations.com
- **Cost Optimization**: Included in Premium and Enterprise support tiers

---

Disclaimer: All cost estimates and deployment times are based on current AWS pricing and may vary by region and usage patterns. Contact our sales team for precise pricing and customized cost analysis based on your specific requirements.

AWS EKS ARCHITECTURE REFERENCE

Amazon EKS

Overview

This reference guide provides detailed architecture diagrams and deployment patterns for Orion's Unified Compute Plane on AWS EKS.

Architecture Components

Core Orion Platform Components

All deployment architectures include these essential Orion platform components:

Juno Workload Nodes

- **Workloads**: Primary compute orchestration for user workloads and applications
- **Helios**: Resource optimization and GPU utilization management (92% efficiency)
- **Purpose**: Customer-facing compute workloads achieving industry-leading resource utilization

Juno Support Nodes

- **Kuiper**: Workload management system for project namespaces
- **Terra**: Application marketplace and ecosystem management
- **Titan**: Authorization service and user management
- **Hubble**: Project portal and workload management dashboard
- **NGINX**: Ingress controller and load balancing
- **Genesis**: Platform initialization and configuration management

AWS Infrastructure Components **Amazon EKS Control Plane**

- Managed Kubernetes API server with high availability
- Automatic updates and security patches
- Multi-AZ control plane for production deployments
- Integrated with AWS IAM for authentication and authorization

Compute Resources

- **CPU Nodes**: General Orion services and control plane components
- **GPU Nodes**: High-performance workloads with optimized resource utilization
- **Auto Scaling Groups**: Dynamic scaling based on workload demands

Networking Components

- **VPC**: Isolated network environment with customizable CIDR blocks
- **Subnets**: Public and private subnet configurations per availability zone
- **Security Groups**: Application-layer firewall rules
- **Network Load Balancer**: High-performance Layer 4 load balancing
- **Internet Gateway**: External connectivity for public deployments
- **NAT Gateway**: Secure internet access for private deployments

Deployment Architecture Patterns

1. Private Single AZ Deployment

Recommended for: High-security environments, development, compliance requirements

Architecture Characteristics

- **Security Model**: Private subnets with no direct internet access
- **Internet Connectivity**: NAT Gateway provides secure outbound access
- **Load Balancing**: Network Load Balancer within VPC
- **Availability**: Single AZ with cost optimization
- **Compliance**: Suitable for regulated environments

Component Layout

```
AWS Account
└── VPC (10.0.0.0/16)
    └── Availability Zone
        ├── Private Subnet (10.0.11.0/24)
```

```
        │   ├── Juno Workload Nodes
        │   │   ├── Workloads (GPU optimized)
        │   │   └── Helios (Resource optimization)
        │   └── Juno Support Nodes
        │       ├── Kuiper (Platform core)
        │       ├── Terra & Titan (Ecosystem)
        │       ├── Hubble (Monitoring)
        │       ├── NGINX (Ingress)
        │       └── Genesis (Configuration)
        ├── NAT Gateway
        ├── Network Load Balancer
        └── Internet Gateway
```

Security Features

- All worker nodes in private subnets
- No direct internet access to compute resources
- Encrypted data in transit and at rest
- VPC-level network isolation

Cost Considerations

- NAT Gateway: AWS Cost + data processing
- Reduced complexity compared to multi-AZ
- Single AZ reduces cross-AZ data transfer costs

2. Public Multi-AZ Deployment

Recommended for: Production environments, high availability, enterprise deployments

Architecture Characteristics

- **High Availability**: Multiple availability zones with automatic failover
- **Fault Tolerance**: Zone-level isolation and redundancy
- **Load Distribution**: Workloads spread across AZs
- **Scalability**: Independent scaling per availability zone
- **Production Ready**: Enterprise-grade reliability

Component Layout

```
AWS Account
└── VPC (10.0.0.0/16)
    ├── Availability Zone A
    │   └── Public Subnet (10.0.1.0/24)
    │       ├── Juno Workload Nodes (AZ-A)
    │       └── Juno Support Nodes (AZ-A)
    ├── Availability Zone B
    │   └── Public Subnet (10.0.2.0/24)
    │       ├── Juno Workload Nodes (AZ-B)
    │       └── Juno Support Nodes (AZ-B)
    ├── Network Load Balancer (Multi-AZ)
    └── Internet Gateway
```

High Availability Features

- Cross-AZ workload distribution
- Automatic node replacement
- Multi-AZ load balancer
- Zero-downtime updates

Cost Considerations

- Higher compute costs due to redundancy
- Cross-AZ data transfer charges
- Enhanced availability justifies premium

3. Public Single AZ Deployment

Recommended for: Development, testing, cost-sensitive deployments, proof-of-concept

Architecture Characteristics

- **Cost Optimized**: Minimal infrastructure overhead
- **Simplified Networking**: Direct internet access for all nodes
- **EKSCTL Ready**: Simplified deployment configuration
- **Development Focus**: Ideal for non-production environments
- **Quick Setup**: Fastest deployment option

Component Layout

```
AWS Account
└── VPC (10.0.0.0/16)
    └── Availability Zone
        └── Public Subnet (10.0.1.0/24)
            ├── Juno Workload Nodes
            │   ├── Workloads (Cost optimized)
            │   └── Helios (Resource optimization)
            └── Juno Support Nodes
                ├── Kuiper (Platform core)
                ├── Terra & Titan (Ecosystem)
                ├── Hubble (Monitoring)
                ├── NGINX (Ingress)
                └── Genesis (Configuration)
```

Cost Optimization Features

- No NAT Gateway required
- Minimal networking complexity
- Single AZ reduces infrastructure costs
- Direct internet connectivity

Development Benefits

- Rapid deployment and teardown
- Simplified debugging and troubleshooting
- Cost-effective for temporary environments
- Easy EKSCTL configuration

Architecture Selection Decision Matrix

Requirement	Private Single AZ	Public Multi-AZ	Public Single AZ
High Security	Excellent	⚠ Good	⚠ Basic
High Availability	Single Point	Excellent	Single Point
Cost Optimization	⚠ Moderate	Highest	Lowest
Development	⚠ Suitable	Overprovisioned	Ideal
Production	⚠ Limited HA	Recommended	Not Recommended

Network Security Considerations

Security Groups Configuration **Orion Platform Security Groups**

- **Workload Nodes**: Ports 80, 443, 22 (SSH), custom application ports
- **Support Nodes**: Kubernetes API (6443), etcd (2379-2380), kubelet (10250)
- **Load Balancer**: HTTP (80), HTTPS (443), health check ports

Network ACLs

- Subnet-level security controls
- Default deny with explicit allow rules
- Separate ACLs for public and private subnets

Data Flow and Communication **East-West Traffic** (Inter-service communication)

- Service mesh for secure service-to-service communication
- Network policies for pod-to-pod communication
- Internal DNS resolution via CoreDNS

North-South Traffic (External access)

- Application Load Balancer for HTTPS termination
- Network Load Balancer for high-performance TCP/UDP
- WAF integration for application-layer security

Resource Optimization by Architecture

Performance Characteristics **Private Single AZ**

- **GPU Utilization**: 92% (same as all architectures)
- **CPU Utilization**: 87% (optimized for single-zone placement)
- **RAM Utilization**: 85% (efficient resource allocation)
- **Network Latency**: Lowest (single AZ, no cross-AZ traffic)

Public Multi-AZ

- **GPU Utilization**: 92% (maintained across zones)
- **CPU Utilization**: 87% (distributed load balancing)
- **RAM Utilization**: 85% (cross-AZ optimization)
- **Network Latency**: Slightly higher (cross-AZ communication)

Public Single AZ

- **GPU Utilization**: 92% (full optimization maintained)
- **CPU Utilization**: 87% (simplified networking overhead)
- **RAM Utilization**: 85% (single-zone efficiency)
- **Network Latency**: Lowest (direct internet, single AZ)

Deployment Planning Worksheet

Architecture Selection Checklist

Security Requirements:

- ✅ Compliance requirements (SOC2, FedRAMP, etc.)
- ✅ Network isolation needs
- ✅ Data residency requirements
- ✅ Access control complexity

Availability Requirements:

- ✅ Acceptable downtime tolerance
- ✅ Multi-zone fault tolerance needs
- ✅ Disaster recovery requirements
- ✅ Maintenance window flexibility

Cost Constraints:

- ✅ Infrastructure budget limits
- ✅ Development vs. production environment
- ✅ Temporary vs. long-term deployment
- ✅ Cost optimization priority level

Performance Requirements:

- ✅ User count and concurrency
- ✅ CPU workload intensity
- ✅ Network latency sensitivity
- ✅ Storage performance needs

Recommended Architecture by Use Case

Use Case	Recommended Architecture	Rationale
Production Enterprise	Public Multi-AZ	High availability, fault tolerance
Regulated Industries	Private Single AZ	Security, compliance, network isolation
Development Teams	Public Single AZ	Cost efficiency, rapid deployment
Proof of Concept	Public Single AZ	Quick setup, minimal complexity
Hybrid Cloud	Private Single AZ	Integration with on-premises
Global Deployment	Public Multi-AZ	Geographic distribution, redundancy

Implementation Resources

EKSCTL Configuration Templates

Each architecture can be adjusted in this EKSCTL configuration files:

- **EKS YAML**: Juno EKS Deployment Config

Deployment Guides

- EKS Deployment Guide - Step-by-step deployment instructions
- EKS Prerequisites - Planning and preparation requirements

Support Resources

- **Architecture Consultation & Technical Support**: support@juno-innovations.com
- **Community Support**: Discord Server for architecture discussions

---

Note: All architectures deliver Orion's signature resource optimization performance. Architecture choice should be based on security, availability, and cost requirements rather than performance considerations.

Disclaimer: All cost estimates and deployment times are based on current AWS pricing and may vary by region and usage patterns. Contact our sales team for precise pricing and customized cost analysis based on your specific requirements.

2.7.3 Self-Managed

Nodes Setup

DEBIAN 12 (BOOKWORM) SETUP GUIDE

Overview

Debian 12 (Bookworm) is Juno's preferred distribution for on-premises installations due to its stability, reliability, and straightforward setup process.

Base Node Configuration

System Updates

First, ensure your system is fully up to date:

```
sudo apt update
sudo apt upgrade -y
```

Swap Configuration

Disable swap for optimal Kubernetes performance:

```
sudo swapoff -a
```

To make this change persistent across reboots, comment out any swap entries in `/etc/fstab`:

```
sudo sed -i '/swap/s/^/# /' /etc/fstab
```

Docker Installation

Docker installation includes containerd and other necessary packages for Kubernetes, with pre-configured container runtimes:

```
sudo apt install curl -y
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

Enable Docker to start on boot:

```
sudo systemctl enable docker
```

Recommended Packages

These additional packages support various storage backends and enhance functionality:

```
sudo apt install open-iscsi nfs-common -y
```

GPU Configuration

Follow these steps to prepare your system for GPU-enabled workloads and rendering nodes.

Update System for GPU Support

Ensure your system is fully up to date:

```
sudo apt update
sudo apt upgrade -y
```

Verify GPU Hardware Detection

Verify that your GPU hardware is properly recognized:

```
ls -la /dev/ | grep dri
```

Video Device Mounted

If this passes, you should see the following output:

```
drwxr-xr-x  3 root root         80 Jan 30 14:03 dri
```

No Output

If you do not see this output, the kernel is not recognizing the GPU. Normally this is due to the kernel not having the correct firmware or the kernel needs to be updated. Please refer to the Debian Wiki for more information.

NVIDIA Driver Installation

To install the NVIDIA drivers, follow these steps. You can also reference the Debian NVIDIA Wiki directly.

1. Enable Non-Free Repositories

First, enable the necessary repositories for NVIDIA drivers:

```
sudo su
echo "deb http://deb.debian.org/debian/ bookworm main contrib non-free non-free-firmware" >> /etc/apt/sources.list
```

Target Translations

If you get the following error:

```
W: Target Translations (non-free-firmware/i18n/Translation-en) is configured multiple times in /etc/apt/sources.list:1 and /etc/apt/sources.list:12
```

Remove the `deb http://deb.debian.org/debian/ bookworm main non-free-firmware` duplicate entry from the `/etc/apt/sources.list` file.

2. Install NVIDIA Drivers

Install the NVIDIA drivers and required dependencies:

```
sudo apt update
sudo apt install -y nvidia-driver firmware-misc-nonfree linux-headers-$(uname -r)
sudo reboot
```

3. Verify Driver Installation

After the system restarts, verify that the drivers are installed correctly:

```
nvidia-smi
```

Drivers Installed

If this passes, you should see the following output but for your GPU model:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 535.216.01             Driver Version: 535.216.01   CUDA Version: 12.2     |
|-------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                     |                      |               MIG M. |
|=====================================+======================+======================|
|   0  NVIDIA GeForce RTX 2080        On  | 00000000:00:10.0 Off |                  N/A |
|  0%   33C    P8              13W / 215W |      1MiB /  8192MiB |      0%      Default |
|                                     |                      |                  N/A |
+-------------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Confirm GPU device mounting:

```
ls -la /dev/dri/
```

Video Device Mounted Properly With NVIDIA Drivers

If this passes, you should see something similar to the following output with your cards and a render device:

```
total 0
drwxr-xr-x  3 root root        120 Jan 30 18:12 .
drwxr-xr-x 18 root root       3.4K Jan 30 18:13 ..
drwxr-xr-x  2 root root        100 Jan 30 18:12 by-path
crw-rw----  1 root video  226,   0 Jan 30 18:12 card0
crw-rw----  1 root video  226,   1 Jan 30 18:12 card1
crw-rw----  1 root render 226, 128 Jan 30 18:12 renderD128
```

NVIDIA Container Toolkit Installation

Set up the NVIDIA Container Toolkit to enable GPU access for containers. This will allow us to run GPU enabled containers which is then provided as a runtime in kubernetes via the CRI.

1. Add NVIDIA Container Toolkit Repository

```
curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --dearmor -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg \
  && curl -s -L https://nvidia.github.io/libnvidia-container/stable/deb/nvidia-container-toolkit.list | \
    sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg] https://#g' | \
    sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list
```

2. Install NVIDIA Container Toolkit

```
sudo apt update
sudo apt install -y nvidia-container-toolkit
```

3. Configure Runtime Environments

For containerd (required for Kubernetes):

```
sudo nvidia-ctk runtime configure --runtime=containerd
sudo systemctl restart containerd
```

If Docker is installed:

```
sudo nvidia-ctk runtime configure --runtime=docker
sudo systemctl restart docker
```

4. Verify Container Toolkit Installation (Optional)

To confirm the NVIDIA Container Toolkit is working correctly:

```
sudo docker run --rm --runtime=nvidia --gpus all ubuntu nvidia-smi
```

Testing the NVIDIA Container Toolkit

If you would like to test the NVIDIA Container Toolkit, you can run the following commands. This is not needed for the installation but is a good way to verify that the toolkit is installed correctly.

```
sudo docker run --rm --runtime=nvidia --gpus all ubuntu nvidia-smi
```

Working Toolkit

If you see the output of `nvidia-smi` then the toolkit is installed correctly.

```
docker run --rm --runtime=nvidia --gpus all ubuntu nvidia-smi
INFO[0000] Loading config from /etc/docker/daemon.json
INFO[0000] Wrote updated config to /etc/docker/daemon.json
INFO[0000] It is recommended that docker daemon be restarted.
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
de44b265507a: Pull complete
Digest: sha256:80dd3c3b9c6cecb9f1667e9290b3bc61b78c2678c02cbdae5f0fea92cc6734ab
Status: Downloaded newer image for ubuntu:latest
Thu Jan 30 23:44:20 2025
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.216.01             Driver Version: 535.216.01   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA GeForce RTX 2080        On  | 00000000:00:10.0 Off |                  N/A |
|  0%   30C    P8              12W / 215W |      1MiB /  8192MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------------------------------------------------------+
```

Next Steps

Once your nodes meet the requirements, you can follow an install method of choice found in our Cluster Deployment Overview

For further assistance, contact Juno Support.

ROCKY LINUX 9.5 SETUP GUIDE

Overview

Rocky Linux is an excellent choice for base nodes in your infrastructure due to its stability, long-term support, and enterprise compatibility.

Base Node Configuration

System Update

Begin with a system update to ensure all packages are current:

```
sudo dnf update -y
```

Docker Installation

While Docker is not strictly required for Kubernetes, installing it provides containerd and other necessary packages for Kubernetes. The installation process also configures container runtimes automatically.

```
# Add Docker repository
sudo dnf config-manager --add-repo https://download.docker.com/linux/rhel/docker-ce.repo

# Install Docker packages
sudo dnf -y install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

# Enable and start Docker service
sudo systemctl --now enable docker
```

Recommended Packages

These additional utilities enhance functionality for networking, storage, and system management:

```
sudo dnf install nfs-utils iscsi-initiator-utils curl rsync -y
```

GPU Configuration

Follow these steps to prepare your system for GPU-enabled workloads and rendering nodes.

Update System for GPU Support

Ensure your system has the latest updates before proceeding with GPU setup:

```
sudo dnf update -y
```

Verify GPU Hardware Detection

Confirm that your system recognizes the GPU hardware:

```
ls -la /dev/ | grep dri
```

Video Device Mounted

If this passes, you should see the following output:

```
drwxr-xr-x  3 root root         80 Jan 30 14:03 dri
```

No Output

If you do not see this output, the kernel is not recognizing the GPU. Normally this is due to the kernel not having the correct firmware or the kernel needs to be updated. Please refer to the Rocky Linux Kernel Documentation for more information.

NVIDIA Driver Installation

To install the NVIDIA drivers, run the following commands. You can also reference the Rocky Nvidia Wiki directly.

1. Disable Nouveau Driver

First, disable the open-source Nouveau driver that may interfere with NVIDIA drivers:

```
sudo grubby --args="nouveau.modeset=0 rd.driver.blacklist=nouveau" --update-kernel=ALL
```

2. Install Required Dependencies

```
# Add EPEL repository
sudo dnf install epel-release -y

# Install development tools package group
sudo dnf groupinstall "Development Tools" -y

# Install kernel development packages
sudo dnf install kernel-devel -y
sudo dnf install dkms -y
```

3. Add NVIDIA Repository and Install Additional Dependencies

```
sudo dnf config-manager --add-repo http://developer.download.nvidia.com/compute/cuda/repos/rhel9/$(uname -i)/cuda-rhel9.repo
sudo dnf install kernel-headers kernel-devel tar bzip2 make automake gcc gcc-c++ pciutils elfutils-libelf-devel libglvnd-opengl libglvnd-glx libglvnd-devel a
cpid pkgconf dkms -y
```

4. Install NVIDIA Driver

Check available driver versions:

```
dnf module list nvidia-driver
```

Choose one of these installation options:

Option 2: Install the latest driver (recommended)

```
sudo dnf module install nvidia-driver:latest-dkms -y
```

Option 1: Install a specific driver version

```
sudo dnf module install nvidia-driver:<version>-dkms -y
```

5. Complete Driver Installation

After installing the driver, register the kernel modules and reboot:

```
sudo dkms autoinstall
sudo reboot
```

6. Verify Driver Installation

After the system restarts, verify that the drivers are installed correctly:

```
nvidia-smi
```

Drivers Installed

If this passes, you should see the following output but for your GPU model:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 535.216.01             Driver Version: 535.216.01   CUDA Version: 12.2     |
|-------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                     |                      |               MIG M. |
|=====================================+======================+======================|
|   0  NVIDIA GeForce RTX 2080        On  | 00000000:00:10.0 Off |                  N/A |
|  0%   33C    P8              13W / 215W |      1MiB /  8192MiB |      0%      Default |
|                                     |                      |                  N/A |
+-------------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Confirm that the GPU devices are properly mounted:

```
ls -la /dev/dri/
```

Video Device Mounted Properly With NVIDIA Drivers

If this passes, you should see something similar to the following output with your cards and a render device:

```
total 0
drwxr-xr-x  3 root root        120 Jan 30 18:12 .
drwxr-xr-x 18 root root       3.4K Jan 30 18:13 ..
drwxr-xr-x  2 root root        100 Jan 30 18:12 by-path
crw-rw----  1 root video  226,   0 Jan 30 18:12 card0
crw-rw----  1 root video  226,   1 Jan 30 18:12 card1
crw-rw----  1 root render 226, 128 Jan 30 18:12 renderD128
```

NVIDIA Container Toolkit Installation

Now we need to configure the NVIDIA Container Toolkit. This will allow us to run GPU enabled containers which is then provided as a runtime in kubernetes via the CRI.

You can also reference the Rocky Nvidia Wiki directly.

1. Add NVIDIA Container Toolkit Repository

```
curl -s -L https://nvidia.github.io/libnvidia-container/stable/rpm/nvidia-container-toolkit.repo | \
  sudo tee /etc/yum.repos.d/nvidia-container-toolkit.repo
```

2. Install NVIDIA Container Toolkit

```
sudo dnf update
sudo dnf install -y nvidia-container-toolkit
```

3. Configure Runtime Environments

For containerd (required for Kubernetes):

```
sudo nvidia-ctk runtime configure --runtime=containerd
sudo systemctl restart containerd
```

If Docker is installed:

```
sudo nvidia-ctk runtime configure --runtime=docker
sudo systemctl restart docker
```

4. Verify Container Toolkit Installation (Optional)

To confirm the NVIDIA Container Toolkit is working correctly:

```
sudo docker run --rm --runtime=nvidia --gpus all ubuntu nvidia-smi
```

Testing the NVIDIA Container Toolkit

If you would like to test the NVIDIA Container Toolkit, you can run the following commands. This is not needed for the installation but is a good way to verify that the toolkit is installed correctly.

```
sudo docker run --rm --runtime=nvidia --gpus all ubuntu nvidia-smi
```

Working Toolkit

If you see the output of `nvidia-smi` then the toolkit is installed correctly.

```
docker run --rm --runtime=nvidia --gpus all ubuntu nvidia-smi
INFO[0000] Loading config from /etc/docker/daemon.json
INFO[0000] Wrote updated config to /etc/docker/daemon.json
INFO[0000] It is recommended that docker daemon be restarted.
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
de44b265507a: Pull complete
Digest: sha256:80dd3c3b9c6cecb9f1667e9290b3bc61b78c2678c02cbdae5f0fea92cc6734ab
Status: Downloaded newer image for ubuntu:latest
Thu Jan 30 23:44:20 2025
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.216.01             Driver Version: 535.216.01   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA GeForce RTX 2080        On  | 00000000:00:10.0 Off |                  N/A |
|  0%   30C    P8              12W / 215W |      1MiB /  8192MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------------------------------------------------------+
```

Firewall Configuration

No Route to Host

When deploying certain kubernetes distributions, you may see the following error in the container logs:

```
Error: No route to host
```

In many cases, this is due to the firewall blocking the connection. You can disable the firewall to check if that is the issue:

```
sudo systemctl stop firewalld
sudo systemctl disable firewalld
```

It's recommended that firewalld is either disabled or properly configured with appropriate whitelisting for your Kubernetes and container networking. However, detailed firewall configuration is beyond the scope of this documentation as each network environment is different.

SELinux Configuration

If disabling the firewall doesn't resolve connectivity issues, you may also need to disable SELinux:

```
# Disable SELinux
sudo setenforce 0
sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
```

This changes SELinux from enforcing mode to permissive mode, which logs policy violations but doesn't enforce them. The second command makes this change permanent across reboots. Note that this may have security implications depending on your environment.

Next Steps

Once your nodes meet the requirements, you can follow an install method of choice found in our Cluster Deployment Overview

For further assistance, contact Juno Support.

UBUNTU 22.04 SETUP GUIDE

Overview

Ubuntu 22.04 LTS (Jammy Jellyfish) is an excellent choice for Orion nodes due to its stability, long-term support, and extensive hardware compatibility.

Base Node Configuration

System Update

Begin with a system update to ensure all packages are current:

```
sudo apt update
sudo apt upgrade -y
```

Swap Configuration

Disable swap for optimal Kubernetes performance:

```
sudo swapoff -a
```

To make this change persistent across reboots, comment out any swap entries in `/etc/fstab`:

```
sudo sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
```

Docker Installation

Docker installation includes containerd and other packages required for Kubernetes, with pre-configured container runtimes:

```
sudo apt install curl -y
curl -fsSL https://get.docker.com -o get-docker.sh
sh get-docker.sh
```

Recommended Packages

These additional utilities enhance functionality for networking, storage, and system management:

```
sudo apt install open-iscsi nfs-common -y
```

GPU Configuration

Follow these steps to prepare your system for GPU-enabled workloads and rendering nodes.

Update System for GPU Support

Ensure your system has the latest updates before proceeding with GPU setup:

```
sudo apt update
sudo apt upgrade -y
```

Verify GPU Hardware Detection

Confirm that your system recognizes the GPU hardware:

```
ls -la /dev/ | grep dri
```

Video Device Mounted

If this passes, you should see the following output:

```
drwxr-xr-x  3 root root         80 Jan 30 14:03 dri
```

No Output

If you do not see this output, the kernel is not recognizing the GPU. Normally this is due to the kernel not having the correct firmware or the kernel needs to be updated.

Kernel Update

If you are updating the kernel, make sure you have a backup of your system and that you are prepared for a reboot.

To update the firmware and kernel, run the following commands:

```
sudo apt install --install-recommends linux-generic-hwe-22.04 -y
sudo reboot
```

NVIDIA Driver Installation

To install the NVIDIA drivers, follow these steps. You can also reference the Ubuntu NVIDIA Documentation directly.

1. Install Ubuntu Drivers Helper

Use Ubuntu's built-in driver detection and installation tool:

```
sudo apt install ubuntu-drivers-common -y
sudo ubuntu-drivers autoinstall
sudo reboot now
```

2. Verify Driver Installation

After the system restarts, verify that the drivers are installed correctly:

```
nvidia-smi
```

Drivers Installed

If this passes, you should see the following output but for your GPU model:

```
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.216.01             Driver Version: 535.216.01   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA GeForce RTX 2080        On  | 00000000:00:10.0 Off |                  N/A |
|  0%   33C    P8              13W / 215W |      1MiB /  8192MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

Confirm GPU device mounting:

```
ls -la /dev/dri/
```

Video Device Mounted Properly With NVIDIA Drivers

If this passes, you should see something similar to the following output with your cards and a render device:

```
total 0
drwxr-xr-x  3 root root        120 Jan 30 18:12 .
drwxr-xr-x 18 root root       3.4K Jan 30 18:13 ..
drwxr-xr-x  2 root root        100 Jan 30 18:12 by-path
crw-rw----  1 root video  226,   0 Jan 30 18:12 card0
crw-rw----  1 root video  226,   1 Jan 30 18:12 card1
crw-rw----  1 root render 226, 128 Jan 30 18:12 renderD128
```

NVIDIA Container Toolkit Installation

Now we need to configure the NVIDIA Container Toolkit. This will allow us to run GPU enabled containers which is then provided as a runtime in kubernetes via the CRI.

1. Add NVIDIA Container Toolkit Repository

```
curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --dearmor -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg \
  && curl -s -L https://nvidia.github.io/libnvidia-container/stable/deb/nvidia-container-toolkit.list | \
    sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg] https://#g' | \
    sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list
```

2. Install NVIDIA Container Toolkit

```
sudo apt update
sudo apt install nvidia-container-toolkit -y
```

3. Configure Runtime Environments

For containerd (required for Kubernetes):

```
sudo nvidia-ctk runtime configure --runtime=containerd
sudo systemctl restart containerd
```

If Docker is installed:

```
sudo nvidia-ctk runtime configure --runtime=docker
sudo systemctl restart docker
```

4. Verify Container Toolkit Installation (Optional)

To confirm the NVIDIA Container Toolkit is working correctly:

```
sudo docker run --rm --runtime=nvidia --gpus all ubuntu nvidia-smi
```

Testing the NVIDIA Container Toolkit

If you would like to test the NVIDIA Container Toolkit, you can run the following commands. This is not needed for the installation but is a good way to verify that the toolkit is installed correctly.

```
sudo docker run --rm --runtime=nvidia --gpus all ubuntu nvidia-smi
```

Working Toolkit

If you see the output of `nvidia-smi` then the toolkit is installed correctly.

```
docker run --rm --runtime=nvidia --gpus all ubuntu nvidia-smi
INFO[0000] Loading config from /etc/docker/daemon.json
INFO[0000] Wrote updated config to /etc/docker/daemon.json
INFO[0000] It is recommended that docker daemon be restarted.
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
de44b265507a: Pull complete
Digest: sha256:80dd3c3b9c6cecb9f1667e9290b3bc61b78c2678c02cbdae5f0fea92cc6734ab
Status: Downloaded newer image for ubuntu:latest
Thu Jan 30 23:44:20 2025
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.216.01             Driver Version: 535.216.01   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA GeForce RTX 2080        On  | 00000000:00:10.0 Off |                  N/A |
|  0%   30C    P8              12W / 215W |      1MiB /  8192MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------------------------------------------------------+
```

Next Steps

Once your nodes meet the requirements, you can follow an install method of choice found in our Cluster Deployment Overview

For further assistance, contact Juno Support.

Ansible deployment

OVERVIEW

We maintain a set of Ansible Playbooks you can use to perform the deployments. They are the recommended method for bigger installations where you'd like to automate the process

We support both online & airgapped deployments within those.

For advanced Ansible users, if you'd like to take the existing role and include it in your existing deployment setup, you can find it here. For just getting up & running, you can skip that and refer to the documentation found on the Playbook Repository.

2.7.4 Air-Gapped

Airgapped installation

This page will guide you through the process of deploying Orion in an airgapped environment with no internet access.

After following the below instructions you will have a functioning deployment of Orion, enabling you to launch workloads on your on-prem infrastructure.

If you need to handle a big number of nodes or would like to automate the process, we maintain a set of Ansible Playbooks you can use to perform the deployments. We support both online & airgapped deployments within those.

The playbook repository contains instructions on how to get running with them. If you prefer to perform the installation without Ansible, continue with the guide below.

REQUIREMENTS

Before you get started, you must have the following available:

- A Kubernetes cluster or hosts to provision it on. We heavily recommend our Quick Start Helper script, which will bootstrap an airgapped cluster for you with correct settings.
- An OCI container image registry available and accessible from your cluster. Common choices are:
- Harbor,
- Private ECR Endpoints,
- Gitea or GitLab's builtin registry
- Your nodes need access to your Git host, such as GitLab, GitHub Enterprise, Gitea, etc.
- The appropriate charts & images forked into your image registry - see Requirements - what you need to mirror/fork

REQUIREMENTS - WHAT YOU NEED TO MIRROR/FORK

Images

Before deploying, make sure you have all the images listed in our Image Guide. Those must be available from within your internal image registry, either as a copy or a proxied project (eg. Harbor's proxycache).

Access to your image registry

In most cases, you have 3 ways to make your images available:

1) Configure your runtime to perform all image pulls through your internal registry.

Most modern runtimes enable you to perform a "rewrite" of image pulls. That means that attempting to pull an image from docker.io will really go to your internal server, however Kubernetes must not be aware of this.

We recommend this method wherever possible - it is most reliable and requires no extra configuration changes for the applications themselves. When using our On-Prem helper script, you will be automatically prompted for the correct settings. The script will then take care of the setup for you.

We will focus on that option in this guide. Note this is also possible on many custom K8s distributions - while the guide won't cover them, you can find helpful details in your distro-specific documentation. Your registry can be both authenticated or unauthenticated - both options work.

- For k3s-based deployments, you can find a reference here
- we configure this automatically for you if you use our helper script or Ansible deployments.
- For CRI-O based custom deployments, RedHat provides an excellent guide.
- For ContainerD based custom deployments, see the upstream docs

2) Using an unauthenticated, RO registry.

Assuming all you do is fork upstream images, an unauthenticated, RO proxy (such as one Harbor can provide) is a valid option. This still requires extra work - you'll need to adjust imagePullSecrets.

3) Populating your own imagePullSecrets

This takes the most work & maintenance from you, hence we recommend using rewrites instead when possible.

Helpful references

While this guide will focus on redirecting all image pulls to your registry via the runtime, in case that does not suit your environment, you'll need to set up and configure imagePullSecrets for both Argo and our deployments.

You can find more details on how those work here The imagePullSecrets can then be passed in to respective charts, starting with argo and continuing with Juno-Bootstrap.

This is not necessary and is much simplified when using the Quick Start Helper script.

Helm charts

To get Orion running in an airgapped environment, you must fork all Helm charts it depends on.

You can fork them either as git repositories or using OCI storage, as per Helm Documentation

We recommend Git repositories for:

- When you want the flexibility to quickly customize
- When you want to move fast
- When you are new to airgapped K8s deployments

OCI can be very fitting when you are looking to keep a more stringent & auditable release process for the charts and would like to work off of your established workflows. Many state-of-the-art approaches such as image signing can be reused for Helm, if your requirements dictate so.

Both approaches will work to deploy Orion.

Below you can find the charts that are necessary to get running:

- Orion Deployment - version/tag v2.0.1
- ingress-nginx - version 4.12.1

While these are not strictly necessary for installation, they are recommended for ease of use during workload creation

- Official Terra Plugins - main branch
- Genesis Deployment - version/tag v2.0.3

PERFORMING THE INSTALLATION - HELPER SCRIPT

Once you meet all of the above prerequisites, you are ready to go! From here, you can jump back into the Quick Start Guide and continue on with it.

If you are running a custom deployment and passing in the imagePullSecrets manually, refer to Juno-Bootstrap and its example values: https://github.com/juno-fx/Juno-Bootstrap/blob/main/values.yaml To get running, you would then install the deployment chart with your adjusted values.

Install script

The install script be downloaded and not immediately ran by using the following command which can then be transferred to your offline server.

```
curl -sL "$(curl -s https://api.github.com/repos/juno-fx/Juno-Bootstrap/releases/latest | grep browser_download_url | grep orion-install-helper | cut -d '"'
-f 4)" > orion-install-helper && chmod +x orion-install-helper
```

Images

DockerHub Availability

Juno container images are publicly available on DockerHub for direct access. You can pull images directly from our DockerHub repository without any special requests or approval process. To launch Genesis, you will need to provide the license key during the deployment process.

Air-Gapped Environments

If you are deploying in an air-gapped environment, you will need to follow the manual image deployment steps below to download and load images into your private registry.

Below you can find images that you will need to ingest for an airgapped deployment.

It tracks latest releases of our charts:

- Genesis
- Orion

The deployment also leverages a few upstream components:

- ArgoCD
- ingress-nginx

If your cluster has those pre-deployed or if you prefer to manage those yourself, it is likely they will work out of the box. The above versions are what we actively test our air-gapped deployment against and what we recommend - they are not technically a hard requirement.

Warning

These container images were correct as of **May 2026 on release branch** `genesis4.0.1-orion4.0.0`, see the latest deployment documentation for the most up to date images

Image Name	Component	Image Version
docker.io/junoinnovations/genesis	Juno Genesis	v5.0.0
docker.io/junoinnovations/titan	Juno Genesis	v2.1.2
docker.io/junoinnovations/terra	Juno Genesis	v2.1.1
docker.io/junoinnovations/hubble	Juno Orion	v6.0.0
docker.io/junoinnovations/kuiper	Juno Orion	v4.1.3
docker.io/junoinnovations/rhea	Juno system wide	v1.2.3
registry.k8s.io/ingress-nginx/controller	ingress-nginx	v1.15.0
registry.k8s.io/ingress-nginx/kube-webhook-certgen	ingress-nginx	v1.6.8
quay.io/argoproj/argocd	ArgoCD	v3.0.9
ghcr.io/dexidp/dex	ArgoCD	v2.41.1
docker.io/library/redis	ArgoCD	7.2.7-alpine

When using our workload images, we recommend you pull the latest stable tag of Helios. The workload image is not included in the list above, since there are multiple choices - we ship multiple distros and flavors that serve different industry-specific use cases.

Here are the provided links for 3rd party air-gapped guides:

- Nvidia GPU Operator

Setting up a basic images repository

To setup a basic image registry for storing and pulling the above list of containers we can use a variety of software, the below guide makes use of the Registry container

1. Pull and run the registry container specifying a local path for persistent storage

a.
```
docker run -d \
  -p 5000:5000 \
  --restart=always \
  --name registry \
  -v /mnt/registry:/var/lib/registry \
  registry:3
```

2. Pull an image locally (I'll use genesis as an example here but this will need to be repeated for the full list of images above)

a. `docker pull docker.io/junoinnovations/genesis:v5.0.0`

3. Tag the downloaded image pointing to the local registry, change localhost to the IP/hostname of the deployed registry container if not on the same host.

a. `docker tag docker.io/junoinnovations/genesis:v5.0.0 localhost:5000/juno-innovations/genesis:v5.0.0`

4. Push the newly tagged container to the local registry

a. `docker push localhost:5000/juno-innovations/genesis:v5.0.0`

5. (Optional) Test it has worked by attempting to pull the image from the registry

a. `docker pull localhost:5000/juno-innovations/genesis:v5.0.0`

Airgapped installation - Example

> **Note**
>
> This guide is designed as an example it is not recommended for a production environment

> **Warning**
>
> This procedure and container images were correct as of **April 2026 on release branch** `genesis3.0.0-orion3.0.0`, see the latest deployment documentation for the most up to date images

This page is a full detailed guide on how to setup Orion in an airgapped environment from scratch.

REQUIREMENTS

Before you get started, you must have the following available:

- A host with internet access to host a container register
- A git server to host helm charts
- A host to install the Juno Orion software on

GATHERING EXTERNAL RESOURCES

Prepare container images

This section will details the steps required to download and store the relevant container images required by Orion and its underlying software. Its worth noting that the Orion host will need direct access to the registry host

1. Start a docker registry to store the downloaded container images in

```
docker run -d \
  -p 5000:5000 \
  --restart=always \
  --name registry \
  -v /mnt/registry:/var/lib/registry \
  registry:3
```

2. Once the above is up and running, you can download and push images using the following bash script, save the contents to a file and run it from a server that has both access to the docker registry and the internet. You will need to adjust the value of LOCAL_REGISTRY to the address of your deployed server. The script makes use of a package called `skopeo` this is required to preserve sha values required by some deployments.

   The below script should be saved into a file e.g. `load_scripts.sh` and then be made executable with `chmod +x load_scripts.sh` before being run `./load_scrips.sh`. **Please** be aware the images used in the below script are used as an example, please refer to the latest deployment charts for the most up to date versions.

```
#!/bin/bash

# --- Pre-flight Check ---
if ! command -v skopeo &> /dev/null; then
    echo " Error: 'skopeo' is not installed or not in PATH."
    echo " Please install it using: sudo dnf install skopeo (Rhel) or sudo apt install skopeo (Ubuntu)"
    exit 1
fi

IMAGES=(
    # Juno images
    "docker.io/junoinnovations/genesis:v5.0.0"
    "docker.io/junoinnovations/titan:v2.1.2"
    "docker.io/junoinnovations/terra:v2.1.1"
    "docker.io/junoinnovations/hubble:v6.0.0"
    "docker.io/junoinnovations/kuiper:v4.1.3"
    "docker.io/junoinnovations/rhea:v1.2.3"

    # Nginx Ingress images
    "registry.k8s.io/ingress-nginx/controller:v1.15.0"
    "registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.6.8@sha256:d7e8257f8d8bce64b6df55f81fba92011a6a77269b3350f8b997b152af348dba"

    # ArgoCD images
```

```
    "quay.io/argoproj/argocd:v3.0.9"
    "ghcr.io/dexidp/dex:v2.41.1"
    "docker.io/library/redis:7.2.7-alpine"

    # k3s images
    "docker.io/rancher/mirrored-pause:3.6"
    "docker.io/rancher/mirrored-coredns-coredns:1.12.1"
    "docker.io/rancher/local-path-provisioner:v0.0.31"
    "docker.io/rancher/mirrored-metrics-server:v0.7.2"
    "docker.io/rancher/klipper-lb:v0.4.13"

    # (Example) Image used for LSIO webtop
    lscr.io/linuxserver/chrome:latest
)

LOCAL_REGISTRY="localhost:5000"

for IMAGE in "${IMAGES[@]}"; do
    echo "------------------------------------------------------"
    echo "Processing: $IMAGE"

    if [[ "$IMAGE" == *"@sha256:"* ]]; then
        # We need to pull the sha but push the tag
        SRC_BASE=$(echo "$IMAGE" | cut -d':' -f1)
        DIGEST=$(echo "$IMAGE" | cut -d'@' -f2)
        SRC_REF="$SRC_BASE@$DIGEST"
        DEST_REF=$(echo "$IMAGE" | sed 's/@sha256:.*//')
    else
        # Standard tag-only image
        SRC_REF="$IMAGE"
        DEST_REF="$IMAGE"
    fi

    DEST_PATH=$(echo "$DEST_REF" | cut -d'/' -f2-)

    SRC_IMAGE="docker://$SRC_REF"
    DEST_IMAGE="docker://$LOCAL_REGISTRY/$DEST_PATH"

    echo "Source:      $SRC_IMAGE"
    echo "Destination: $DEST_IMAGE"

    skopeo copy \
        --dest-tls-verify=false \
        --multi-arch all \
        --preserve-digests \
        "$SRC_IMAGE" \
        "$DEST_IMAGE"

    if [ $? -eq 0 ]; then
        echo "Successfully synced!"
    else
        echo "ERROR: Failed to sync $IMAGE"
    fi
    echo ""
done
```

Prepare Git repos

The following repositories are required to be mirrored in a local git host somewhere available to your Orion install. The exact commands will be different depending on how your git server is setup, this script is provided but may not be relevant to your setup.

1. "https://github.com/juno-fx/Orion-Deployment"
2. "https://github.com/kubernetes/ingress-nginx"
3. "https://github.com/juno-fx/Terra-Official-Plugins"
4. "https://github.com/juno-fx/Genesis-Deployment"

```
#!/usr/bin/env bash
# Usage: ./mirror_repos.sh <TARGET_BASE_URL>
# e.g.   ./mirror_repos.sh git@gitlab.example.com:mygroup

set -euo pipefail

TARGET="${1:?Usage: $0 <target_base_url>}"
TARGET="${TARGET%/}"

REPOS=(
    "https://github.com/juno-fx/Orion-Deployment"
    "https://github.com/kubernetes/ingress-nginx"
    "https://github.com/juno-fx/Terra-Official-Plugins"
    "https://github.com/juno-fx/Genesis-Deployment"
)

WORK_DIR="/tmp/git-mirror"
mkdir -p "$WORK_DIR"
```

```
for URL in "${REPOS[@]}"; do
    NAME="$(basename "$URL")"
    DIR="$WORK_DIR/$NAME.git"

    echo ">>> $NAME"

    if [[ -d "$DIR" ]]; then
        git -C "$DIR" fetch --all --prune --tags
    else
        git clone --mirror "$URL" "$DIR"
    fi

    git -C "$DIR" push --mirror "$TARGET/$NAME.git"

    echo ""
done

echo "Done."
```

Download the Juno Oneclick installer and install wizard

Download the following and transfer them to your (future) Orion host. Take a note of the path(s) as they will be required during install

- One click installer can be downloaded from this page
- Install wizard can be downloaded by running the following command

```
curl -sL "$(curl -s https://api.github.com/repos/juno-fx/Juno-Bootstrap/releases/latest | grep browser_download_url | grep orion-install-helper | cut -d '"' -f 4)" > /tmp/orion-install-helper && chmod +x /tmp/orion-install-helper
```

No gateway configured

If your air gapped server is not configured with a network gateway then K3s may error during install as it uses this information to setup its internal networking. This can be resolved by creating a k3s config file prior to install.

File: `/etc/rancher/k3s/config.yaml` Contents:

```
node-ip: 192.168.1.2        # Your actual node IP
cluster-cidr: 10.42.0.0/16
service-cidr: 10.43.0.0/16
flannel-iface: eth0         # Specify your interface explicitly
```

RUNNING THE WIZARD

During the running of the Oneclick install you will be prompted for information regarding your setup. This guide will assist you with answering those questions but there will be some situations where the answers supplied are not relevant to you.

Run the install script copied to the server in the previous step

```
./orion-install-helper
```

The first section of the install script is concerned with collecting Installation information. Fill this out as needed

Confirm offline install

After confirming the install information is correct you will be prompted to decide if this is an offline install, here we choose `[y]`

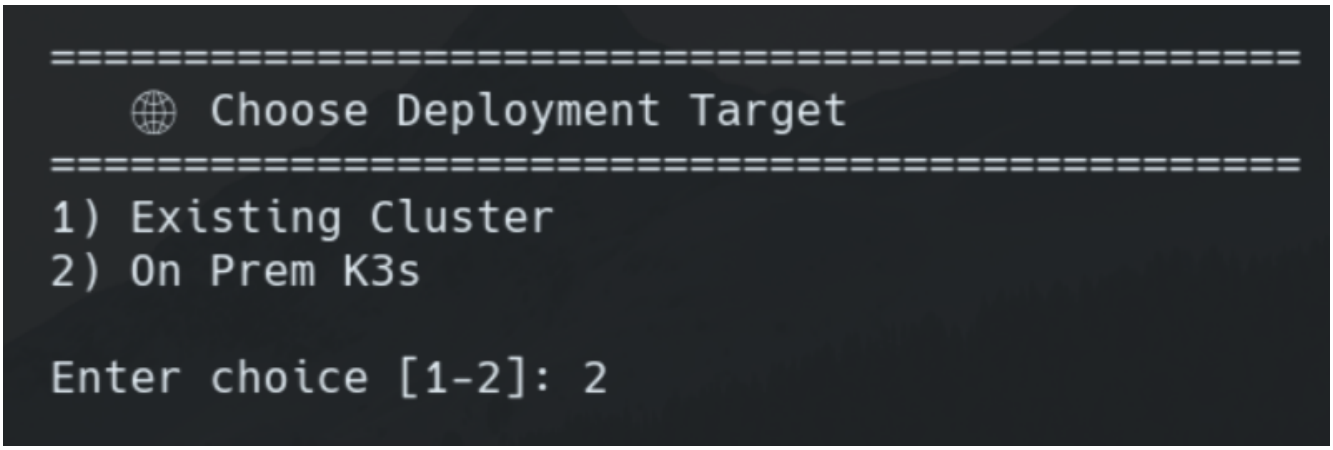
Chart URLS

You will now be asked for the chart URLs. These will be stored in your git server, copy the HTTP URL to the ingested git repo and enter it into the wizard. Confirm it is a git repo and enter the chart path as seen below.

```
Is this an offline installation? [y/N]: y
Enter Genesis chart URL [https://github.com/juno-fx/Genesis-Deployment.git]: http://some-where.com/git/Genesis-Deployment
? Is this a git repo? [y/N]: y
Enter the chart path within the repo: /
 Enter Genesis chart version [v2.0.3]: v2.0.3
Enter ingress-nginx chart URL [https://kubernetes.github.io/ingress-nginx]: http://some-where.com/git/ingress-nginx
? Is this a git repo? [y/N]: y
Enter the chart path within the repo: charts/ingress-nginx
 Enter ingress-nginx chart version [4.12.1]: helm-chart-4.15.0
Writing final .values.yaml...
.values.yaml has been created with your configuration.
```

Cluster install

In this guide we will be selecting K3s to be installed as part of the setup. Select `[2]` at the "Choose Deployment Target" section



Onceclick Installer

You will now be asked for the path to the Oneclick installer archive. Enter the full path

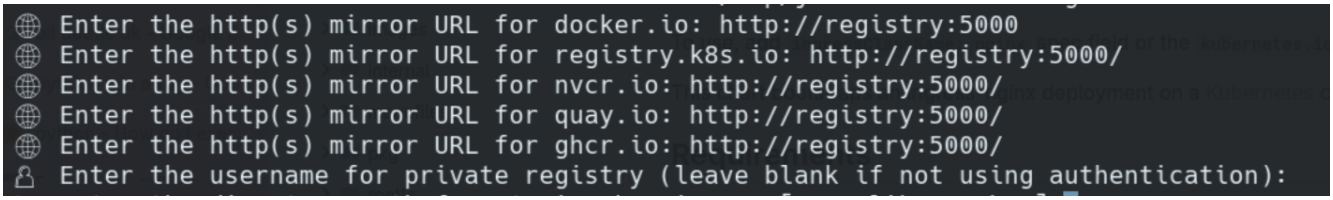
```
==============================================
   Official Juno Innovations K3s Provisioner for On-Premises
==============================================

Note: the installer can be downloaded from: https://github.com/juno-fx/K8s-Playbooks/releases
Enter the path to the oneclick archive: /tmp/juno-oneclick.tar.gz
```

> **Warning**
>
> If the path is invalid or the installer cannot access the archive you will be prompted again

Container Registry(s)

Next is the full path to the container registries. You are prompted for several but in this setup we can answer with the same URL for each one, this will be the path to the registry we setup at the beginning of this guide. By default the registry does not ask for username or password so we have left that blank here.



Ready to install

If you have a separate disk you wish to store K3s images other than root enter it in the next step, in this guide we will accept the default You will be asked if you're ready to install, when selected the application will now be installed via a built in Ansible role. Congratulations on your new Juno Orion install!

ACCESS THE UI AND SET UP WORKLOADS

Access the Orion install web interface by any browser and log in using the credentials entered into the wizard at the beginning.

Projects

In order to create workloads you will need to setup a project, we will need to point this at the locally available git repo containing the Orion chart.

1. In the Juno Orion portal, select "Projects" from the left hand menu
2. Select "CREATE PROJECT"
3. You will be presented with a form, input a name and a group ID as prompted and click "next"
4. The next window is much longer and requires more input from us

a. Change the "Orion Repository" to the locally cloned git repo

b. Set the Orion Version to the latest Orion version e.g. `v4.0.0`

5. Click "deploy" when ready

For more information on project creation see the dedicated page on Projects

Terra plugins

In order to deploy plugins we need to setup the terra plugin system, again this will make use of a previously downloaded git repo

1. In the left have side select the "App Store"
2. Select "SOURCE REPOS" from the new window
3. Click the "NEW SOURCE" button
4. We are presented with a form to populate, complete and submit as follows

a. URL should be set to the git fork completed earlier e.g. http://our-git-repo:3000/juno/terra-official-plugins

b. We suggest using "main" as the "Ref"

c. (Optional) if your git repository requires authentication then please populate the bottom two auth fields

5. Once the repository has been installed the "STORE" should be populated with all the available plugins from Terra

For more information on Terra Plugins see the dedicated page on Terra

Workload Images

The workloads created by Terra plugins will be using standard containers that will need to be ingested into your container registry.

In this example we will be setting up the LSIO Webtop package. We assume a storage mount has already been created for this step, see here for further instructions on how to set this up.

(Optional) Adding lsio as a registry

For ease of use we will need to add "lsio" as a registry source in the k3s settings, this is only necessary if your registry is using HTTP and not HTTPS If you are using the k3s cluster installed via the bootstrap wizard then run the following command replacing the endpoint value with your registry value

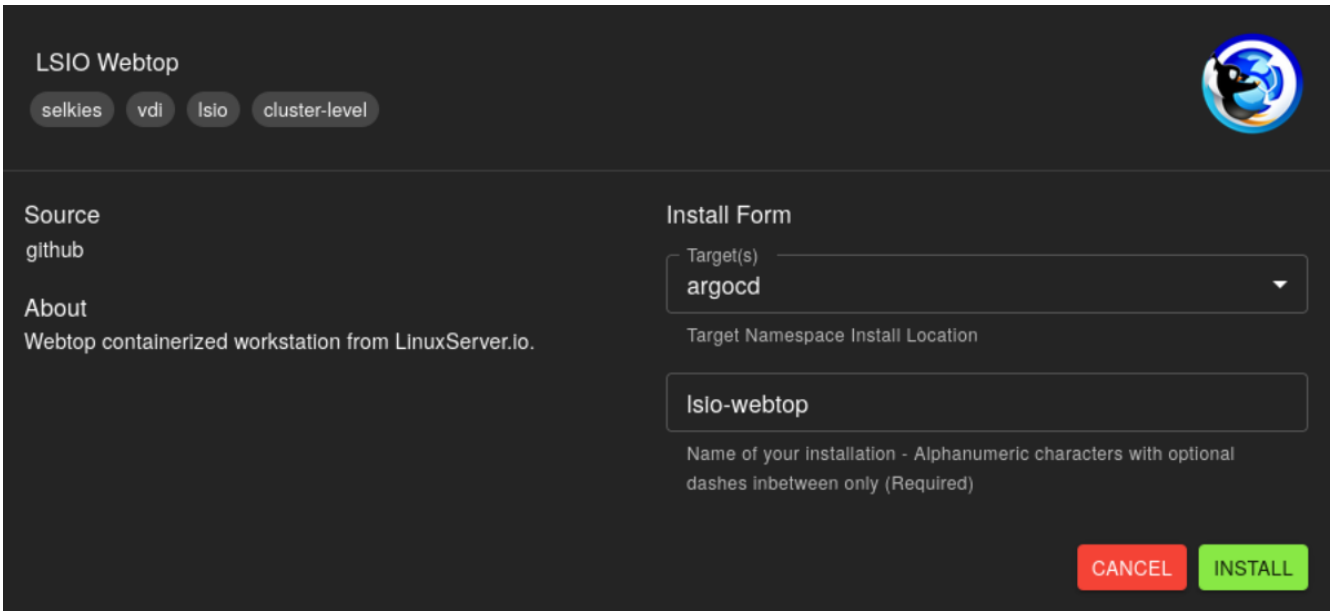
```
sudo jq '.mirrors["lscr.io"] = {"endpoint": ["http://YOUR_REGISTRY_HERE"]}' /etc/rancher/k3s/registries.yaml | sudo tee /etc/rancher/k3s/registries.yaml.tmp && s
udo mv /etc/rancher/k3s/registries.yaml.tmp /etc/rancher/k3s/registries.yaml
```

Then restart k3s systemctl restart k3s

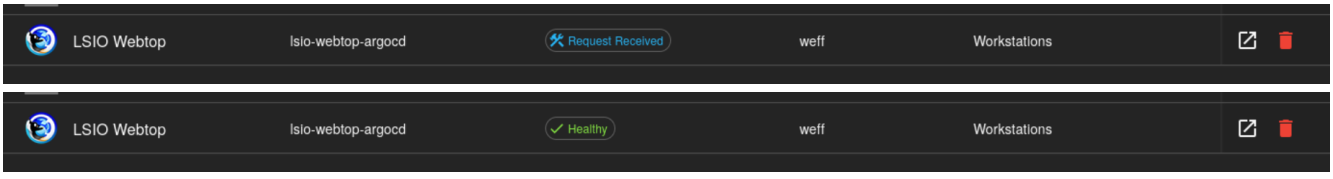
```
systemctl restart k3s
```

This will need to be repeated on all k3s nodes

From the App store locate the LSIO Webtop plugin and install it, selecting a friendly name and clicking install



Wait for the app to be ready by checking the "INSTALLS" tab



Setup the workload

1. Select the Workloads window from the left hand navigation bar and click "CREATE WORKLOAD"
2. From the Version drop down select the lsio-webtop option
3. Assign a group using the button and resulting form
4. (Optional) If your registry is using HTTPS then you can update the "registry" text box with the URL of your container registry housing the image
5. Click "Submit"

   You can now deploy the LSIO workload on your air gapped system!

For more information on workloads see the dedicated page on Workloads

3. Guides

3.1 Introduction

Welcome to Orion, we at Juno Innovations are so happy you chose to utilize our platform to help optimize your infrastructure. This guide is intended for first time users, who have just installed and deployed Orion. Here we will guide you through some of the basic first steps to getting up and running so you can harness the power of Orion.

Once installed you will be able to login to your Genesis admin dashboard. You can see our full indepth Genesis documentation here

> **Note**
>
> If you haven't already installed Orion please follow our install guide first.

3.1.1 Step 1: Setup Authorization Providers

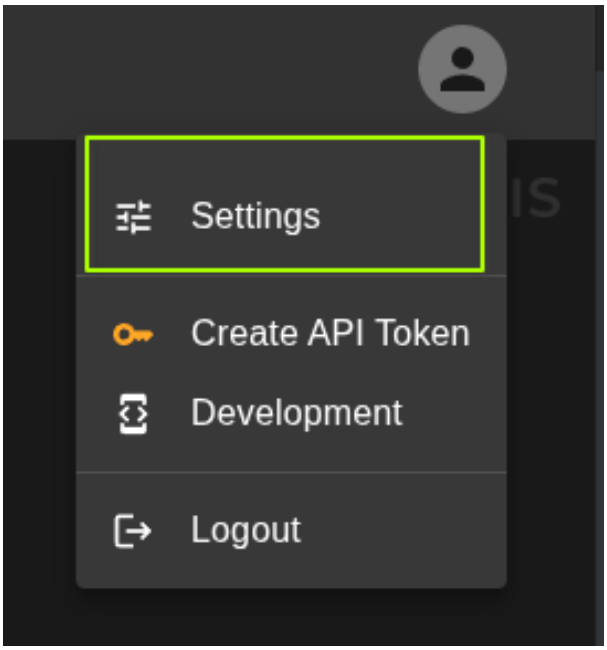
By default, via our `OneClick` installer you will only have the BASIC authorization setup. If you customized your deployment beyond the `OneClick` installer, and already setup your authorization you can skip this step.

> **Note**
>
> We recommend leaving the BASIC authorization available until you are able to confirm your authorization provider is configured and working properly. Otherwise you may risk locking yourself out of your Orion deployment. Once confirmed, we highly recommend removing the BASIC authorization.

> **Warning**
>
> While you can skip adding another more robust authorization provider, and continue using the BASIC authorization. We do not recommend the BASIC authorization for production environments. This is best used for testing, and development purposes.

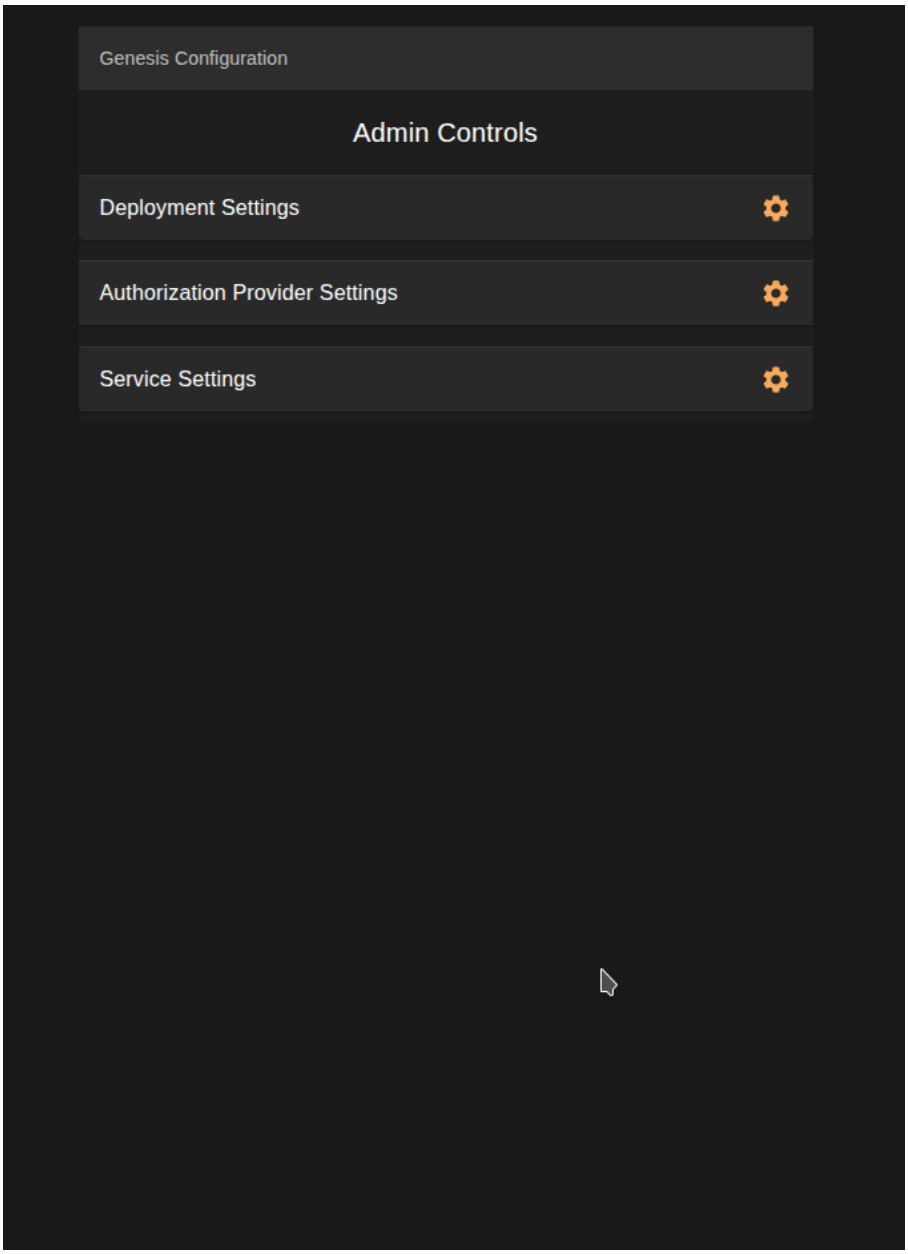
Navigate to the Admin Settings Page

Only admins will have access to the Genesis Admin settings page. You can navigate to this page via the user profile drop down in the top right corner of Genesis.



Fill in Authorization Provider Form

Expand the authorization provider settings, and then expand your choice of provider. You can now fill out the provider form and submit your configuration. Once configured properly you should now have access to use this provider when logging into Orion.



3.1.2 Step 2: Install Orion Essentials Terra Bundle

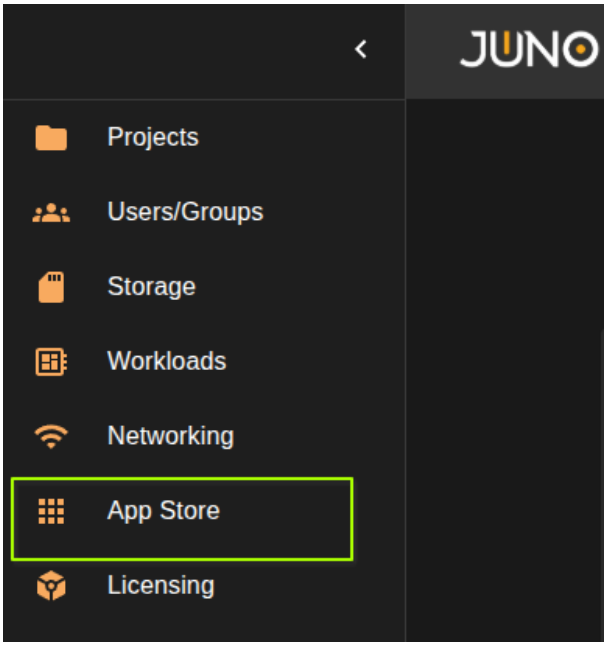
Via our Terra app store you can easily install many different types of plugins directly into your cluster. Please see our Terra App store documentation for a full breakdown of how to navigate the ui and install plugins.

The Orion Essentials bundle installs the ArgoCD dashboard, giving administrators fast and convenient access to cluster deployments, along with our Helios workload schema.

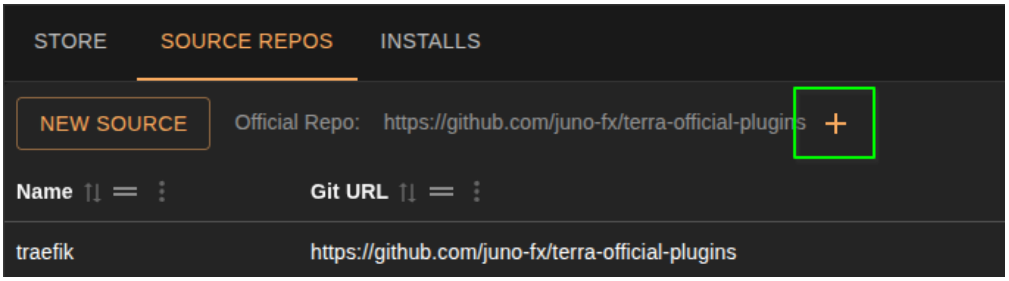
Once installed, you can create Helios workload templates and start launching Helios containerized workstations within Orion.

To learn more about our different plugin types, including dashboards and workloads please see our Terra documentation

Navigate to the Terra App Store

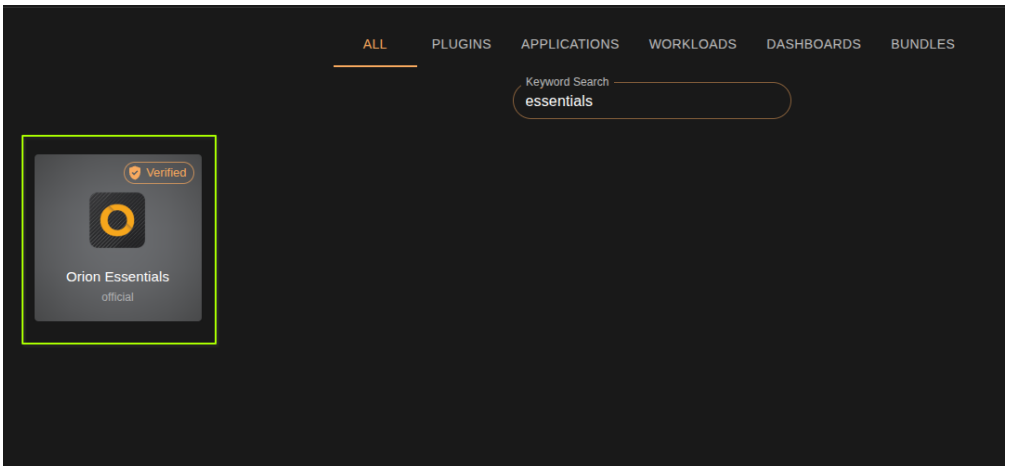


Add the Official Source Repo

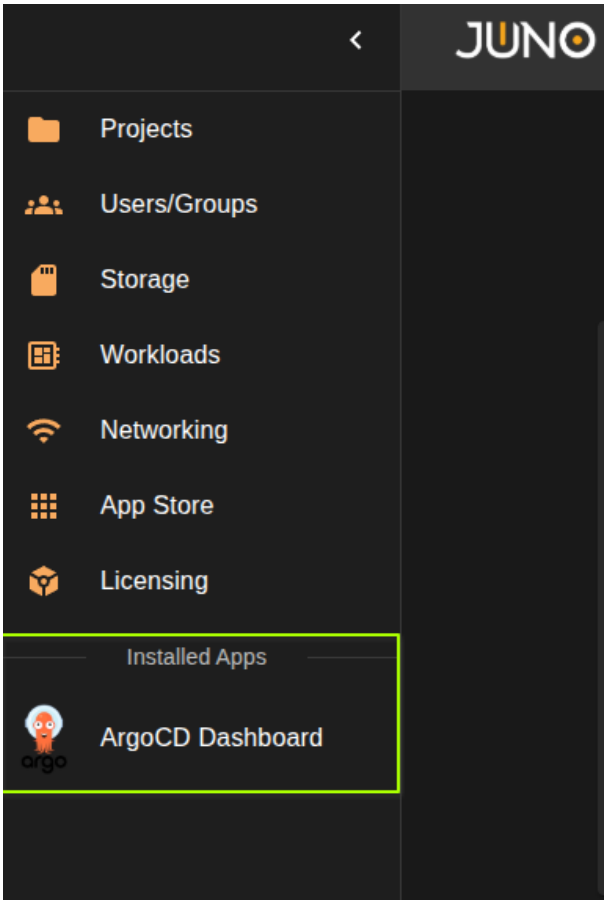


Install Orion Essentials Bundle

Now that the official repo has been added, use the search bar to search for the Orion Essentials bundle. Once found, click the plugin and fill out the install form.



Once installed you will see the ArgoCD dashboard within the Genesis navigation menu, and the Helios workload schema will now be available within the workload creation wizard.

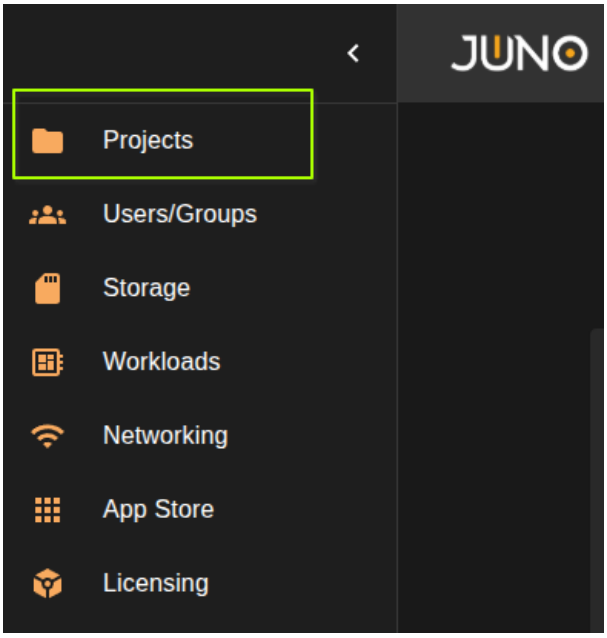


3.1.3 Step 3: Create a Project

Within the Genesis admin dashboard, you can easily create new environments. We call these isolated environments `projects`. You can learn more in our Project documentation including a detailed breakdown of how to create a project.

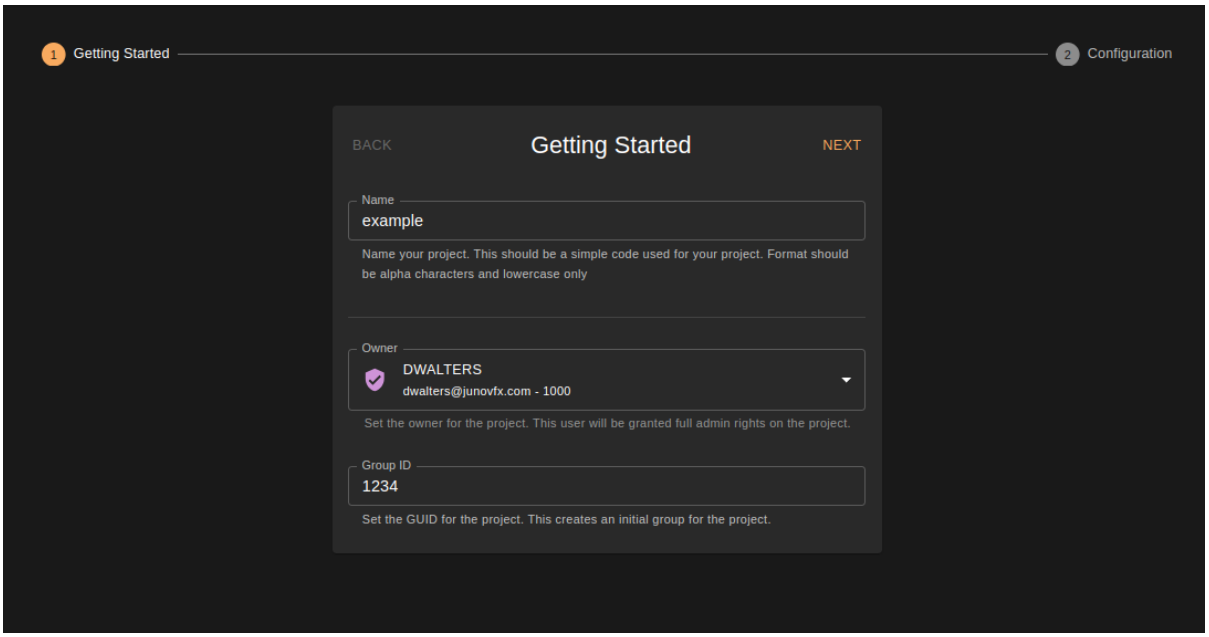
Navigate to the Projects Page

Use the sidebar navigation to navigate to our projects page. Here you can see and manage your project environments.



Create a Project

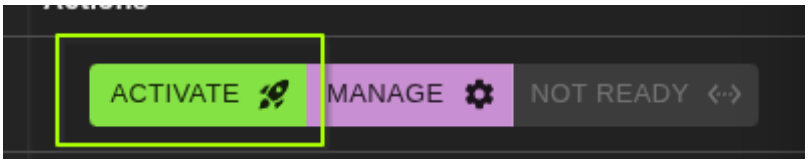
Click the "CREATE PROJECT" button, and fill out the project creation form. Please see our project documentation for more details.



Activate You Project

Now that you've created your project you will want to `Activate` the project. You can activate and hibernate a project at any time. Hibernating a project will automatically scale down the project resources, saving you on your overall compute.

In the projects table, click the `Activate` button in your projects row. Once active, you will see the button shift to `Hibernate` and the Connect button appear.



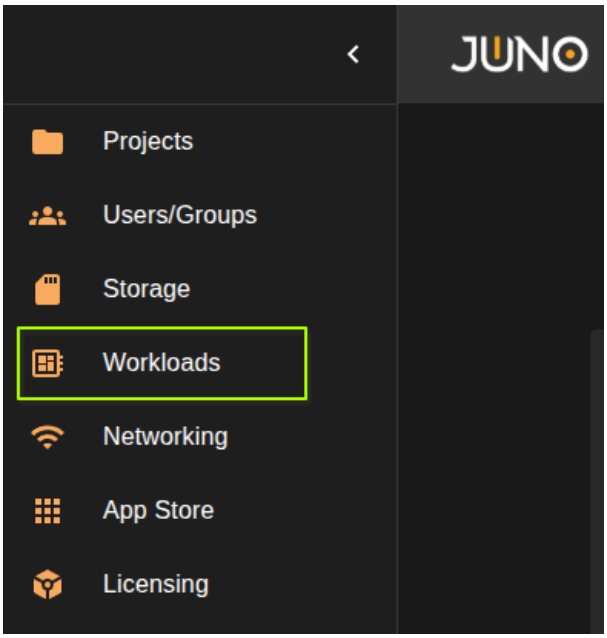
3.1.4 Step 4: Create a Workload

Now that you've installed Helios and created a project we can create a Helios workstation template to launch in our new project environment.

For more information about our open source Helios containerized workstations, please visit our Helios documentation

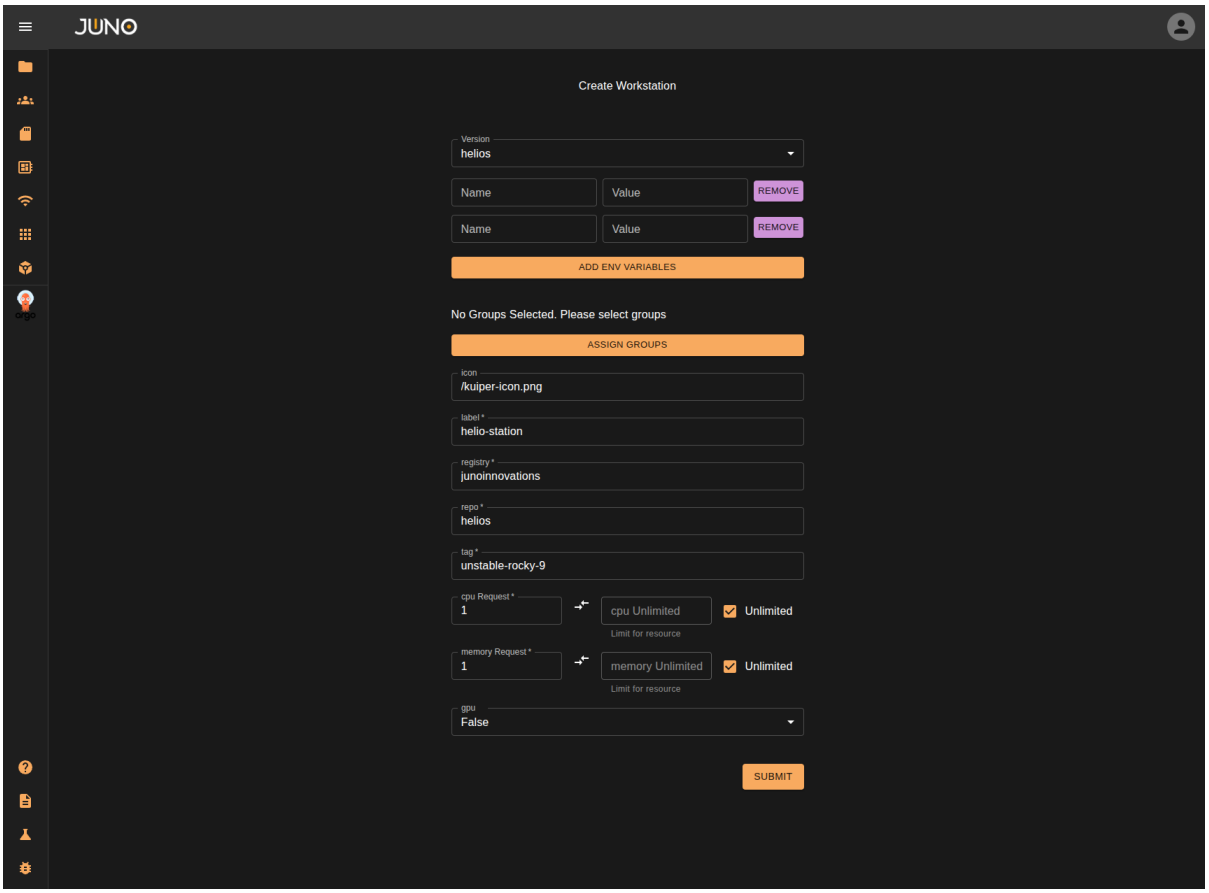
Navigate to the Workloads Page

Use the sidebar navigation to navigate to our workloads page. Here you can see and manage your workload templates. These templates can then be used to launch your containerized workloads within an Orion project.

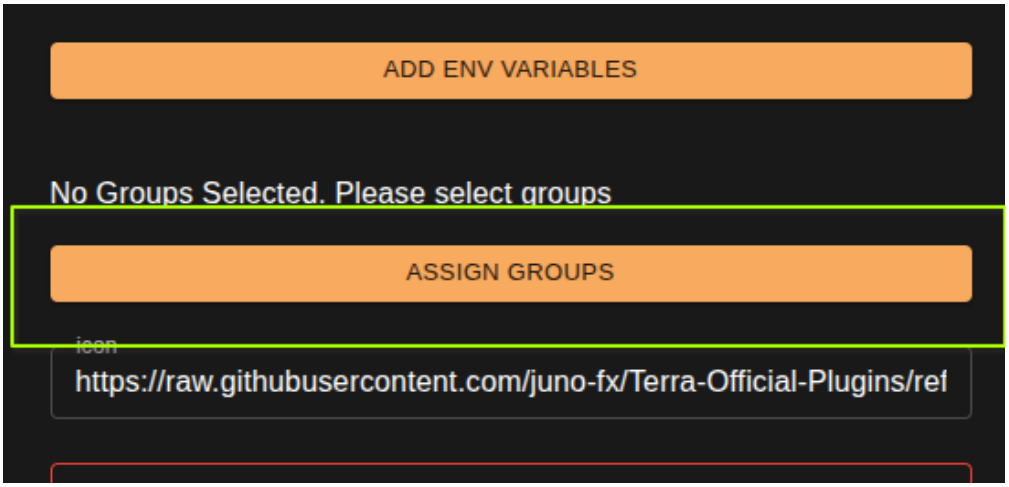


Create a Workload

Click the "CREATE WORKLOAD" button. From there you'll want to select the Helios schema from the dropdown, and fill in the workload form. Please see our workload documentation for more details.



You will want to be sure to assign the newly created project to your templates groups. This will ensure your template is available in the project catalog for users to request and launch.



3.1.5 Step 5: Add and Modify your Nodes

If you utilized our `OneClick` on-prem installer you may only have 1 node currently connected to your Orion cluster.

From the networking page, you can see your active nodes, adjust their labels, adjust their network policies, and if your cluster is an on-prem k3s cluster you will be able to add Ansible credentials and provision new nodes directly from this page.

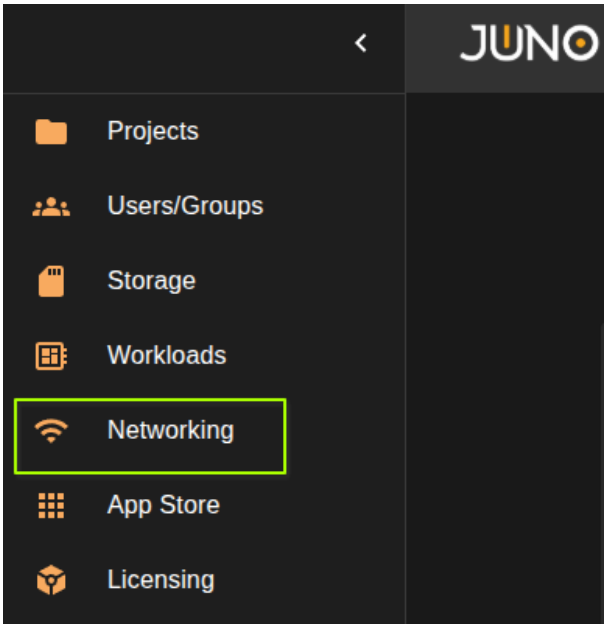
For more information about our networking page please visit our Networking documentation

Note

In order for a workload to launch you will need to ensure you have at least 1 node labeled as `workstation`

Navigate to the Networking Page

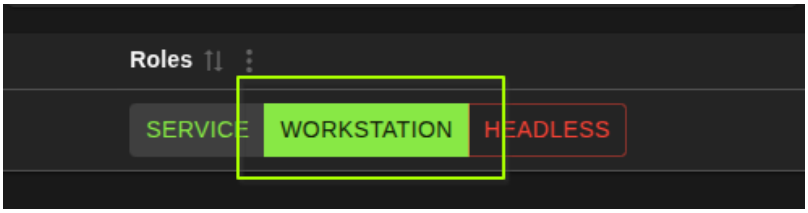
Use the sidebar navigation to navigate to our networking page. Here you can see and manage your connected nodes



Add a Workstation Node

While, you can add more nodes to your cluster, for now lets just ensure we have at least 1 node with the `workstation` role label applied. This will ensure we are able to launch and connect to our workloads.

Find a node in the table, and click the `workstation` button to apply the new label.



Note

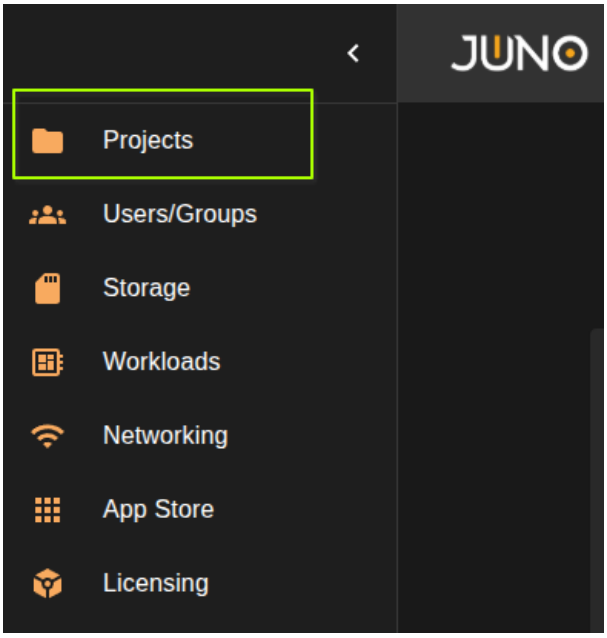
Please note once you have finished your getting started tutorial, it is recommended you have at least 2 service nodes. Along with at least 1 workstation node.

3.1.6 Step 6: Connect to Your Project

Now that you have created a workload template, and have it assigned to your project. Lets connect to your project and launch up the workload.

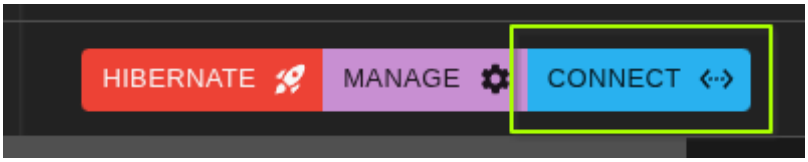
Navigate to the Projects Page

Use the sidebar navigation to navigate to our projects page. Here you can see and manage your project environments.



Click the Connect Button

From the projects page, you should see your newly created project, and it should already be active. Simply click the `Connect` button to connect to your project environment. From there you will be able to login and begin launching your workloads.



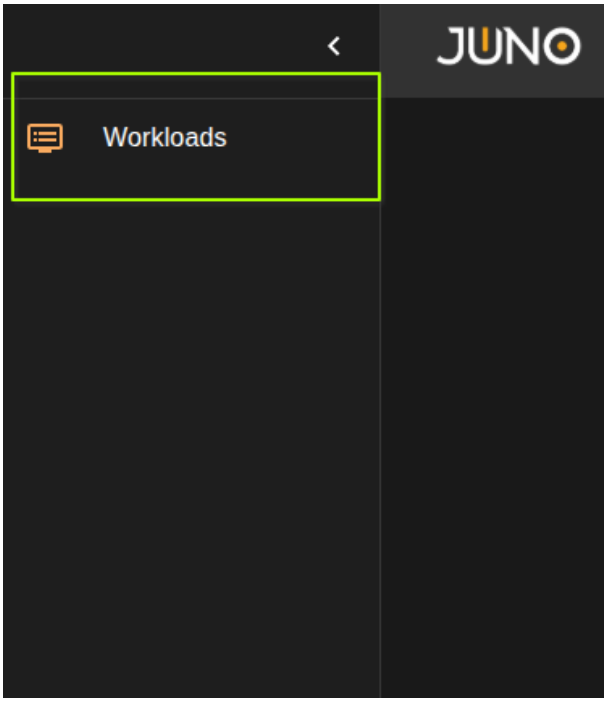
3.1.7 Step 7: Launch a Workload

From here you should be logged into your project environments dashboard. We call this page `Hubble`. It's here you can begin launching and interacting with your workloads.

To learn more about our project environment dashboard including launching and connecting to workloads please see our Hubble documentation

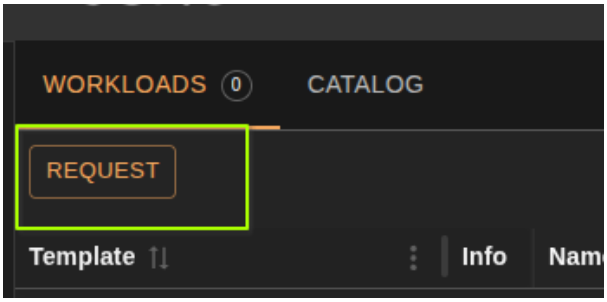
Navigate to the Workloads page

Use the sidebar navigation to navigate to our workloads page. Here you can see, manage, and request new workloads.



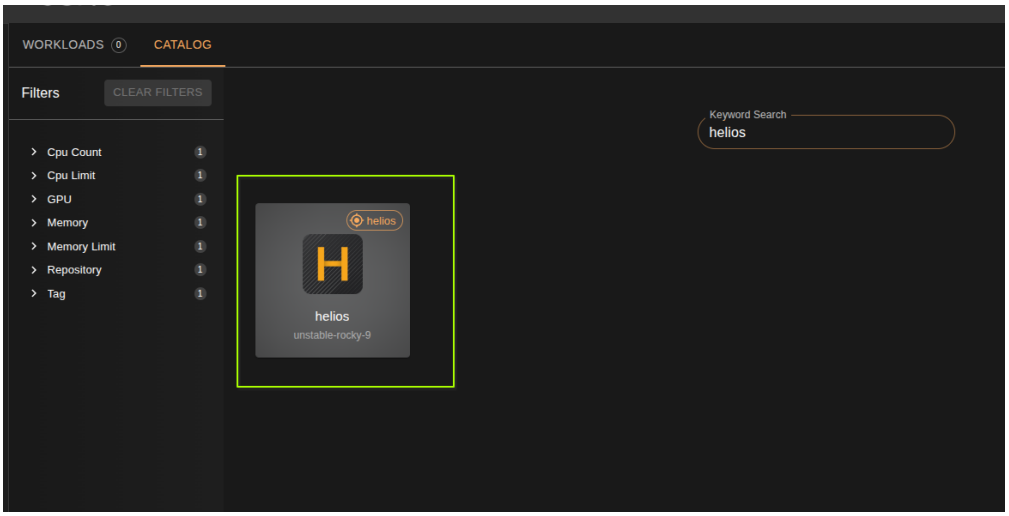
Navigate to the Workload Catalog

From the workloads page, you can either navigate directly to the catalog tab to see available workloads to launch, or click the `Request` button.

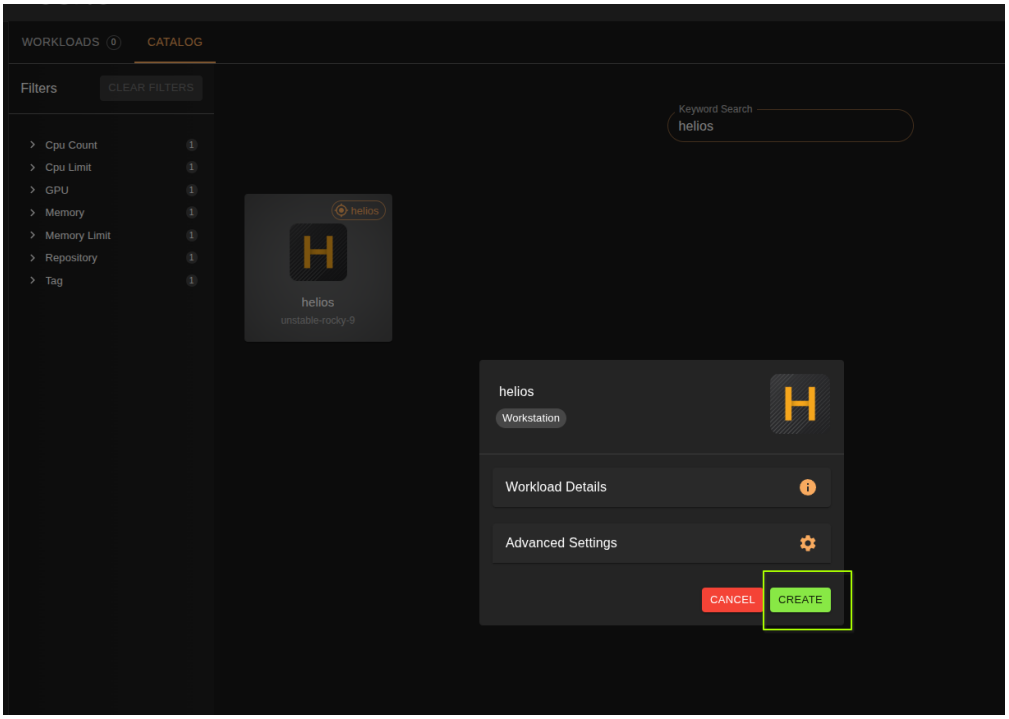


Request a Workload

From the catalog page, you can search and filter your workload templates. Find and click the `Helios` workload template you crated in the previous step.

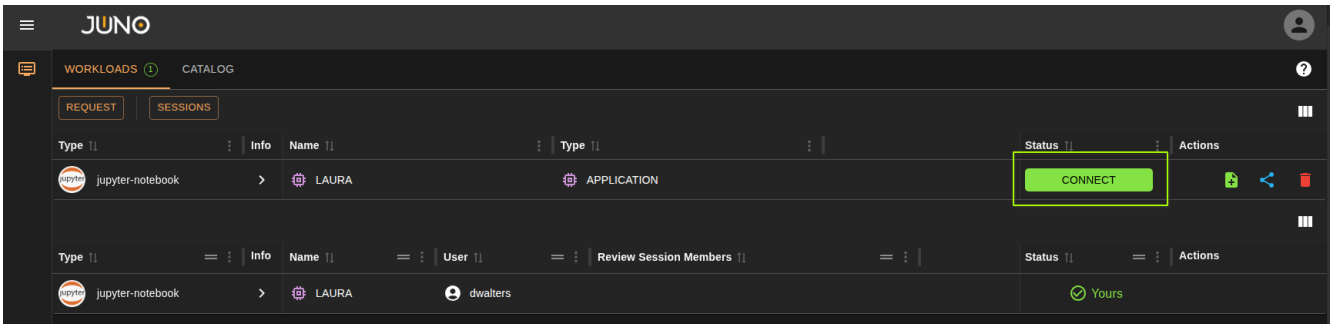


You should now see a pop-up where you can view additional details and `create` your workload. Click the create button to begin launching your workload.

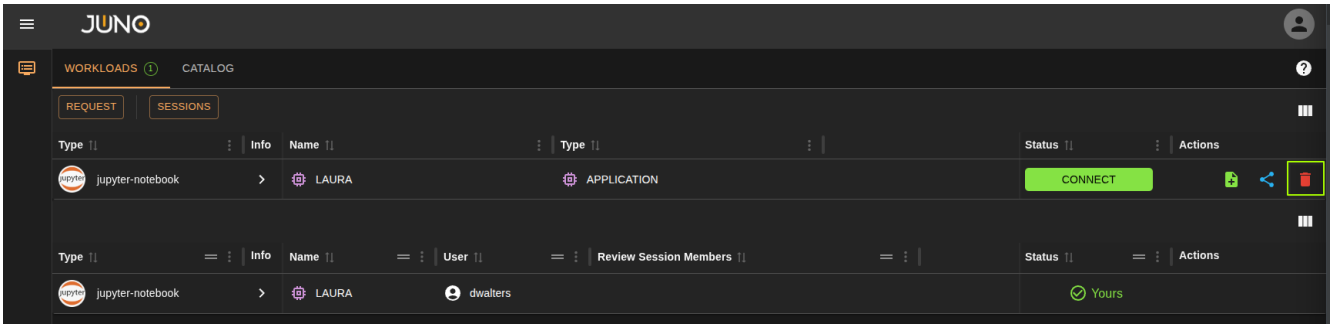


Connect to Workload

Once your workload has been provisioned, and deployed you will be able to connect to it. Click the `CONNECT` button to begin interacting with your workload.



Once you are done, simply click the trashcan icon to delete your workload, and scale your compute back down.



3.2 Conclusion

You have now successfully setup your Orion authorization, created a project, installed plugins, created a workload template, and launch a workload! This getting start tutorial is just the beginning. Please see our full product documentation to see all the Orion has to offer. We here at Juno Innovations not only hope Orion is able to revolutionize how you manage your infrastructure and compute, but also allows you to have fun while doing it.

3.3 Virtual Machine Workflow

3.3.1 On-prem

Juno Orion Virtual Machines via KubeVirt

Managing your on-prem virtual machines has never been easier with Orion and KubeVirt. Create vm workloads, golden images, and migrate your vm's with ease.

WHAT IS KUBEVIRT?

KubeVirt extends Kubernetes to run traditional virtual machines alongside containers. This enables you to:

- **Run Windows workloads** on your Kubernetes infrastructure (Windows 10, Windows 11, Windows Server)
- **Consolidate infrastructure** by managing VMs and containers through a single platform
- **Leverage Orion's orchestration** for VM lifecycle management, resource allocation, and golden image deployment
- **Enable VDI workflows** with built-in connection broker capabilities through dynamic port assignment

With Orion + KubeVirt, you can run Adobe Creative Suite, Autodesk products, Houdini, DaVinci Resolve, and other Windows-only applications within your Kubernetes cluster—all managed through the familiar Orion interface.

COMING FROM OTHER PLATFORMS?

Coming from Proxmox, VMware, or Similar Platforms? If you're familiar with traditional hypervisors, here's how Orion + KubeVirt maps to concepts you already know:

Traditional VM Concept	Orion Equivalent
VM Template	Workload Template (Genesis → Workloads)
ISO Library	HTTP mounts with direct URLs (or self-hosted ISOs over HTTP)
Datastore / Storage	DataVolumes and PVCs (managed automatically)
Clone / Linked Clone	Golden Image → Clone mount type
vCPU / Memory allocation	CPU/Memory requests and limits in template
Port forwarding / NAT	Port configuration in template
Console / VNC	Connect button → URL/Name option
VM Snapshot	Clone action on DataVolumes

Key Differences to Keep in Mind:

- **No GUI for all settings.** Some configurations that are point-and-click in Proxmox are handled through template definitions in Orion. The tradeoff is repeatability — once your template is right, every deployment is identical.
- **VNC is barebones.** You're used to a decent console experience in Proxmox/VMware. Here, the VNC viewer is intentionally minimal — it's for setup and emergencies, not daily use. Plan on RustDesk or similar for actual work.
- **Resource limits are strict.** No "unlimited" or "ballooning" — you define what the VM gets, and that's what it gets. Please plan accordingly. This feature is being explored.
- **Kubernetes-native networking.** Instead of virtual switches and bridges, you're working with Ports and Kubernetes services. The Connect button abstracts most of this for you.

What You'll Find Familiar:

- VirtIO drivers — same process as Proxmox
- Golden image workflow — conceptually identical to template creation
- ISO booting and installation — same BIOS/UEFI experience
- Windows quirks — still the same Windows quirks

The mental shift is thinking of VMs as "workloads" managed through templates rather than individual machines you configure one at a time. Once that clicks, you'll find the workflow surprisingly efficient for deploying at scale.

Getting Started

In this guide we will walk you through the following:

- Installing KubeVirt and our Generic Ephemeral VM plugins
- Create and configure a Windows VM workload Template
- Launch and connect to a Windows VM workload template
- Setup and install Windows along with Rustdesk
- Create a golden image of our VM

If you are new to Orion, please see our getting started docs before following this guide.

HARDWARE REQUIREMENTS

Minimum specifications for running Windows VMs:

Resource	Minimum	Recommended
CPU Cores	16 cores	20+ cores
RAM	32 GB	64 GB
Storage	200 GB free	500+ GB free

Important: These are cluster-level minimums. Individual Windows VMs require at least **4 cores** and **8 GB RAM** each. Plan your capacity accordingly.

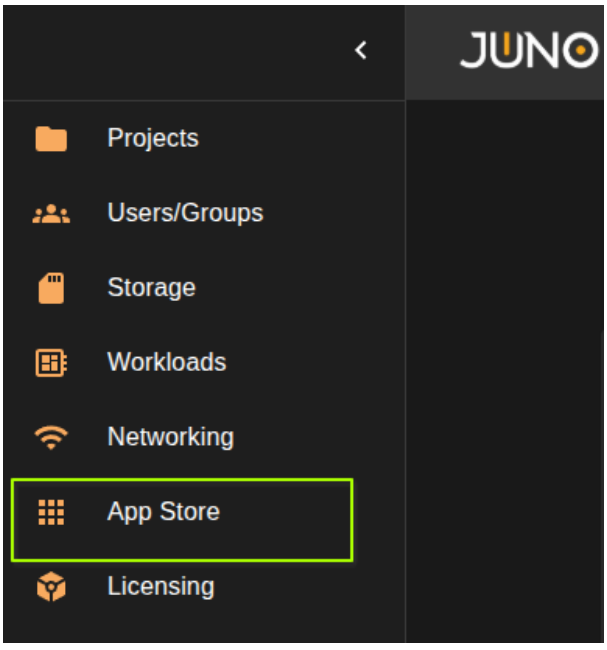
STEP 1: INSTALL THE KUBEVIRT PLUGIN

Start by installing the KubeVirt plugin from the Terra Official Plugins repo.

Important

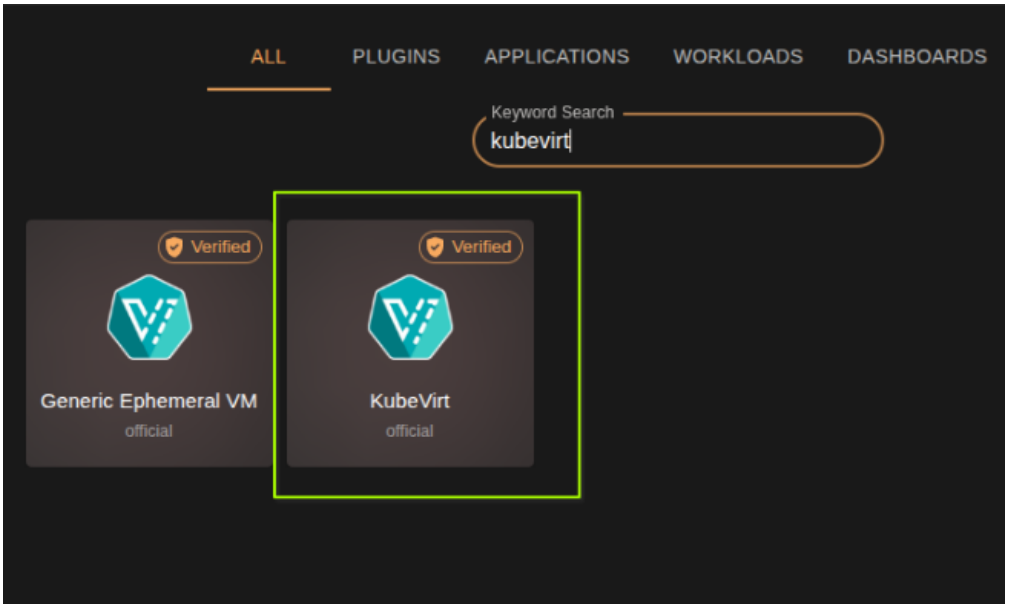
Install this plugin FIRST. The Generic Ephemeral VM plugin depends on KubeVirt being installed.

Navigate to the Terra App Store

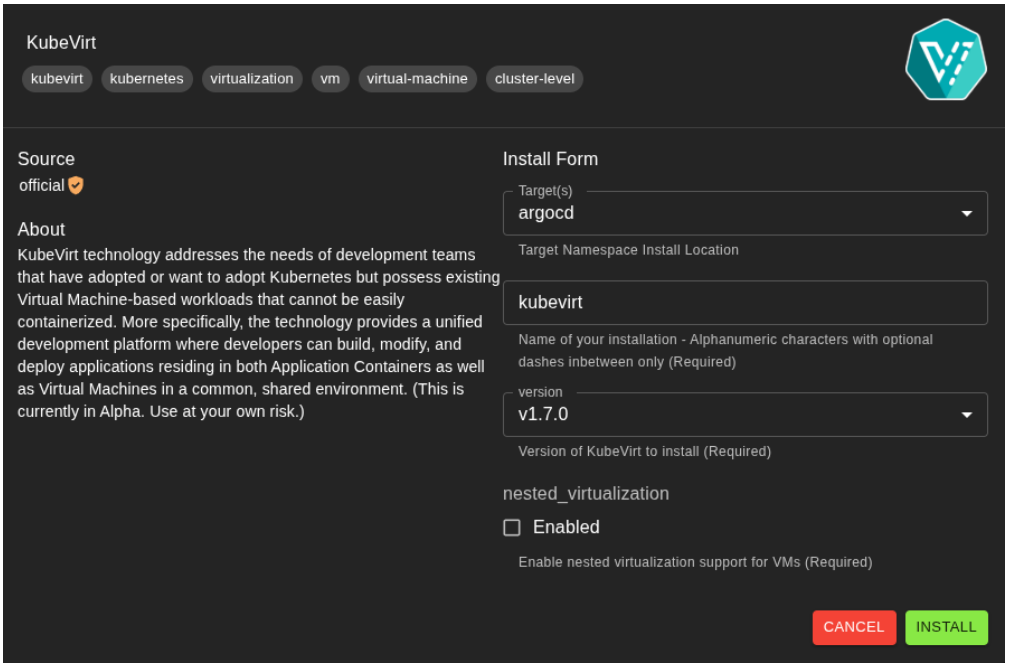


Install KubeVirt

Search for the Kubevirt plugin in the Terra App Store.



Once found, select the plugin, fill out the form and click **Install**



Note

For clusters composed of Virtual Machines (meaning the cluster is deployed on top of VMs), nested virtualization must be enabled. Bare-metal deployments typically don't require this.

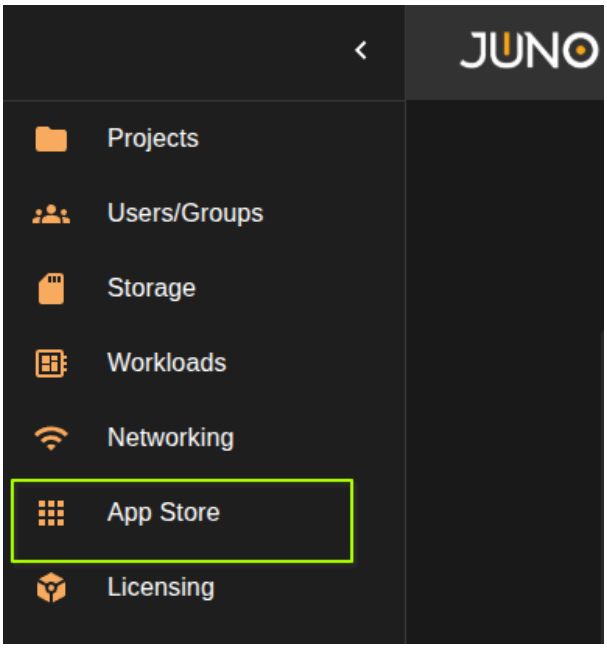
STEP 2: INSTALL THE GENERIC EPHEMERAL VM PLUGIN

This plugin provides the schema/template for creating VM workloads.

Important

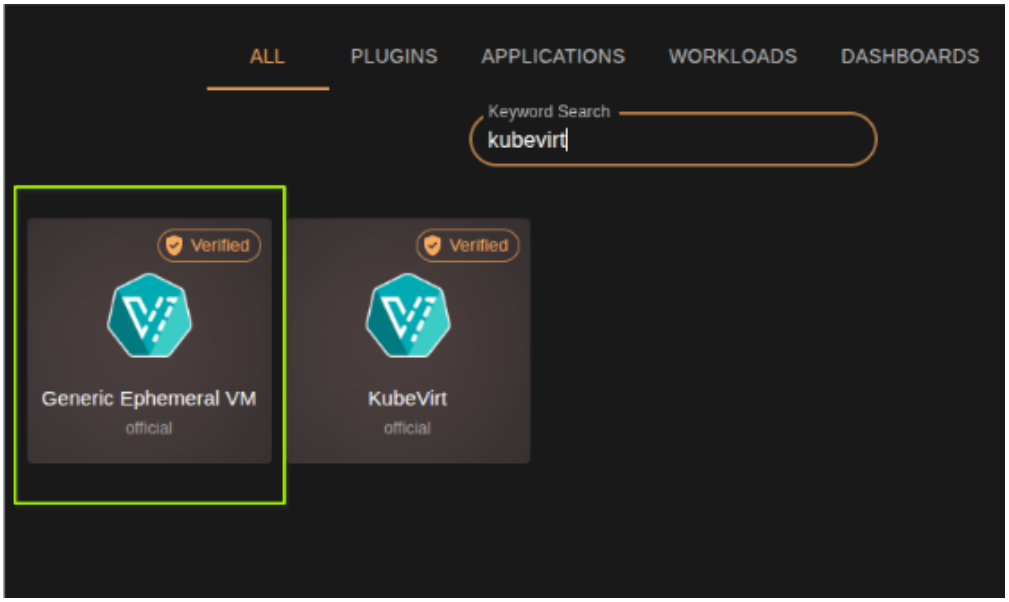
Install this plugin SECOND. It requires KubeVirt to be installed first.

Navigate to the Terra App Store

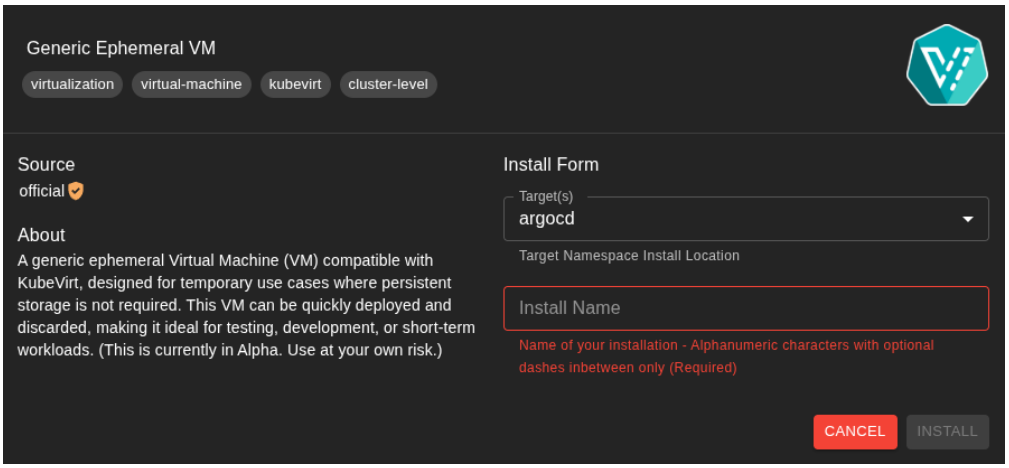


Install Generic Ephemeral VM

Search for the Generic Ephemeral VM plugin in the Terra App Store.



Once found, select the plugin, fill out the form and click **Install**



Create a Windows VM Workload Template

For this example, we will create a Windows VM workload template

> **Note**
>
> If you haven't installed the needed plugins please see our VM Getting Started documentation

STEP 1: GET THE WINDOWS ISO URL

You don't need to download the ISO manually — Orion uses KubeVirt's native HTTP downloader to import ISOs directly.

> **Tip**
>
> You can use Windows 10, Windows 11, or any bootable ISO. You can also serve ISOs from your local network over HTTP if you have them available internally. Microsoft imposes a 24 hour limit on the validity of this link. If you get a 404 error during import, you need to request a new link from this page.

Gather your ISO image URL

1. Navigate to: https://www.microsoft.com/en-us/software-download/windows10ISO
2. Select your edition and language
3. **Right-click** on the 64-bit download button and select **Copy link address**
4. Save this URL—you'll use it in the template creation form

STEP 2: CREATE THE WORKLOAD TEMPLATE

Navigate to the Workload template Creation Form

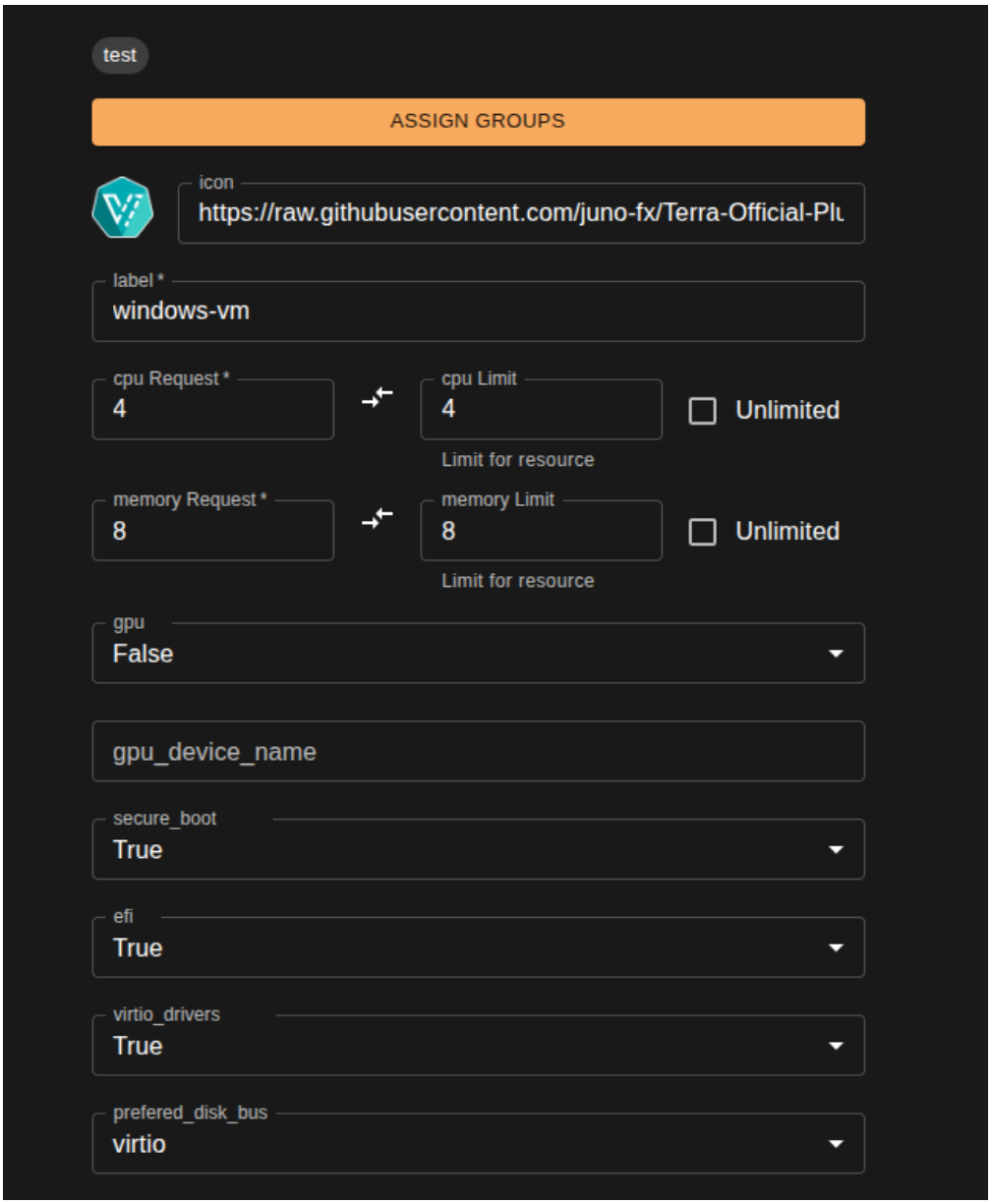
1. From the Genesis **Workloads** table, click **Create Workload**
2. Select the **VM schema** you installed previously in the getting started section.

Configure Windows VM Settings

With your VM schema selected we can begin configuring our template.

Set the following values for a Windows VM:

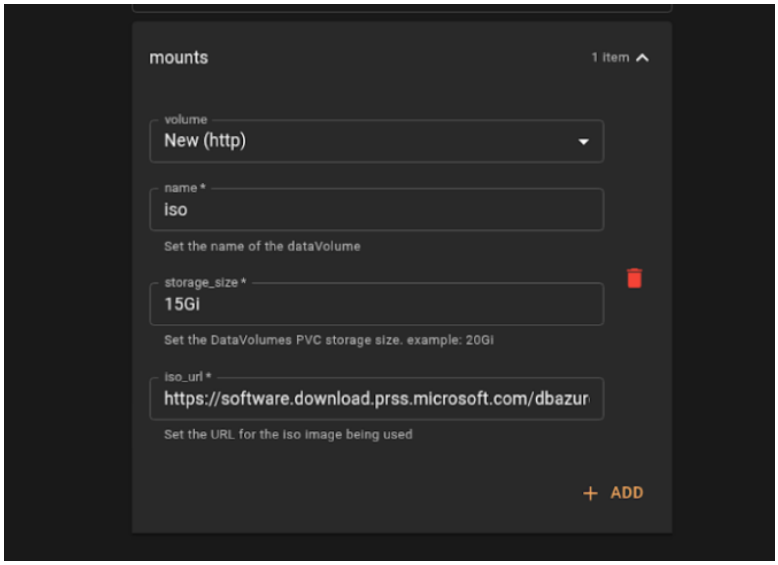
Setting	Value	Notes
Label	Your choice (e.g., "Windows10")	Displayed in the catalog
Groups	Select your project group	Add other groups to restrict access
secure_boot	`true`	Required for Windows
efi	`true`	Required for Windows
virtio_drivers	`true`	Required for Windows — loads drivers at boot
CPU Request	4	Minimum for Windows
CPU Limit	4 (or higher)	Do NOT use "unlimited"
Memory Request	8Gi	Minimum for Windows
Memory Limit	8Gi (or higher)	Do NOT use "unlimited"



Warning

Important: The "unlimited" checkbox does not work for virtual machines. Always set explicit CPU and memory limits. Also, spaces are **NOT** allowed in Labels.

Add the ISO Mount (HTTP)

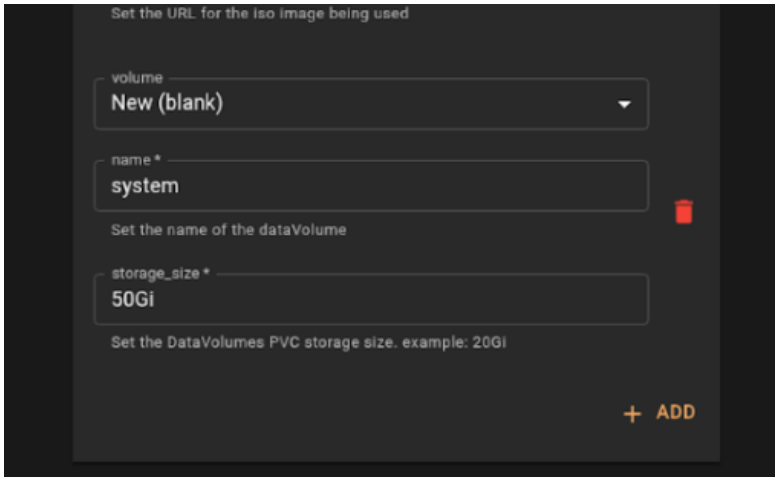


1. Click **+ Add** to add a new mount
2. Set the mount type to **HTTP**
3. Configure:

- **Name:** `iso (or preference)`
- **iso_url:** Paste the Windows ISO URL from Step 1
- **storage_size:** `8Gi` (must be larger than the ISO file—Windows 10 is ~5.7 GB)

> **Warning**
>
> The size value format is important. Always use `Gi` suffix for gigabytes (e.g., `8Gi`, `15Gi`). Incorrect formatting will cause volume creation to fail.

Add the System Disk (Blank)



1. Click **+ Add** to add another mount
2. Set the mount type to **Blank**
3. Configure:

- **Name:** `system`
- **storage_size:** `50Gi`

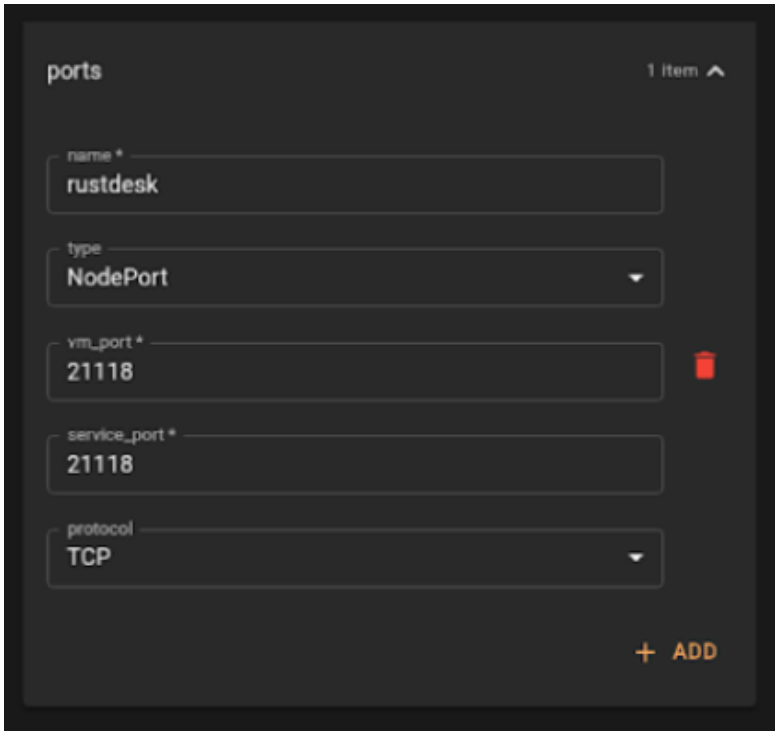
Info

This creates a 50 GB disk where Windows will be installed. Plan this size based on your intended use — you cannot resize it after creation, but you can attach additional drives later if needed.

Configure RustDesk Port

As a demonstration, we will use RustDesk as a performant VDI solution. Configure the port exposure:

Setting	Value
Name	`rustdesk`
Type	`NodePort`
vm_port	`21118`
service_port	`21118`
Protocol	`TCP`



Info

When you set a port, it overwrites the defaults — you're not adding to them.

Why NodePort? This dynamically assigns a port and exposes it to the network, enabling Orion to function as a connection broker. You can launch thousands of VMs, and each will receive a unique port assignment accessible via the Connect button.

Other Use Cases for Ports:

- **Remote desktop protocols** (RGS, Parsec, etc.)
- **Game servers** (Minecraft, dedicated servers)
- **Development servers** (database access, web services)
- **Any application requiring direct network access**

Submit the Template

Click **Submit** to create the workload template. It will now appear in your project's workload catalog.

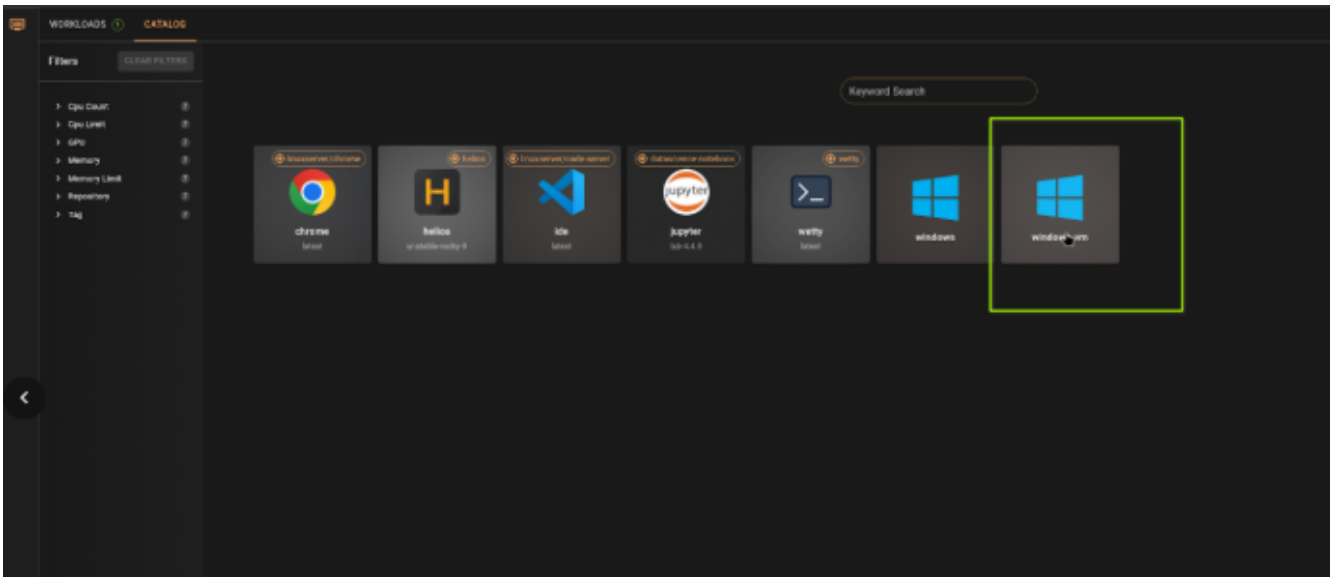
Launch and Connect to Your Windows VM

With you VM template now created, and assigned to a project. You now can connect to your project, and launch the Windows VM workload. From there we can begin installing windows, and setting up our golden image.

STEP 1: LAUNCH YOUR WORKLOAD

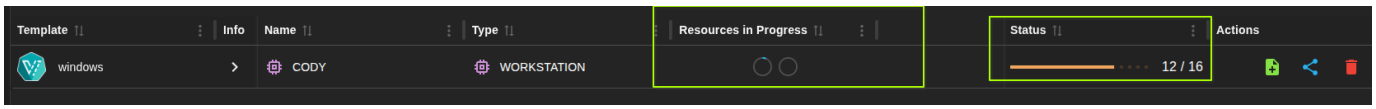
Launch Workload

1. Navigate to the project you have assigned your new Windows VM Workload.
2. Navigate to the **Workloads** table in Hubble
3. Click **Request** to open the catalog
4. Find your Windows VM workload and click **Create**



STEP 2: MONTIOR THE LAUNCH

Your VM will begin launching. The status shows a progress bar with resource counting (e.g., "⅗ ready"). You will also see progress wheels for some of our resource actions. Giving you insight into how much longer until the VM is ready. Hovering over the progress wheels will show the percentage complete.



ıats Happening

1. Orion imports your ISO via HTTP (this can take several minutes depending on your connection)
2. Once import completes, the VM boots
3. All associated resources provision and initialize

tails Panel

Click the dropdown arrow on your workload to see the expanded details panel.

This includes:
- **Resources table** showing all components (ConfigMaps, DataVolumes, PVCs, Ingress, VirtualMachine, etc.)
- **Current state** of each resource
- **Node assignment** for each resource
- **Last event** for troubleshooting
- **Actions** available for each resource type

Tip

To learn more about our available actions, see our resource actions documentation

Note

ISO import time depends on your internet connection and the ISO size. For a 5.7 GB Windows ISO, expect 5-15 minutes. You only have to do this once per Windows ISO version.

STEP 3: CONNECT TO THE VM

Once the VM shows as ready, click the **Connect** button. For this particular workload you will be presented with a couple of different connection options.

nnection Options

Connection options can very between workloads and is dependent on how the author of the workload configured the chart. For this Windows VM example, you will see the ability to connect directly to an HTTP endpoint, as well as a port option. Once connected we will walk through the windows installation process.

For this initial Windows installation, select the VNC option.

Option	Use Case
URL/Name (VNC)	Initial setup and maintenance only
Port	For RustDesk and other VDI connections (primary use)

Warning

VNC is for **installation and maintenance only**. Performance is poor and not suitable for regular work. Once Windows is installed and RustDesk is configured, use the Port connection for actual VDI work.

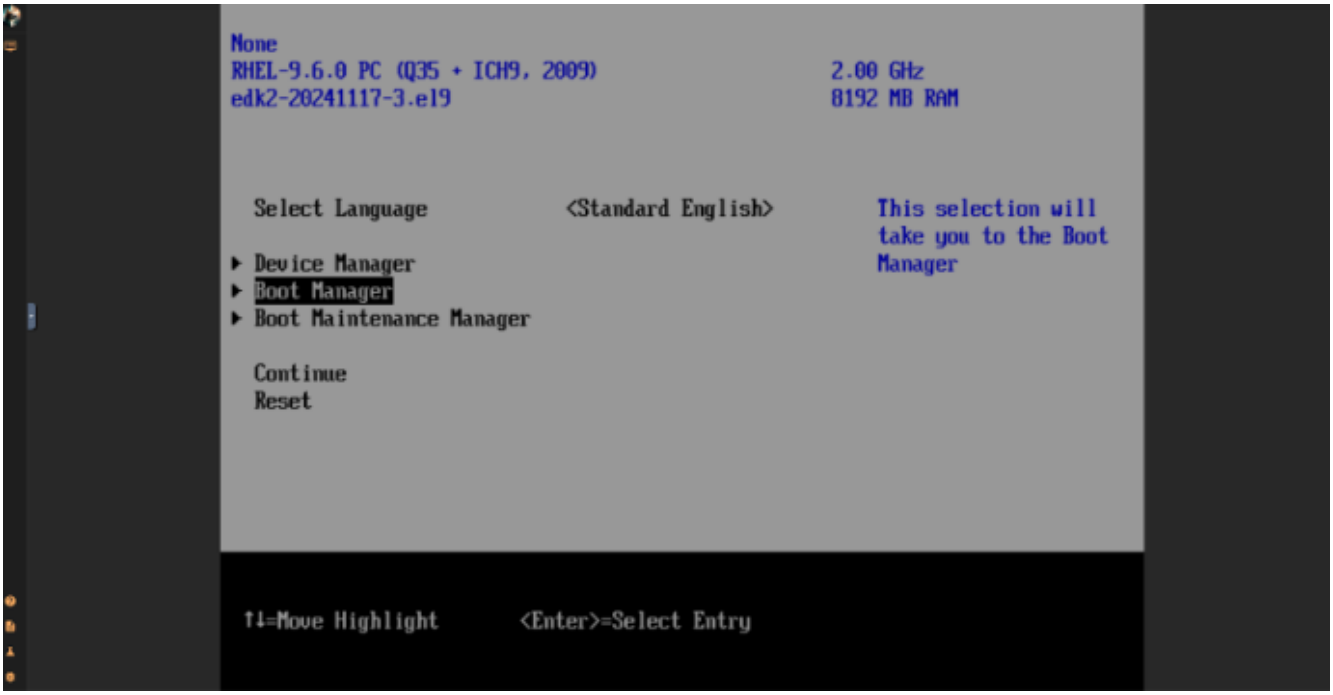
Windows Installation and Setup

STEP 1: BOOT FROM THE ISO

On first boot, the BIOS may not automatically identify the ISO as bootable.

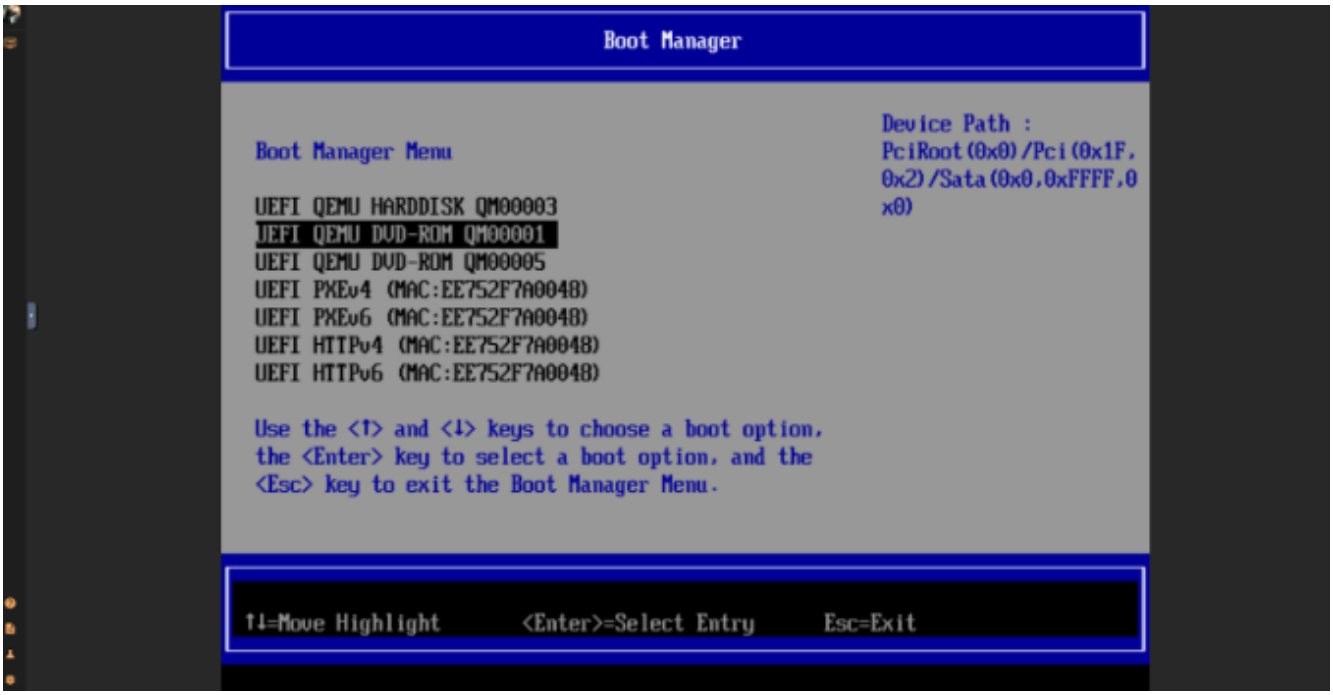
Enter the boot manager

1. Press **Enter** when prompted to open the BIOS
2. Select **Boot Manager**



Update boot settings

1. Select the first **QEMU DVD-ROM**
2. Press **Enter** when asked to boot from disk



STEP 2: INSTALL WINDOWS

Proceed through the windows setup screens.

Important

1. When asked for a product key, select **"I don't have a product key"**. You can enter your product key after your drivers are installed.
2. Select **Custom: Install Windows only** (not Upgrade)
3. Select the **50 GB drive** we defined during our template creation as the installation target
4. Wait for installation to complete (Windows may reboot several times—this is expected)
5. When Windows asks about internet connection: Select **"I don't have internet"**
6. Select **"Continue with limited setup"**

Why no internet? Windows doesn't ship with VirtIO network drivers. The network adapter won't function until you install drivers in the next section.

VNC Notes

While using the VNC viewer during installation:

- **Mouse behavior may be erratic**—this is normal and only affects VNC
- **No audio**— VNC does not support audio
- **Disconnections**—VNC may disconnect and reconnect periodically; this is expected

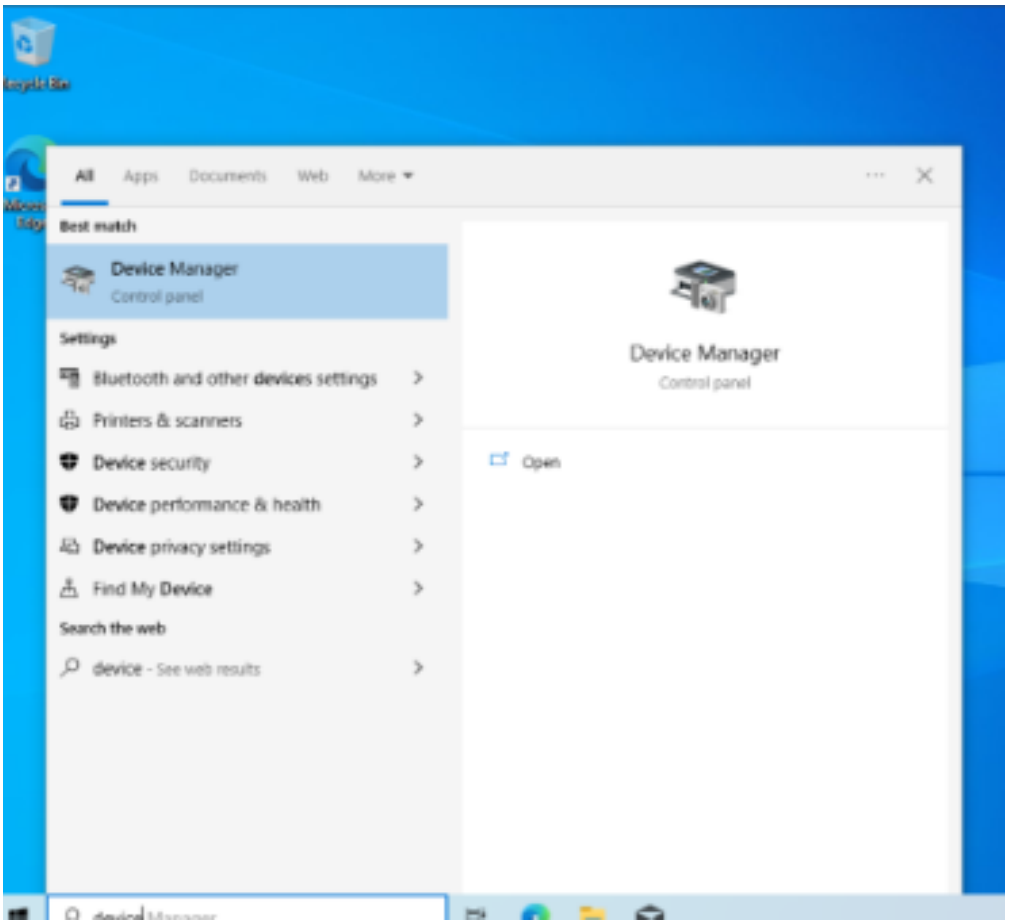
These issues are resolved when you switch to RustDesk or other VDI solutions for regular use.

STEP 3: INSTALL VIRTIO DRIVERS

With Windows installed, you need to install VirtIO drivers for network access and other virtual devices.

Open Device Manager

1. Search for **"Device Manager"** in Windows
2. Open it



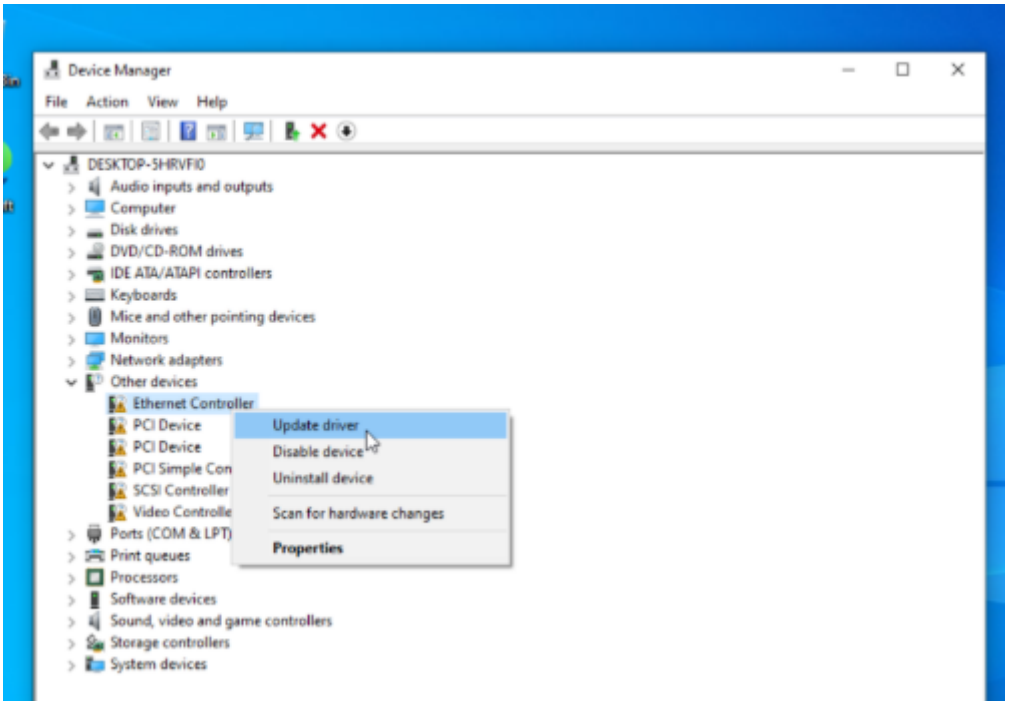
Identify Missing Drivers

Under **"Other devices"**, you'll see several items with yellow caution icons. These are the VirtIO devices needing drivers, including your network adapter.

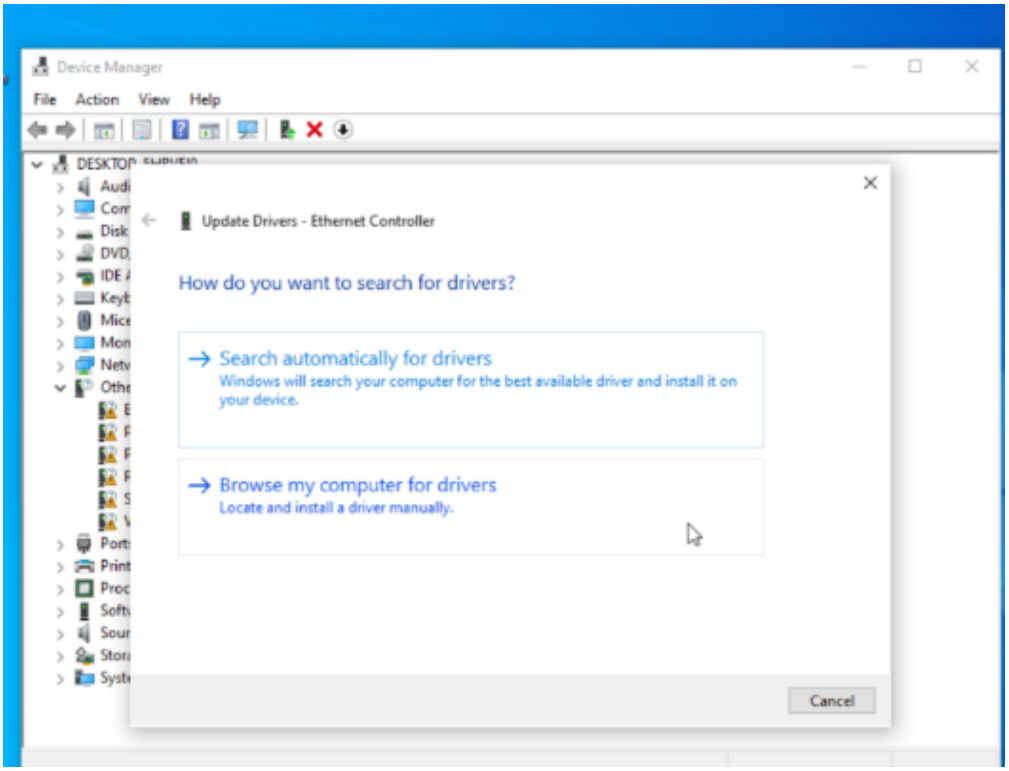
Install Drivers

For each device with a caution icon:

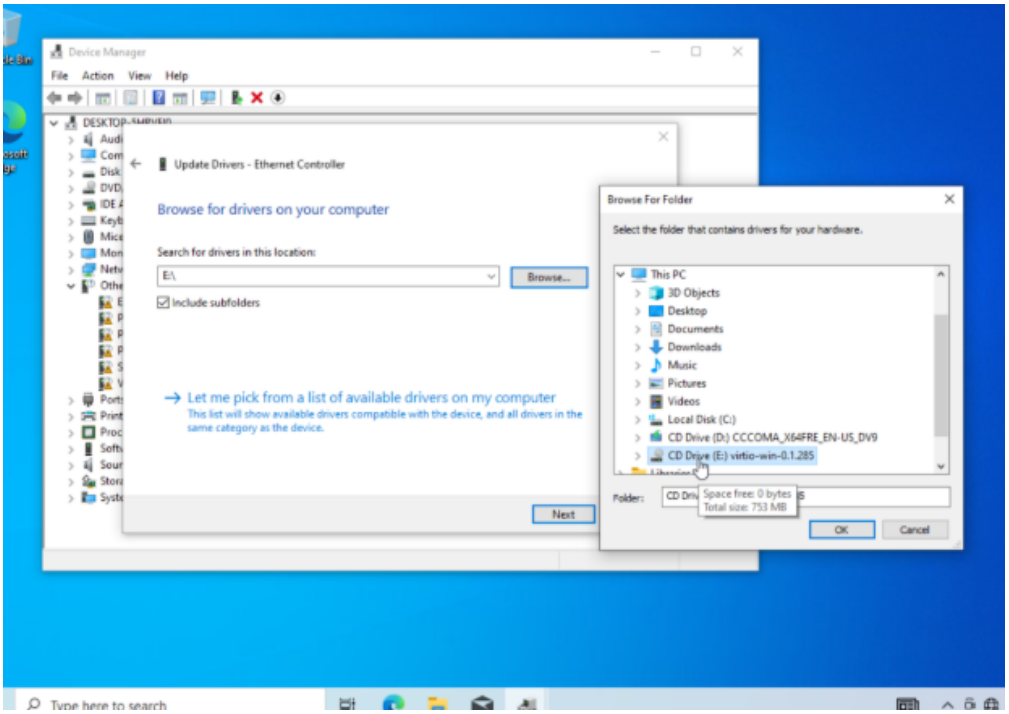
1. Right-click the device → **Update driver**



1. Select **"Browse my computer for drivers"**



1. Click **Browse** and navigate to the CD drive containing "**virtio**" in the name
2. Select the root of the disk and click **OK**



1. Click **Next** to install

> **Note**
>
> Repeat for ALL devices except the **"Video Controller"**— there is no VirtIO driver for video, so you can skip that one.

STEP 4: VERIFY NETWORK CONNECTION

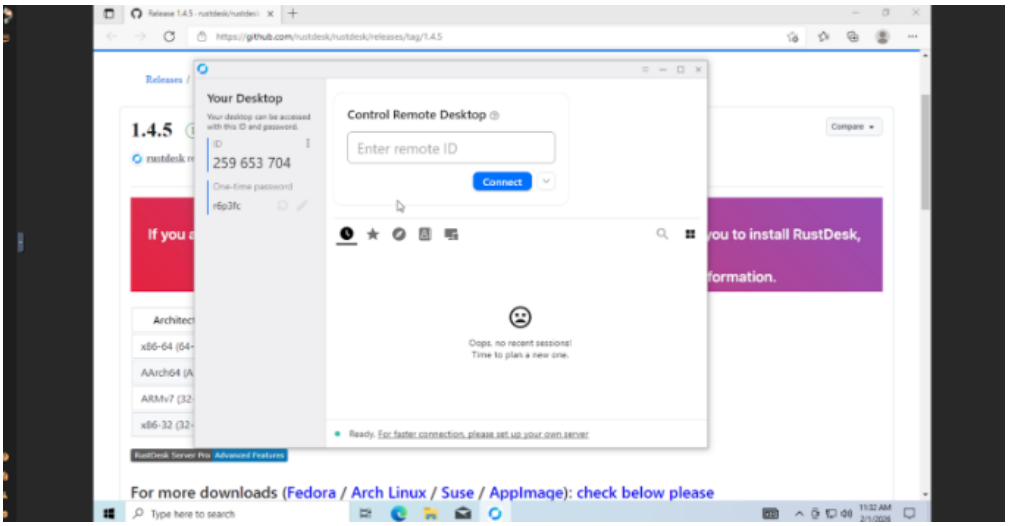
After installing the Ethernet controller driver, Windows should connect to the internet. Verify by opening a browser or checking network status.

STEP 5: INSTALL AND CONFIGURE RUSTDESK

With internet access available, install RustDesk for performant remote access.

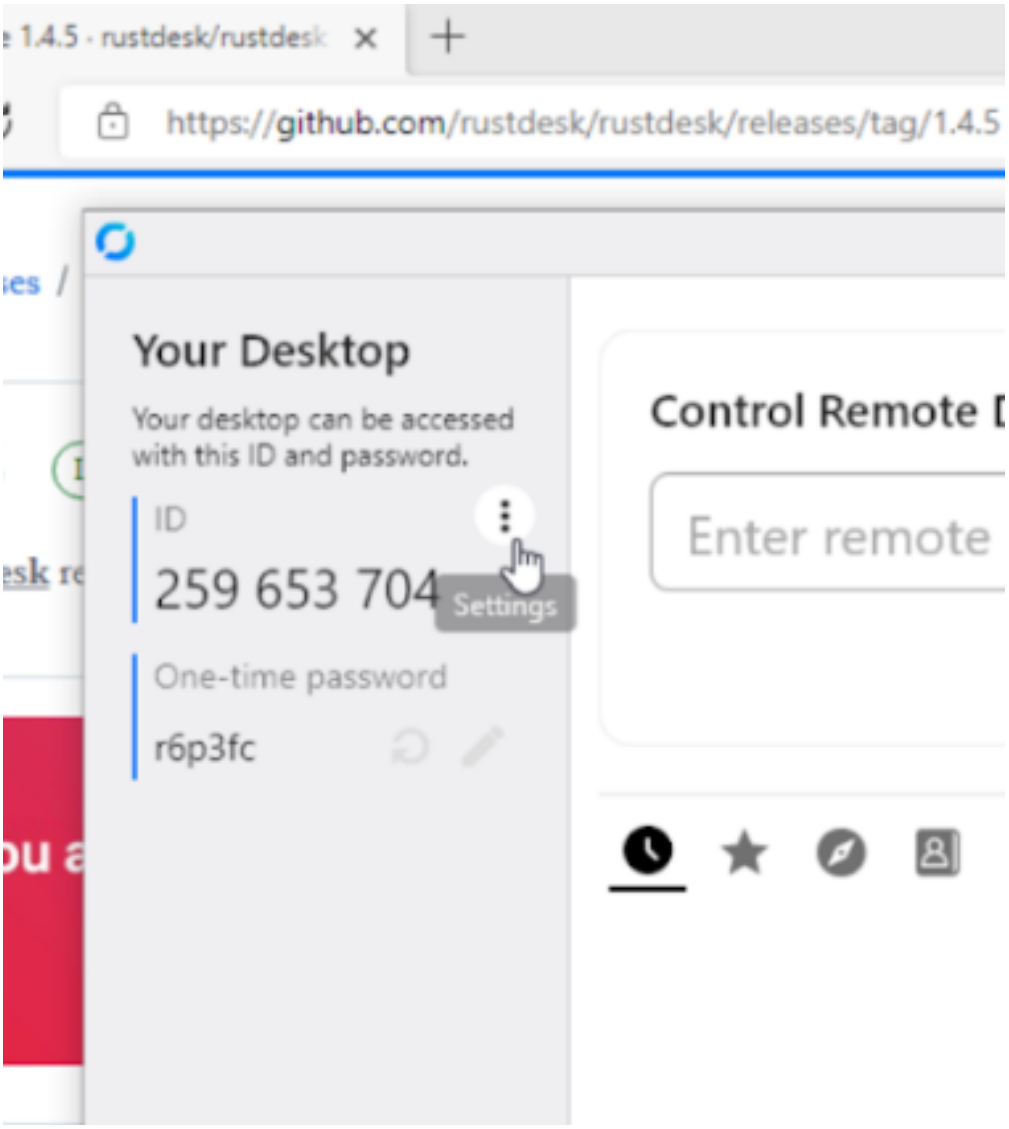
Download and Install

1. Navigate to https://rustdesk.com
2. Download the Windows installer
3. Run the installer and complete setup

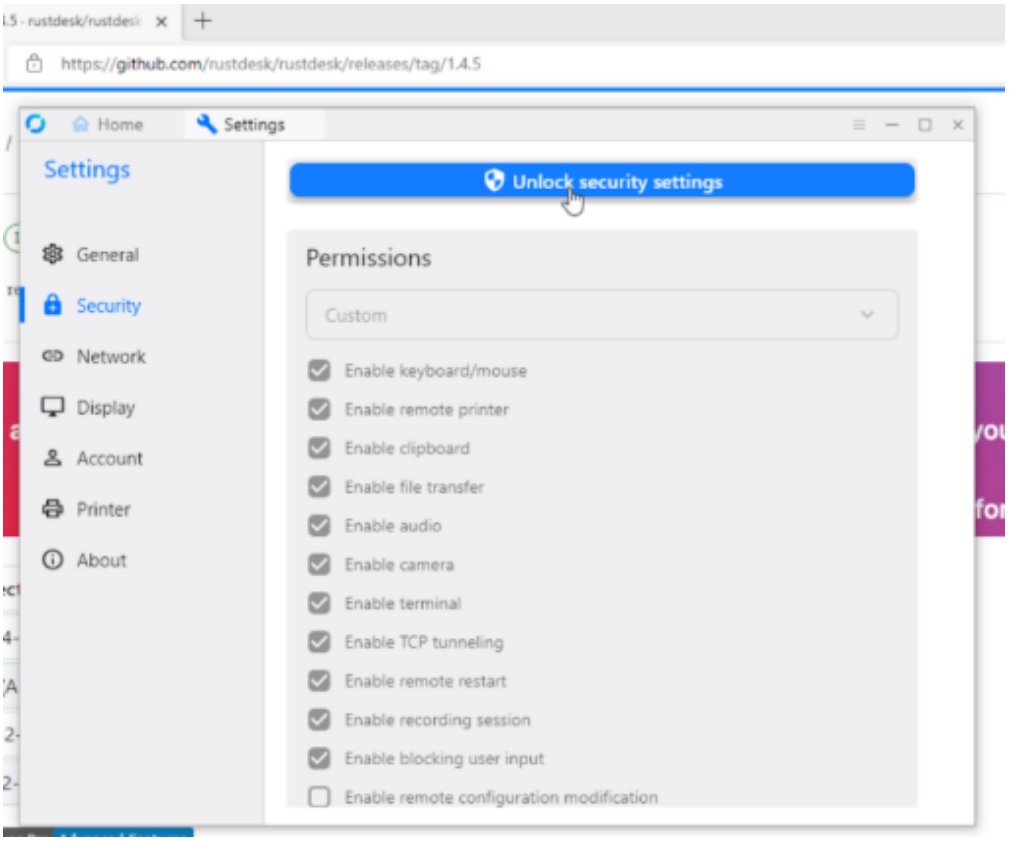


Configure RustDesk

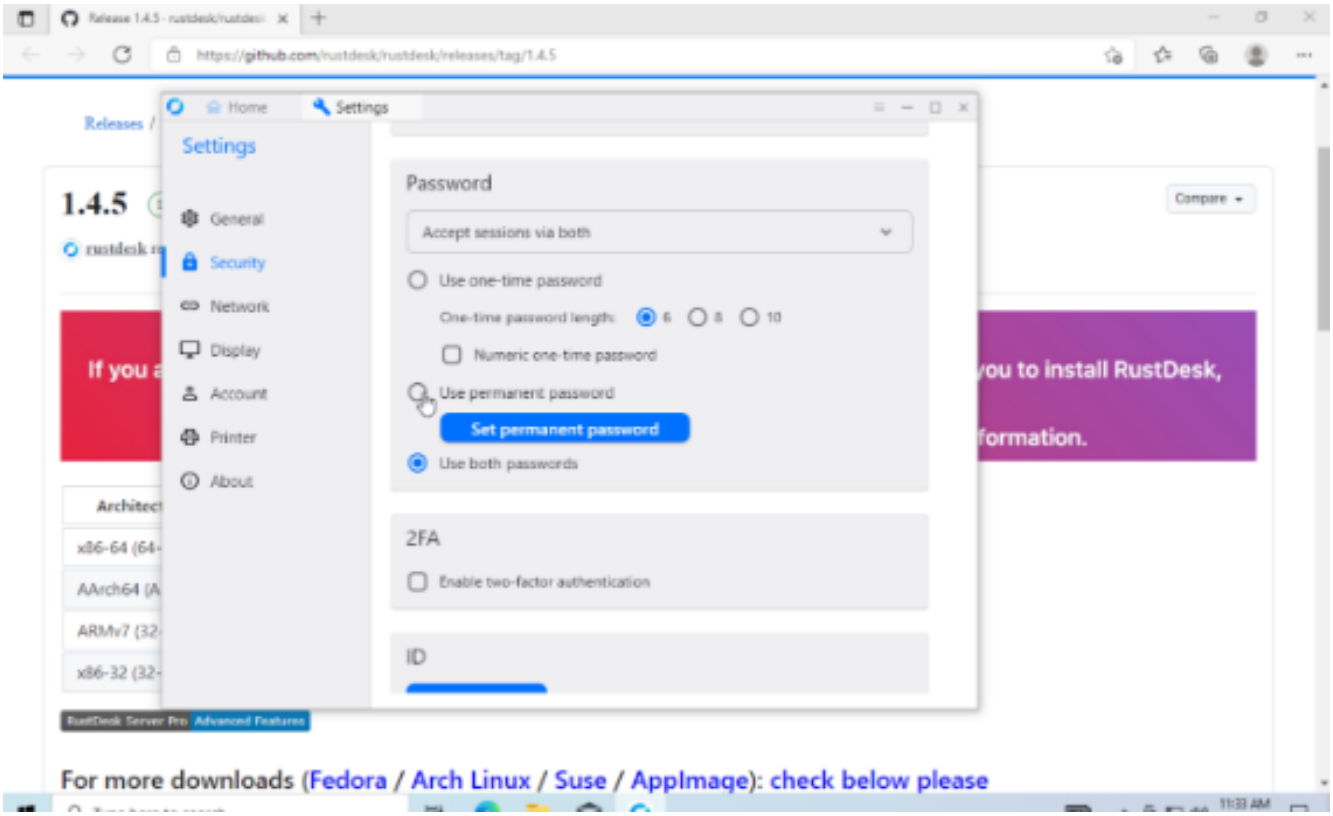
1. Open RustDesk and click **Settings** (gear icon)



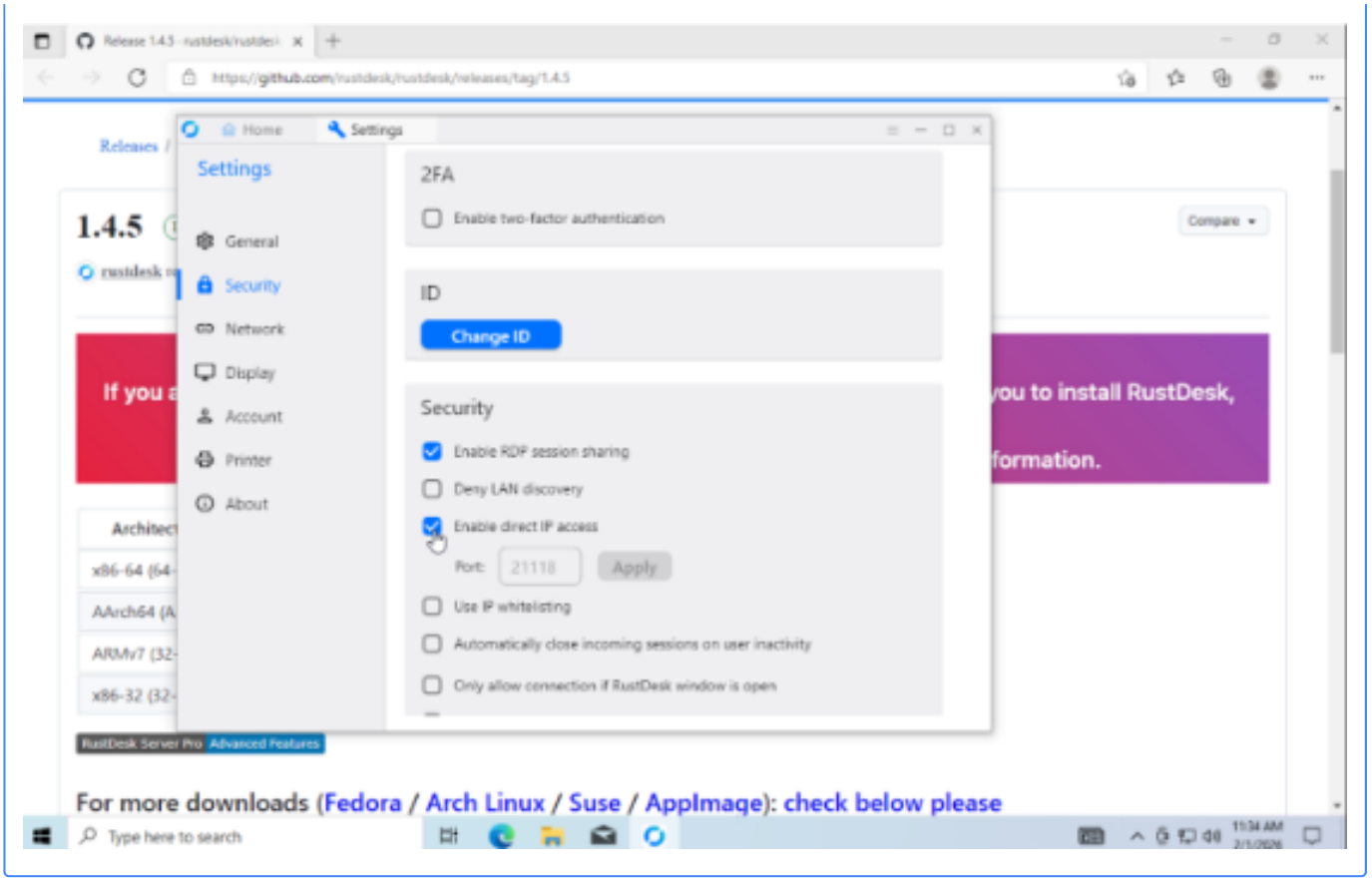
1. Go to **Security**



1. Click **"Unlock security settings"**
2. Scroll to **"Use Permanent Password"** and set your password



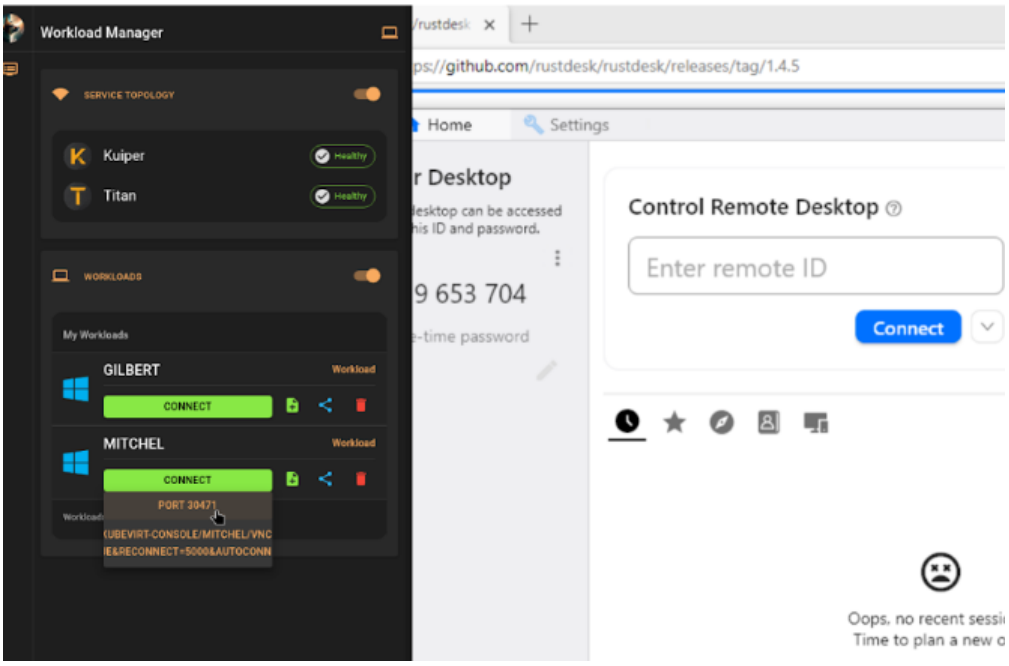
1. Scroll to **"Enable direct IP access"** and enable it



STEP 6: TEST THE CONNECTION

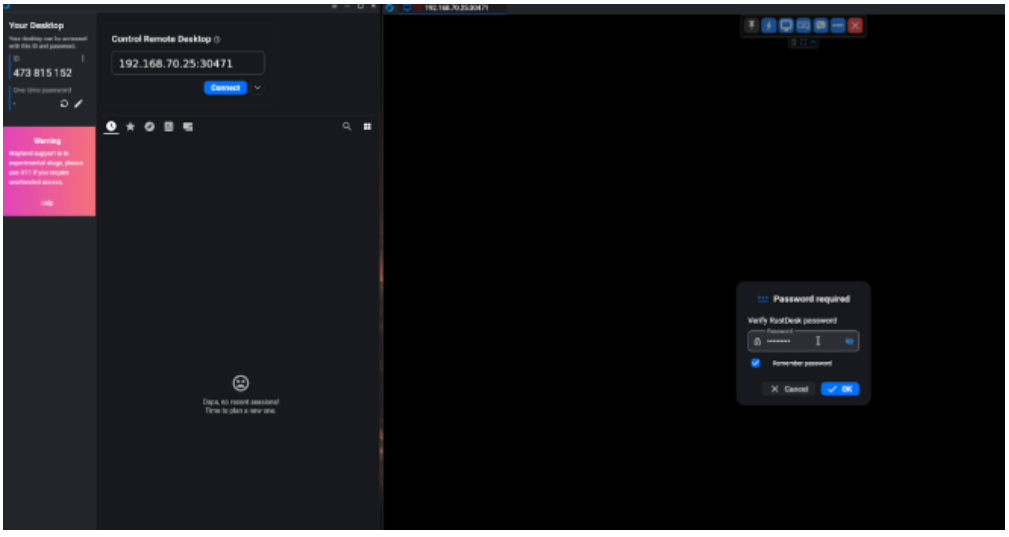
Connect to Restdesk via Port

1. In Orion, click the **Connect** button on your workload
2. Select the **Port** option and copy the port number



> **Tip**
>
> If you have Quick Bar enabled, hover over the workload sidebar icon to see active workloads and quickly copy connection info.

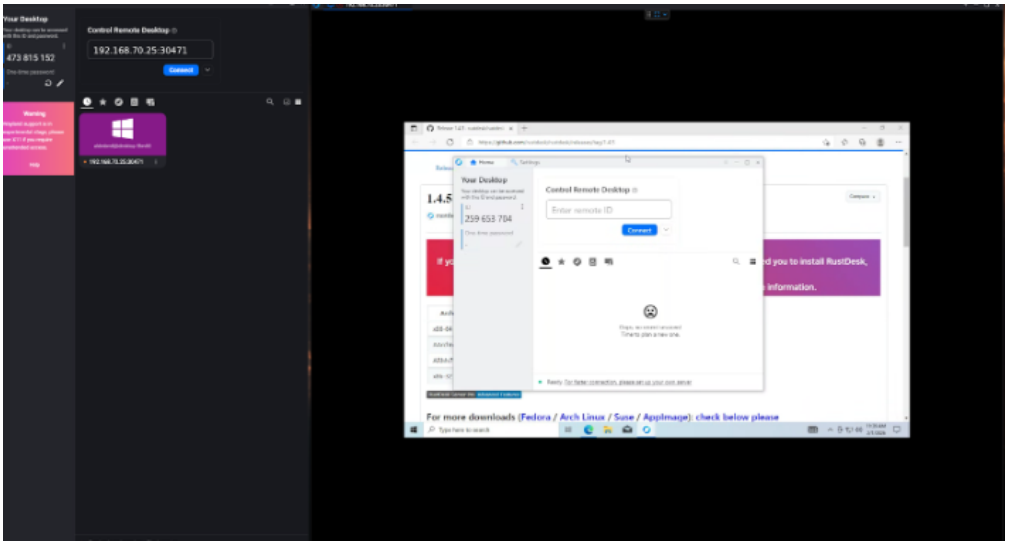
1. Open RustDesk on your **local machine**



Enter the connection: `<node-hostName>:<port> OR <node-ip>:<port>`

```
(e.g., superalpha.juno-innovations.com: 31234 `OR` `192.168.1.100:31234`)
```

1. Click **Connect** and enter your permanent password



USING OTHER VDI SOLUTIONS

The RustDesk setup demonstrated here follows the same pattern for other direct-connect VDI solutions. Whether you're using HP Anyware (RGS), Parsec, Nice DCV, or similar tools, the process is:

1. Install your VDI software inside the Windows VM
2. Configure it for direct IP access with a static/permanent password
3. Set up the appropriate port(s) in your Orion workload template
4. Use the **Port** option from the Connect button to get your assigned port
5. Connect from your local client using `<node-hostName>:<port> OR <node-ip>:<port>`

Orion's dynamic port assignment works as a connection broker for any VDI solution that supports direct connections — you're not locked into RustDesk.

Create a Golden Image

WHY GOLDEN IMAGES MATTER

A golden image is your "master copy" — every VM you deploy from it will be an exact clone. This means:

- **Everything you install gets replicated.** If you install RustDesk, Chrome, and your creative suite into the golden image, every VM spun up from it has those applications ready to go.
- **Everything you forget gets replicated too.** If you skip Windows updates, misconfigure a setting, or leave test files on the desktop, every VM inherits those issues.
- **Changes require a new golden image.** Need a different version of Houdini? Want to add a new application? You'll need to either update your golden image and redeploy, or install manually on each VM.

BEFORE YOU CLONE, ASK YOURSELF:

- Are all necessary drivers installed and working?
- Is my VDI solution (RustDesk, etc.) configured correctly?
- Have I installed the core applications my users need?
- Are Windows updates applied?
- Is the system configured the way I want it (display settings, power settings, etc.)?
- Have I removed any temporary files or test data?

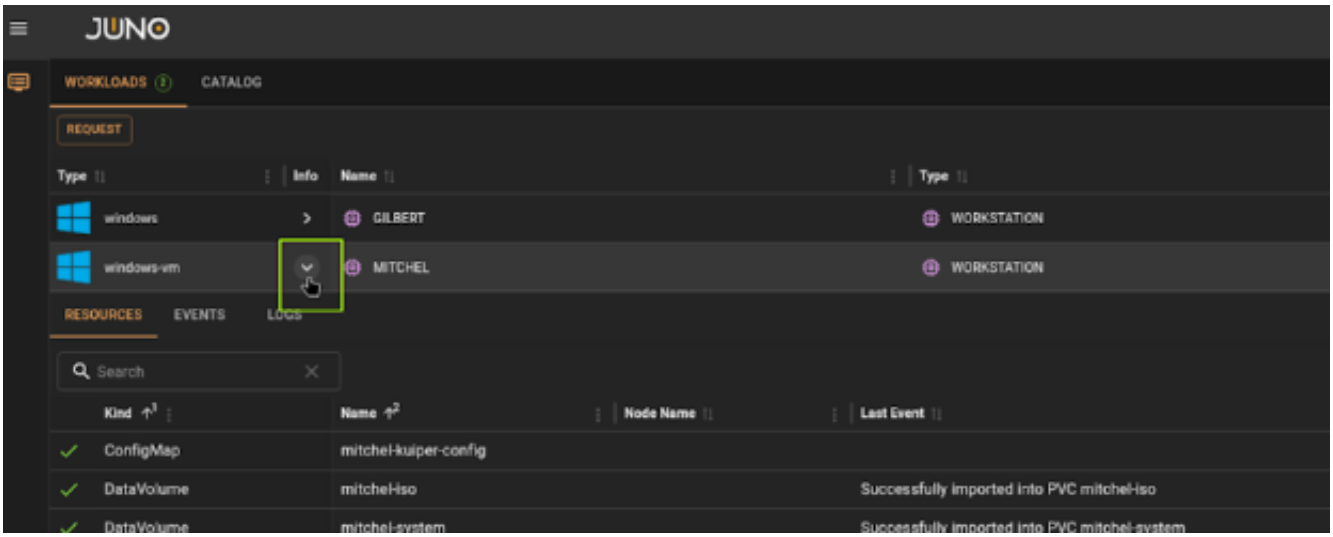
Taking an extra 15 minutes to get your golden image right saves hours of troubleshooting and redeployment later. Treat it like baking a master template — once it's in production, that's what everyone gets.

Now that Windows is configured with drivers and RustDesk, create a golden image for fast deployment of future VMs.

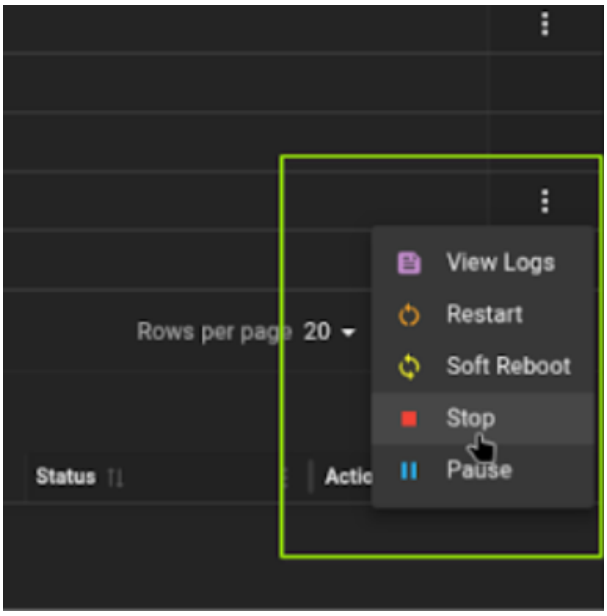
STEP 1: STOP THE VM

Find and stop VM resource

1. Open the **Resources** table on your VM workload



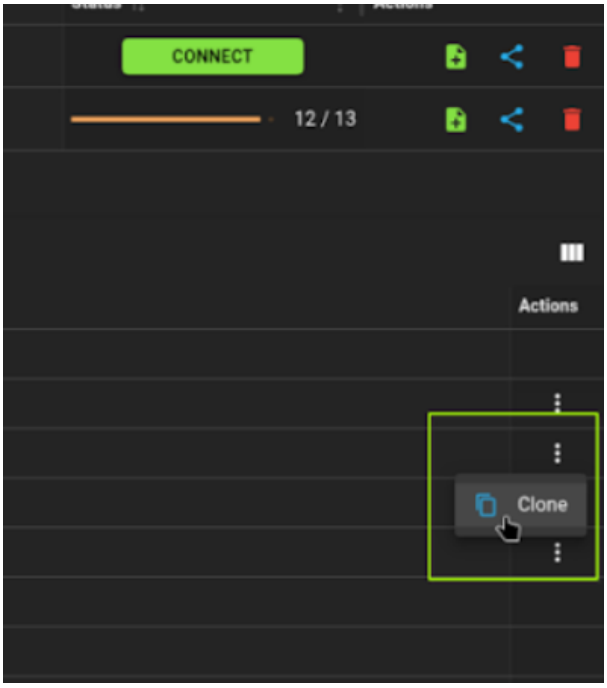
1. Find the **VirtualMachine** resource, open the actions menu and click **Stop**



STEP 2: CLONE THE SYSTEM DISK

Clone System Disk Data Volume

1. Find the **Data Volume** with "system" in the name. As you defined when creating your workload template
2. Click the **Actions** dropdown
3. Select **Clone**



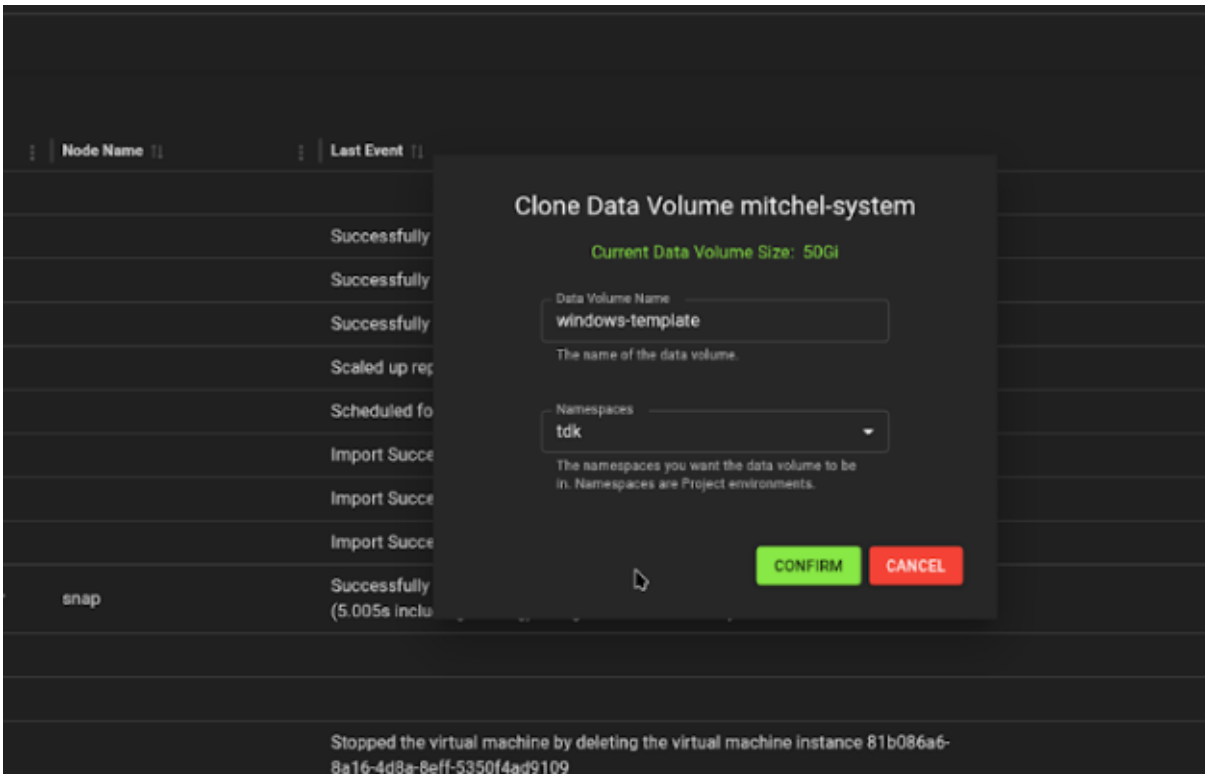
Tip

To learn more about our available actions, see our resource actions documentation

STEP 3: NAME YOUR GOLDEN IMAGE

Name Your Golden Image

1. Enter a name (e.g., `windows-template`)
2. Select the namespace you'd like the Data Volume to be cloned to
3. The size will be automatically detected
4. Click to start the clone



Note

Cloning can take several minutes depending on disk size. You will see a progress wheel providing you with the current clones progress percentage. You may need to refresh the page to see the updated status — we're aware this sometimes misses update cycles.

STEP 4: VERIFY THE CLONE

Once complete, your golden image appears in the list. This image contains:

- Windows with all updates applied during setup
- VirtIO drivers installed
- RustDesk configured and ready

Tip

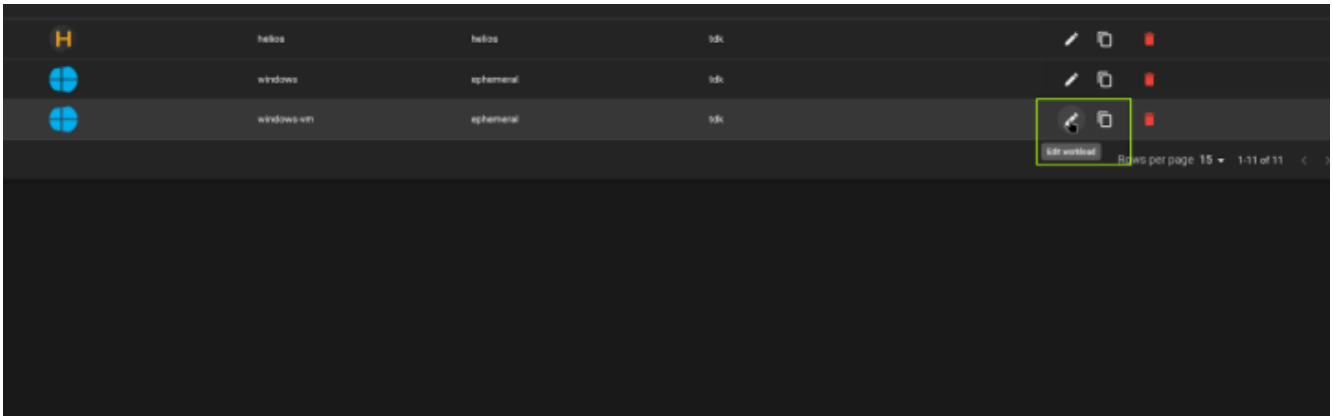
Cloned Data Volumes will also appear in the Genesis storage page in the Data Volumes tab.

Create a VM Template from Your Golden Image

STEP 1: CREATE THE WORKLOAD TEMPLATE

Edit or Duplicate the Workload Template

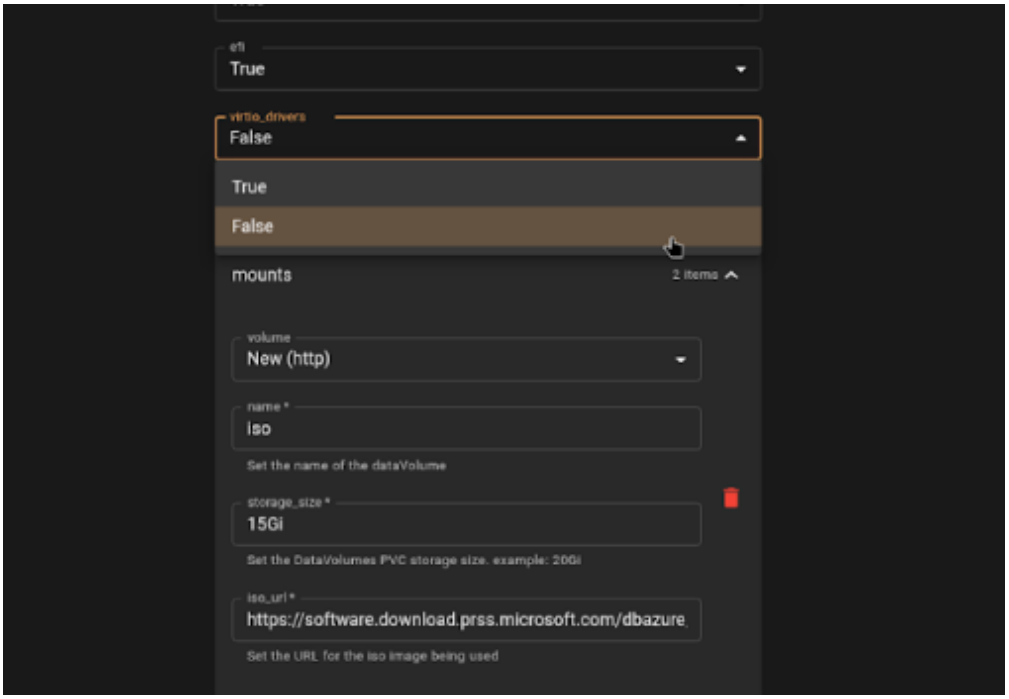
1. Return to Genesis → **Workloads** table
2. Either edit the original template or duplicate it first



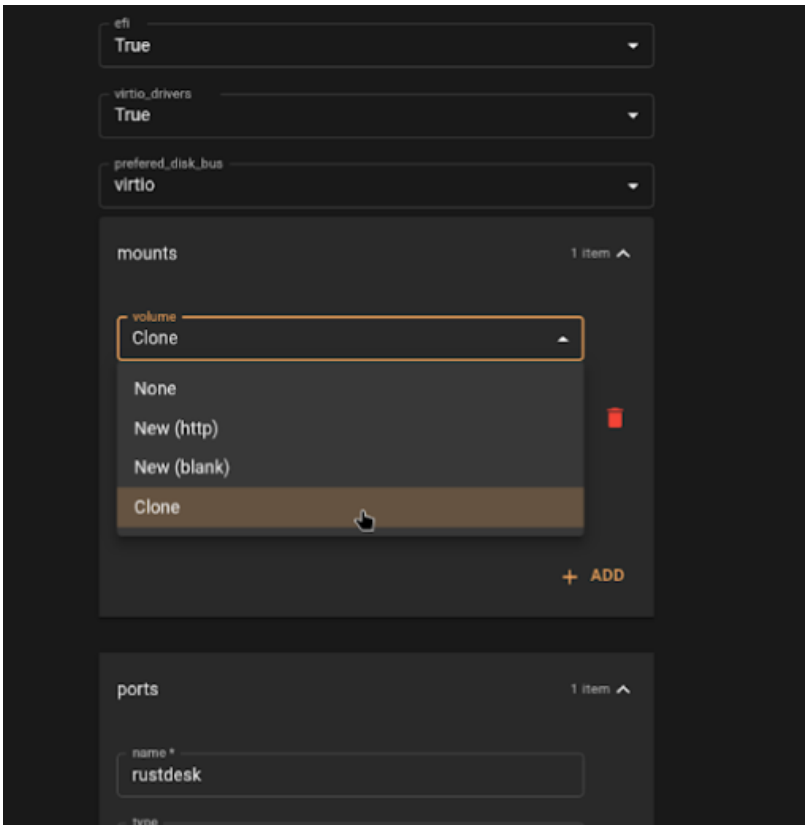
Update the Settings

Make the following changes:

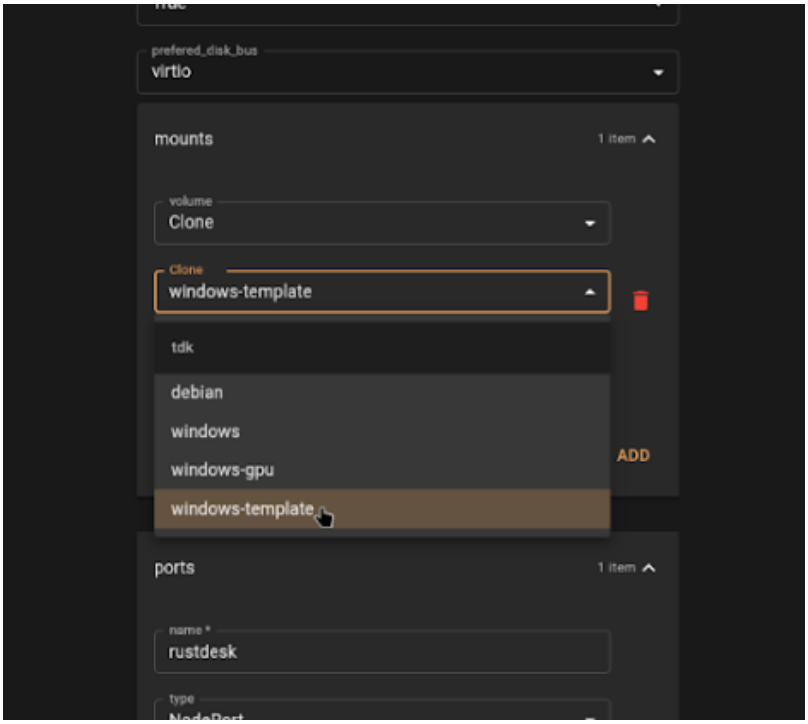
1. **virtio_drivers:** Set to `false` (drivers are now in the golden image)



1. Change the ISO mount from **HTTP** to **Clone**



1. Set **clone** to your golden image name (e.g., `windows-template`)



1. **Remove the blank system mount** (if still present)—the golden image includes the system disk

Submit Change

Click **Submit** to update the template.

You new Windows VM workload template built from your golden image is now ready to use. You can now navigate back to your project, and begin launching and connecting to your new workload template just as you did previously with your first Windows template. However now, you will not need to go through the install process, and can connect via Rustdesk as soon as the workload is ready.

GPU Passthrough

GPU PASSTHROUGH SETUP GUIDE

Debian 13 · VFIO · KubeVirt · RTX 2080

Overview

This guide walks through configuring an NVIDIA RTX 2080 GPU for VFIO passthrough on Debian 13, enabling GPU-accelerated virtual machines in KubeVirt environments. While written specifically for Debian 13, these instructions should work on other distributions with minor modifications.

What we're doing

1. Uninstalling NVIDIA drivers to expose the raw GPU hardware
2. Binding the GPU to VFIO drivers instead of native graphics drivers
3. Configuring the system to maintain VFIO bindings across reboots
4. Integrating with Juno's GPU operator for KubeVirt passthrough

Prerequisites

- Both KubeVirt and NVIDIA GPU Operator require VFIO drivers to pass devices to guest VMs
- The GPU must not be in use by the host OS (no native drivers loaded)
- IOMMU support must be enabled in BIOS and kernel

> **Info**
>
> If you're using the GPU directly on the host through the NVIDIA GPU Operator (without passthrough), the setup process is simpler and doesn't require VFIO configuration.

GPU Device Reference

The RTX 2080 (TU104 architecture) presents as four separate PCI functions. **All functions must be bound to VFIO for passthrough to work.**

PCI Address	Function	Device ID	Purpose
`0000:08:00.0`	VGA / GPU Core	`10de:1e82`	Primary graphics processor
`0000:08:00.1`	HD Audio	`10de:10f8`	HDMI/DisplayPort audio
`0000:08:00.2`	USB Controller	`10de:1ad8`	USB 3.1 host controller
`0000:08:00.3`	USB-C / UCM-UCSI	`10de:1ad9`	USB Type-C controller

> **Note**
>
> **Enterprise GPU Note:** Not all NVIDIA enterprise cards include audio and USB controllers. Consumer cards like the RTX 2080 typically have all four functions.

VERIFY PREREQUISITES

Step 1: Verify Initial GPU State

Before beginning, confirm the GPU is detected and identify its PCI addresses. We can do this by listing our NVIDIA devices, and checking our driver status.

rify Initial GPU State

List all NVIDIA devices:

```
lspci | grep -i nvidia
```

Check driver status:

```
nvidia-smi
```

Expected output for RTX 2080:

```
08:00.0 VGA compatible controller: NVIDIA Corporation TU104 [GeForce RTX 2080] (rev a1)
08:00.1 Audio device: NVIDIA Corporation TU104 HD Audio Controller (rev a1)
08:00.2 USB controller: NVIDIA Corporation TU104 USB 3.1 Host Controller (rev a1)
08:00.3 Serial bus controller: NVIDIA Corporation TU104 USB Type-C UCSI Controller (rev a1)
```

mportant

Record these PCI addresses — you'll use them throughout this guide. **Replace `08:00.x` with your actual addresses in all subsequent commands.**

Step 2: Verify IOMMU Is Enabled

IOMMU (Input-Output Memory Management Unit) must be enabled for device passthrough to function. We will need to check our current kernel via the command line, and ensure the necessary flags are present.

rify IOMMU Is Enabled

Check current kernel command line:

```
cat /proc/cmdline
```

Look for one of these flags: - `intel_iommu=on` — Intel CPUs - `amd_iommu=on` — AMD CPUs

If neither flag is present:

1. Edit GRUB configuration:

```
sudo nano /etc/default/grub
```

2. Add the appropriate flag to `GRUB_CMDLINE_LINUX_DEFAULT`:

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash intel_iommu=on"
```

3. Update GRUB and reboot:

```
sudo update-grub
sudo reboot
```

SETUP VFIO

Step 1: Load VFIO Kernel Modules

VFIO (Virtual Function I/O) provides the framework for safe device passthrough to virtual machines. We will want to load our modules for the current session and then ensure they load automatically on boot.

> **Important**
>
> This persistence configuration is critical. If VFIO bindings don't survive reboots, it's usually because this step was skipped.

ad Modules for current Session

```
sudo modprobe vfio
sudo modprobe vfio_pci
sudo modprobe vfio_iommu_type1
```

ke modules load automatically on boot

```
cat <<EOF | sudo tee /etc/modules-load.d/vfio.conf
vfio
vfio_pci
vfio_iommu_type1
EOF
```

Step 2: Identify GPU PCI Functions and Device IDs

Locate all PCI functions and their vendor:device IDs for your GPU.

t NVIDIA Devices with device IDs

List Devices:

```
lspci -nn | grep -i nvidia
```

Expected output (TU104/RTX 2080):

```
08:00.0 VGA compatible controller [0300]: NVIDIA Corporation TU104 [GeForce RTX 2080] [10de:1e82] (rev a1)
08:00.1 Audio device [0403]: NVIDIA Corporation TU104 HD Audio Controller [10de:10f8] (rev a1)
08:00.2 USB controller [0c03]: NVIDIA Corporation TU104 USB 3.1 Host Controller [10de:1ad8] (rev a1)
08:00.3 Serial bus controller [0c80]: NVIDIA Corporation TU104 USB Type-C UCSI Controller [10de:1ad9] (rev a1)
```

The device IDs are shown in brackets: `[10de:1e82]`, `[10de:10f8]`, etc.

> **Important**
>
> Remember to Replace 08:00.x with your actual addresses in all subsequent commands.

rify current kernel driver

Verify current kernel driver:

```
lspci -v -s 08:00.0
```

Example output:

```
08:00.0 VGA compatible controller: NVIDIA Corporation TU104 [GeForce RTX 2080] (rev a1)
    Subsystem: eVga.com. Corp. Device 2081
    Flags: bus master, fast devsel, latency 0, IRQ 74, IOMMU group 14
    Memory at fb000000 (32-bit, non-prefetchable) [size=16M]
    Memory at d0000000 (64-bit, prefetchable) [size=256M]
    Memory at e0000000 (64-bit, prefetchable) [size=32M]
    I/O ports at d000 [size=128]
    Expansion ROM at 000c0000 [disabled] [size=128K]
    Capabilities: <access denied>
    Kernel driver in use: nouveau
    Kernel modules: nouveau
```

Driver status check:

- If **"Kernel driver in use: nouveau"** or another driver → Continue to Step 4
- If **"Kernel driver in use: vfio-pci"** → Skip to Step 6 (already configured)

Step 3: Identify the IOMMU Group

All devices in an IOMMU group must be passed through together or bound to VFIO.

ıd the IOMMU Group Number

Find group number:

```
readlink /sys/bus/pci/devices/0000:08:00.0/iommu_group
```

Example output:

```
../../../kernel/iommu_groups/14
```

This GPU is in IOMMU group 14.

t all devices in the group

List all devices in the group:

```
for d in /sys/kernel/iommu_groups/<insert-group-number>/devices/*; do
  lspci -nnk -s "$(basename "$d")"
done
```

What to check:

- All four GPU functions should be in the same group
- Ideally, no unrelated devices share this group
- If unrelated devices are present, you may need ACS override patches (advanced topic)

If "Kernel driver in use: vfio-pci" for all devices → Skip to Step 6

Step 4: Bind All GPU Functions to vfio-pci

Manually bind each PCI function to the VFIO driver. **Run these commands as root** or with `sudo`.

Important

Critical: Replace `08:00.x` with your actual PCI addresses from Step 3.

A Core (08:00.0)

```
echo vfio-pci > /sys/bus/pci/devices/0000:08:00.0/driver_override
echo 0000:08:00.0 > /sys/bus/pci/drivers/vfio-pci/bind
```

Audio (08:00.1)

```
echo vfio-pci > /sys/bus/pci/devices/0000:08:00.1/driver_override
echo 0000:08:00.1 > /sys/bus/pci/drivers/vfio-pci/bind
```

B Controller (08:00.2)

Note: Unbind from `xhci_hcd` driver first if present.

```
echo vfio-pci > /sys/bus/pci/devices/0000:08:00.2/driver_override
echo 0000:08:00.2 > /sys/bus/pci/drivers/xhci_hcd/unbind 2>/dev/null || true
echo 0000:08:00.2 > /sys/bus/pci/drivers/vfio-pci/bind
```

B-C / UCM-UCSI (08:00.3)

```
echo vfio-pci > /sys/bus/pci/devices/0000:08:00.3/driver_override
echo 0000:08:00.3 > /sys/bus/pci/drivers/vfio-pci/bind
```

Step 5: Verify VFIO Binding

Confirm all GPU functions are now using the VFIO driver.

-check the IOMMU group

Check IOMMU group:

```
for d in /sys/kernel/iommu_groups/14/devices/*; do
  lspci -nnk -s "$(basename "$d")"
done
```

Each device must show:

```
Kernel driver in use: vfio-pci
```

If any device shows a different driver (e.g., `xhci_hcd`, `nouveau`):

- Manually unbind it: `echo 0000:XX:XX.X > /sys/bus/pci/drivers/<driver_name>/unbind`
- Then bind to vfio-pci: `echo 0000:XX:XX.X > /sys/bus/pci/drivers/vfio-pci/bind`

Step 6: Make VFIO Binding Persistent

The manual bindings from Step 5 won't survive a reboot. We need to configure the kernel to automatically bind these devices to VFIO.

thod 1: GRUB Kernel Parameters (Recommended)

1. Edit GRUB configuration:

```
sudo nano /etc/default/grub
```

2. Add all device IDs to the kernel command line:

Find the line starting with `GRUB_CMDLINE_LINUX_DEFAULT` and add the `vfio-pci.ids` parameter:

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash intel_iommu=on vfio-pci.ids=10de:1e82,10de:10f8,10de:1ad8,10de:1ad9"
```

Device ID mapping (from Step 3):

- `10de:1e82` → VGA Core
- `10de:10f8` → HD Audio
- `10de:1ad8` → USB Controller
- `10de:1ad9` → USB-C Controller

3. Update GRUB and rebuild initramfs:

```
sudo update-grub
sudo update-initramfs -u
```

4. Reboot to apply changes:

```
sudo reboot
```

If kernel parameters don't work, use a systemd service to bind devices at boot.

thod 2: Systemd Service (Optional Backup)

1. Create the service file:

```
sudo nano /etc/systemd/system/vfio-pci-bind.service
```

2. Paste this content (adjust PCI addresses):

```
[Unit]
Description=Bind RTX 2080 PCI devices to vfio-pci
After=local-fs.target
Requires=local-fs.target

[Service]
Type=oneshot
ExecStart=/bin/bash -c 'for dev in 0000:08:00.0 0000:08:00.1 0000:08:00.2 0000:08:00.3; do \
  echo vfio-pci > /sys/bus/pci/devices/$dev/driver_override; \
  echo $dev > /sys/bus/pci/drivers/vfio-pci/bind 2>/dev/null || true; \
done; \
echo 0000:08:00.2 > /sys/bus/pci/drivers/xhci_hcd/unbind 2>/dev/null || true'
RemainAfterExit=yes

[Install]
WantedBy=multi-user.target
```

3. Enable and start the service:

```
sudo systemctl daemon-reload
sudo systemctl enable vfio-pci-bind.service
sudo systemctl start vfio-pci-bind.service  # Optional: test without reboot
```

Step 7: Confirm Configuration After Reboot

After the system restarts, verify VFIO bindings persisted correctly.

eck all GPU functions

Check all GPU functions:

```
lspci -nnk | grep -A3 -E "08:00\."
```

All four functions should report:

```
Kernel driver in use: vfio-pci
```

If bindings didn't persist:

1. Check kernel command line: `cat /proc/cmdline` (should show `vfio-pci.ids=...`)
2. Verify initramfs was rebuilt: `ls -lh /boot/initrd.img-$(uname -r)`
3. Check systemd service status: `systemctl status vfio-pci-bind.service`

Host Setup Complete

At this point:

- The RTX 2080 is bound to VFIO drivers
- The host OS no longer uses the GPU for graphics
- The entire IOMMU group is viable for passthrough
- KubeVirt can now attach this GPU to virtual machines

Next steps:

Configure Juno's GPU operator and create KubeVirt VMs with GPU passthrough.

USING GPU PASSTHROUGH

Using GPU Passthrough in Virtual Machines

Now that you've installed the NVIDIA GPU operator, and your nodes have been configured for GPU passthrough and VFIO, we can attach our GPU to a KubeVirt VM.

> **Info**
>
> If you haven't installed the GPU operator, or configured your nodes for GPU passthrough you can see our guide here

Attach GPU to a KubeVirt VM Workload

1. Navigate to **Workloads** in Juno
2. Select your VM template
3. In the **GPU** section, you'll see a True/False toggle
4. Enable GPU passthrough
5. Enter the exact device name from the allocatable resources:
6. Example: `nvidia.com/TU104_GEFORCE_RTX_2080`

GPU allocation rules

- **Currently:** Each VM can request exactly **one GPU**
- **Multiple GPUs:** Multiple GPU passthrough per VM is coming soon (TBD)
- **Device name:** Must match the exact string from `kubectl get nodes` output

TROUBLESHOOTING

Troubleshooting

Common Issues

GPU not showing in allocatable resources:

- Verify VFIO binding: `lspci -nnk -s 08:00.0` should show `vfio-pci`
- Check GPU Operator logs: `kubectl logs -n gpu-operator-resources <pod-name>`
- Ensure `vm-passthrough` label is set on the correct node

IOMMU group contains other devices:

- You may need ACS override patches (advanced)
- Consider passing through the entire IOMMU group
- Consult your motherboard documentation for IOMMU configuration

Bindings don't persist after reboot:

- Verify kernel parameters: `cat /proc/cmdline | grep vfio-pci.ids`
- Check initramfs date: `ls -lh /boot/initrd*`
- Enable systemd service as backup: `systemctl enable vfio-pci-bind.service`

VM fails to start with GPU:

- Check VM definition includes correct device name
- Verify host has available GPU resources: `kubectl describe node <node-name>`
- Review KubeVirt logs: `kubectl logs -n kubevirt <virt-launcher-pod>`

Verification Commands Reference

```
# Check IOMMU enabled
cat /proc/cmdline | grep iommu

# List VFIO modules
lsmod | grep vfio

# Check device driver
lspci -nnk -s 08:00.0

# View IOMMU groups
find /sys/kernel/iommu_groups/ -type l

# Check node allocatable resources
kubectl get nodes -o json | jq '.items[].status.allocatable'

# GPU Operator status
kubectl get pods -n gpu-operator-resources
kubectl describe node <node-name> | grep -A10 Allocatable
```

Additional Resources

- **VFIO Documentation:** kernel.org/doc/vfio
- **KubeVirt GPU Passthrough:** kubevirt.io
- **NVIDIA GPU Operator:** docs.nvidia.com/datacenter/cloud-native
- **Juno Documentation:** Contact your Juno representative for the latest integration guides

---

Document Version: 1.0
Last Updated: February 2025
Tested On: Debian 13, RTX 2080, KubeVirt 1.x, NVIDIA GPU Operator 24.x

Best Practices

Golden Image Management

- **Create purpose-specific golden images** (e.g., one for users with creative or technical apps, one for development)
- **Document what's installed** in each golden image
- **Refresh golden images** periodically with Windows updates
- **Test thoroughly** before deploying widely

Resource Planning

- **Plan disk sizes upfront** — you cannot resize, but you can add additional drives
- **Leave headroom** on your cluster — VMs need dedicated resources
- **Monitor node capacity** when deploying multiple VMs

What NOT to Do

- **Don't use VNC for daily work**—it's for maintenance only
- **Don't set resources to "unlimited"**—always use explicit limits
- **Don't deploy to production clusters** during beta
- **Don't expect VM migration** yet—VMs stay on their initial node

3.3.2 Cloud

Juno Orion AWS EC2 Instances via Crossplane

Managing your AWS EC2 instances has never been easier with Orion and Crossplane.

WHAT IS CROSSPLANE?

Crossplane extends Kubernetes to manage cloud infrastructure including AWS EC2 instances. Allowing you to run AWS EC2 instances alongside containers.

GETTING STARTED

In this guide we will walk you through the following:

- Installing Crossplane, the Crossplane AWS provider, and our Crossplane EC2 workload
- Launch and connect to an EC2 workload template

If you are new to Orion, please see our getting started docs before following this guide.

STEP 1: INSTALL THE CROSSPLANE PLUGIN

Start by installing the Crossplane plugin from the Terra Official Plugins repo.

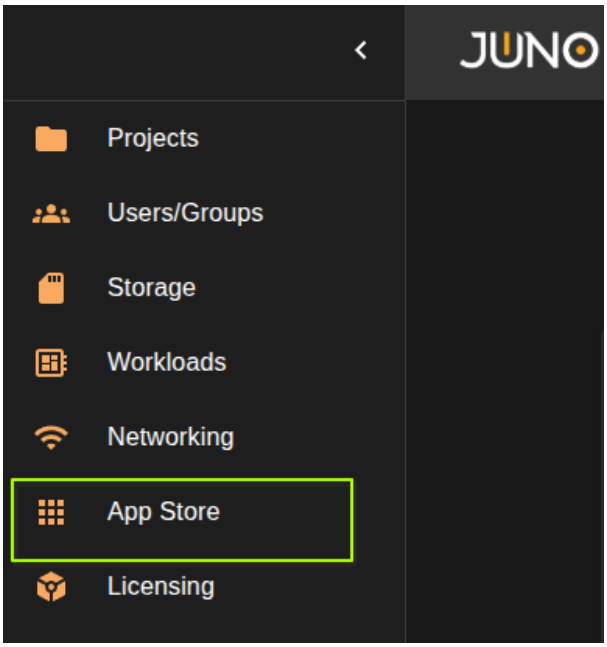
Important

Install this plugin FIRST. The provider and workload plugins require Crossplane to be installed.

Note

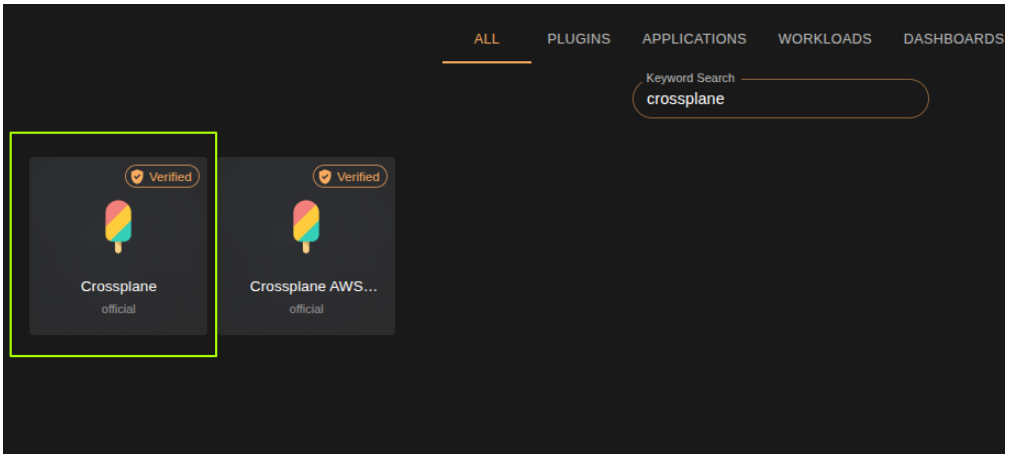
If you already have Crossplane installed you can skip this step

Navigate to the Terra App Store

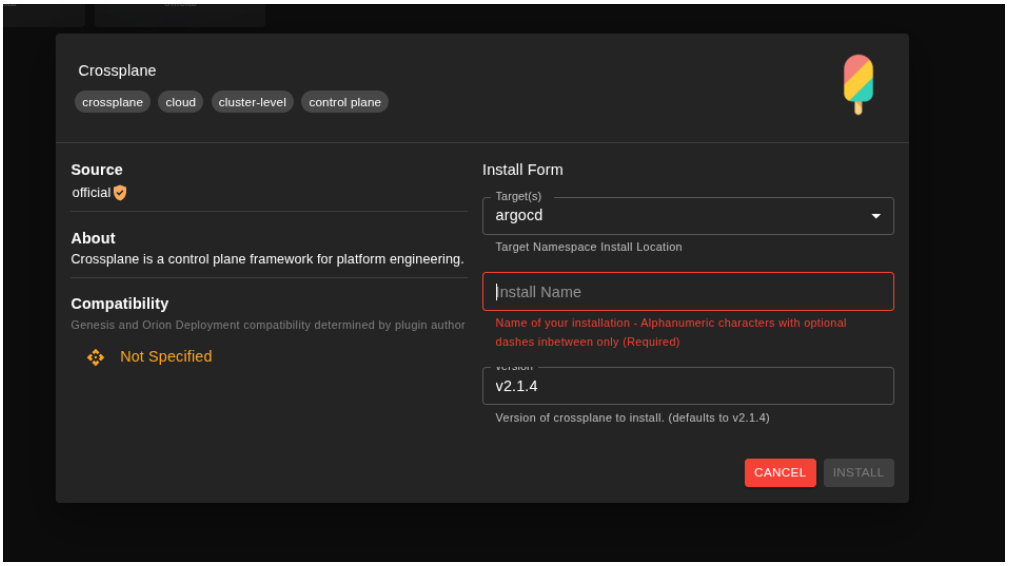


Install Crossplane

Search for the Crossplane plugin in the Terra App Store.



Once found, select the plugin, fill out the form and click **Install**



STEP 2: INSTALL THE CROSSPLANE AWS PROVIDER PLUGIN

This plugin installs the AWS kubernetes Provider and ProviderConfig, providing a way for Crossplane to communicate with your AWS account. The AWS Provider will include all the AWS Kubernetes API groups, and the ProviderConfig will store your AWS credentials in a secret, which will then be referenced when making calls to your AWS account.

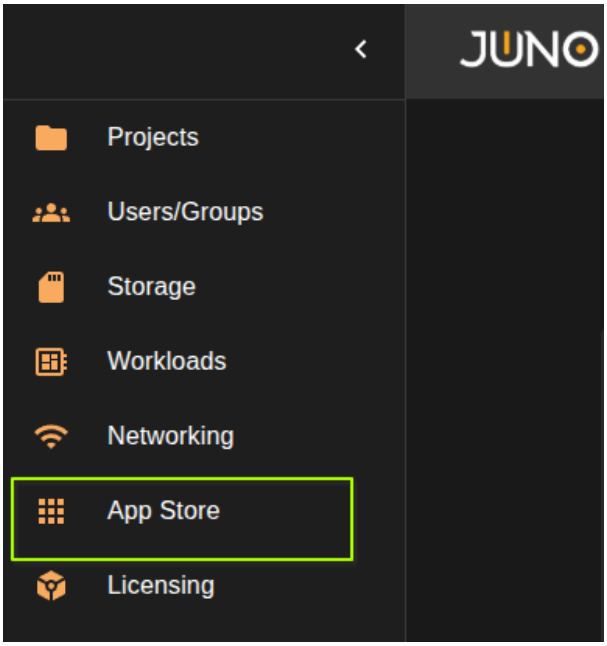
Important

Install this plugin SECOND. It requires Crossplane to be installed first.

Note

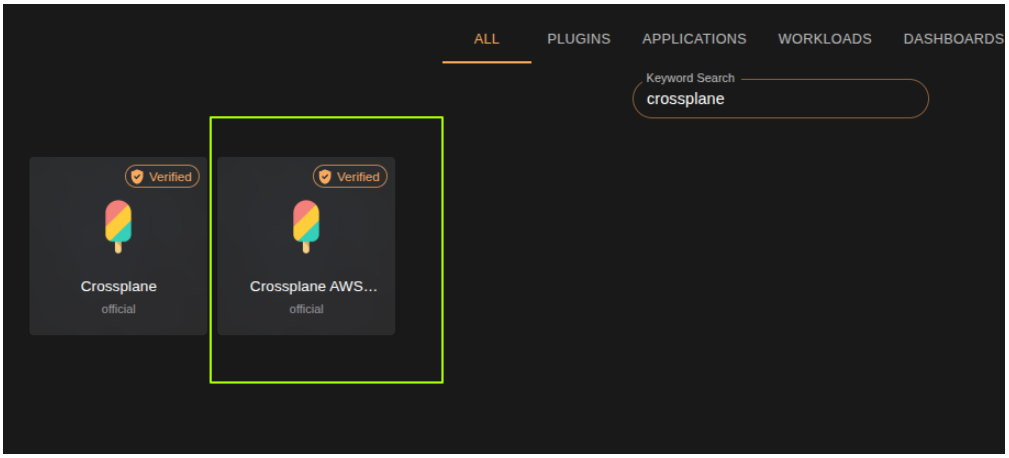
If you already have a Crossplane AWS provider installed you can skip this step

Navigate to the Terra App Store

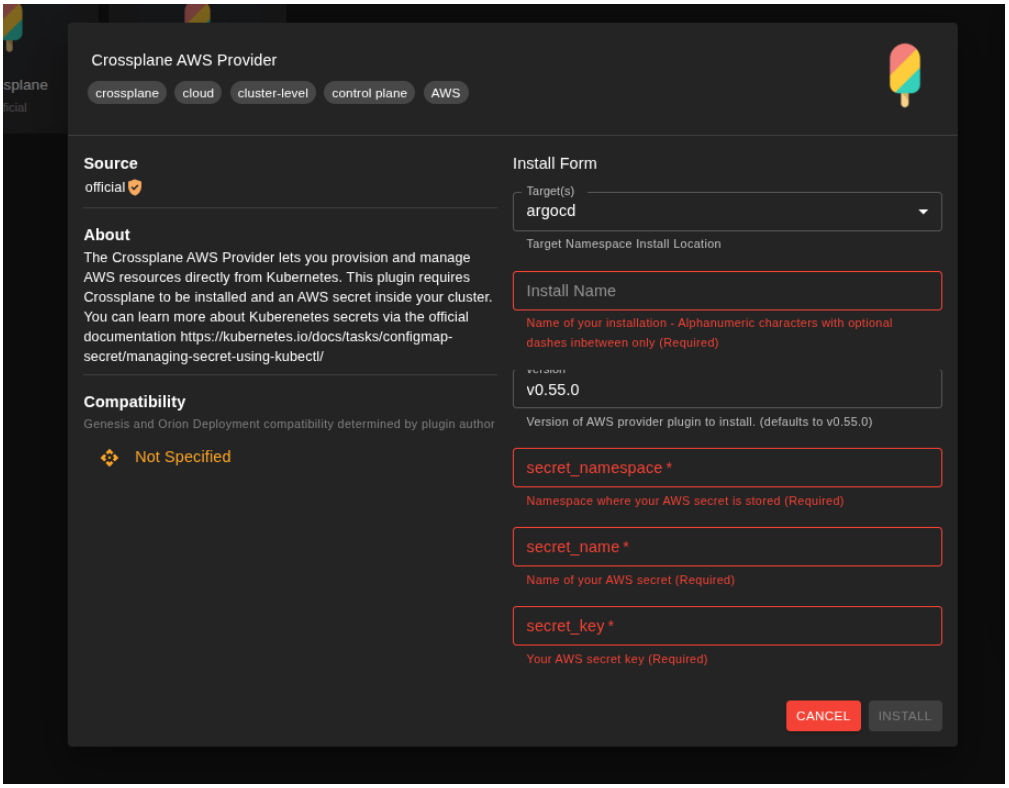


Install Crossplane AWS Provider

Search for the Generic Ephemeral VM plugin in the Terra App Store.

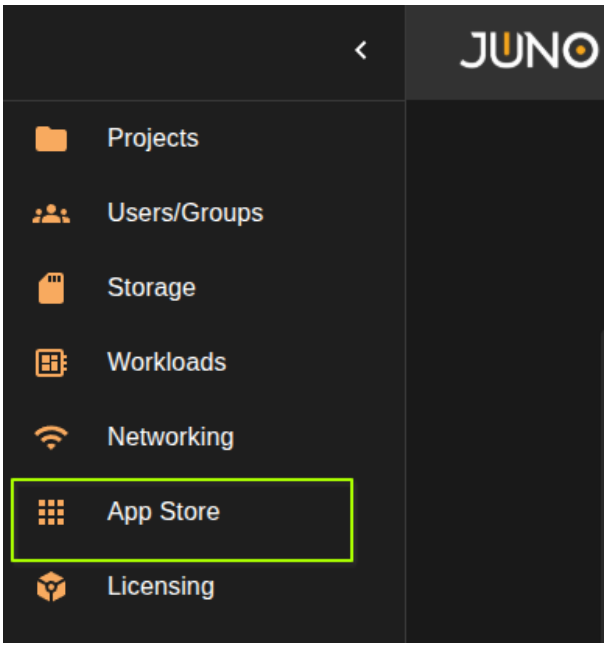


Once found, select the plugin, fill out the form and click **Install**



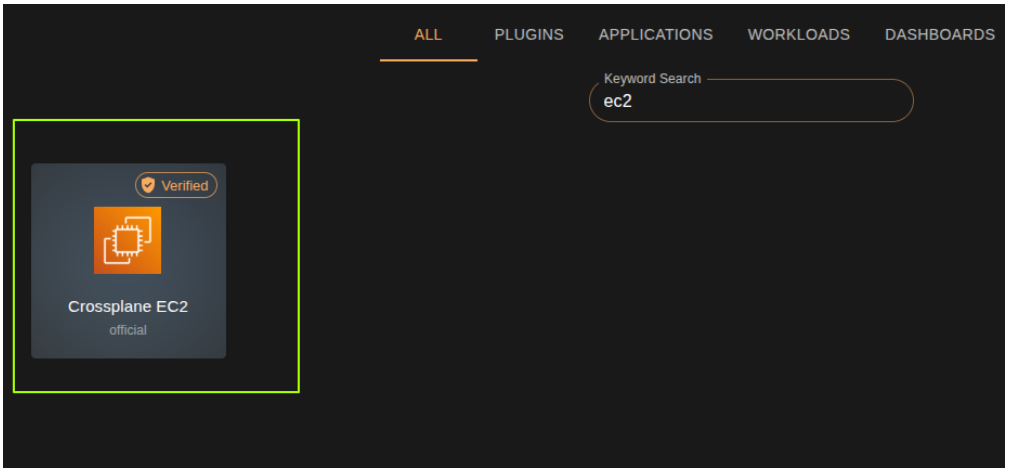
STEP 3: INSTALL THE CROSSPLANE EC2 WORKLOAD PLUGIN

Navigate to the Terra App Store

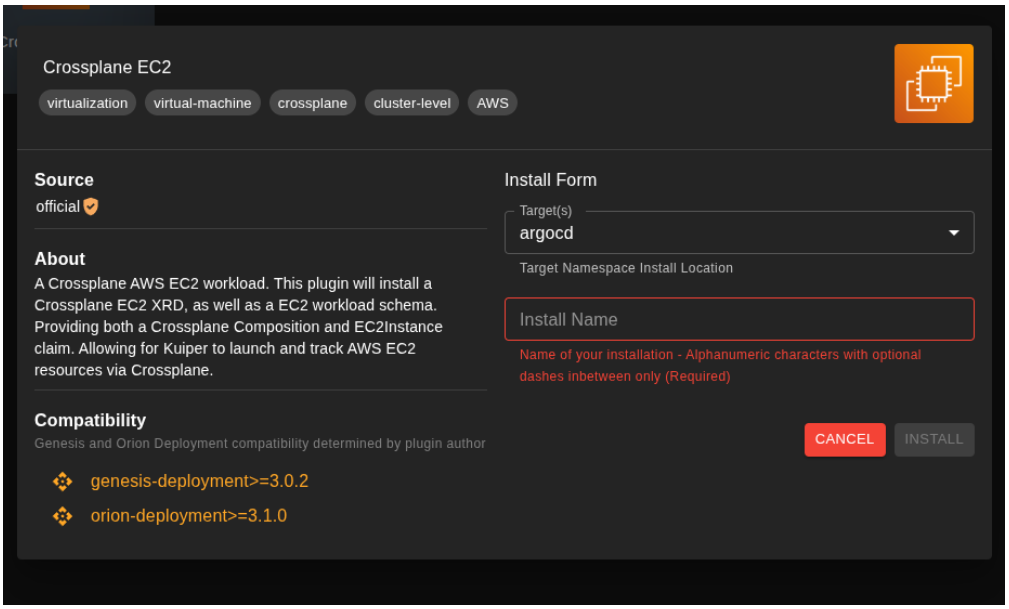


Install Crossplane EC2

Search for the Crossplane EC2 plugin in the Terra App Store.



Once found, select the plugin, fill out the form and click **Install**



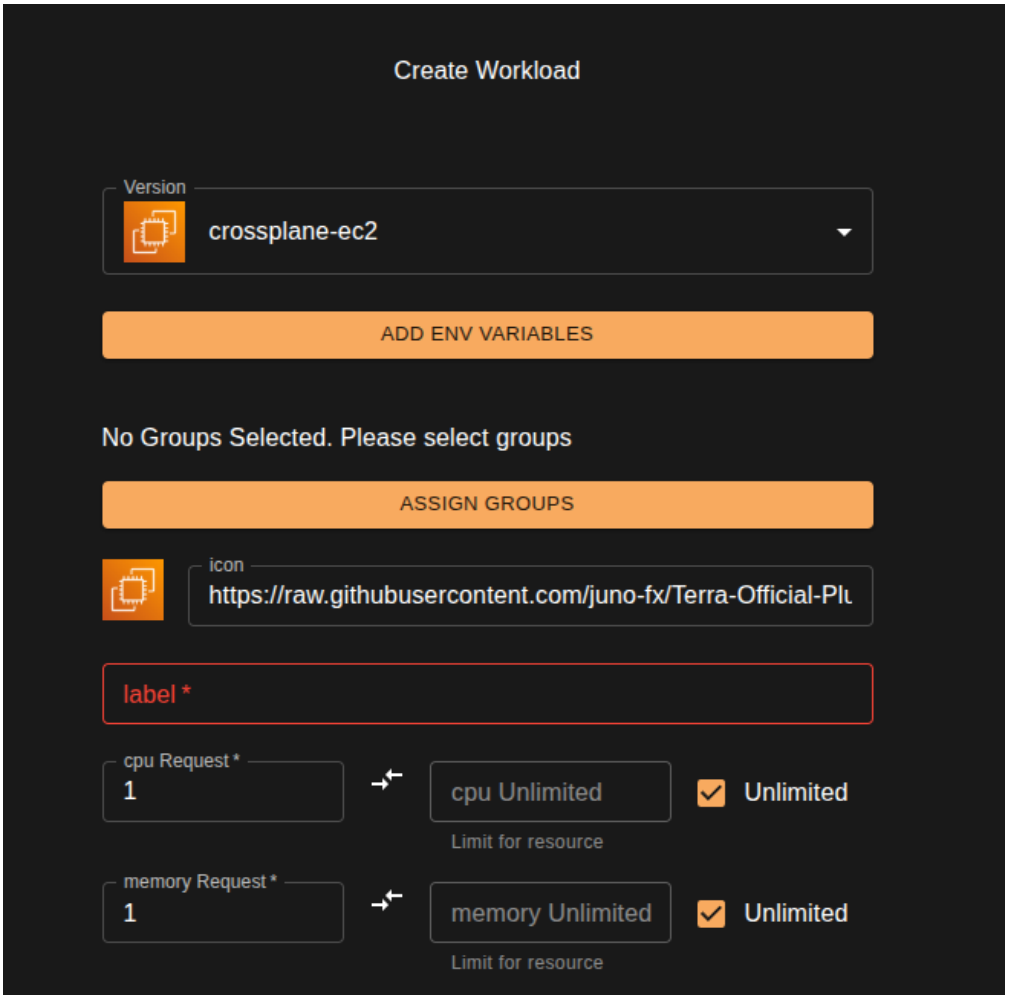
STEP 5: CREATE THE EC2 WORKLOAD TEMPLATE

Navigate to the Workload template Creation Form

1. From the Genesis **Workloads** table, click **Create Workload**
2. Select the **EC2 schema** you installed in the previous step.

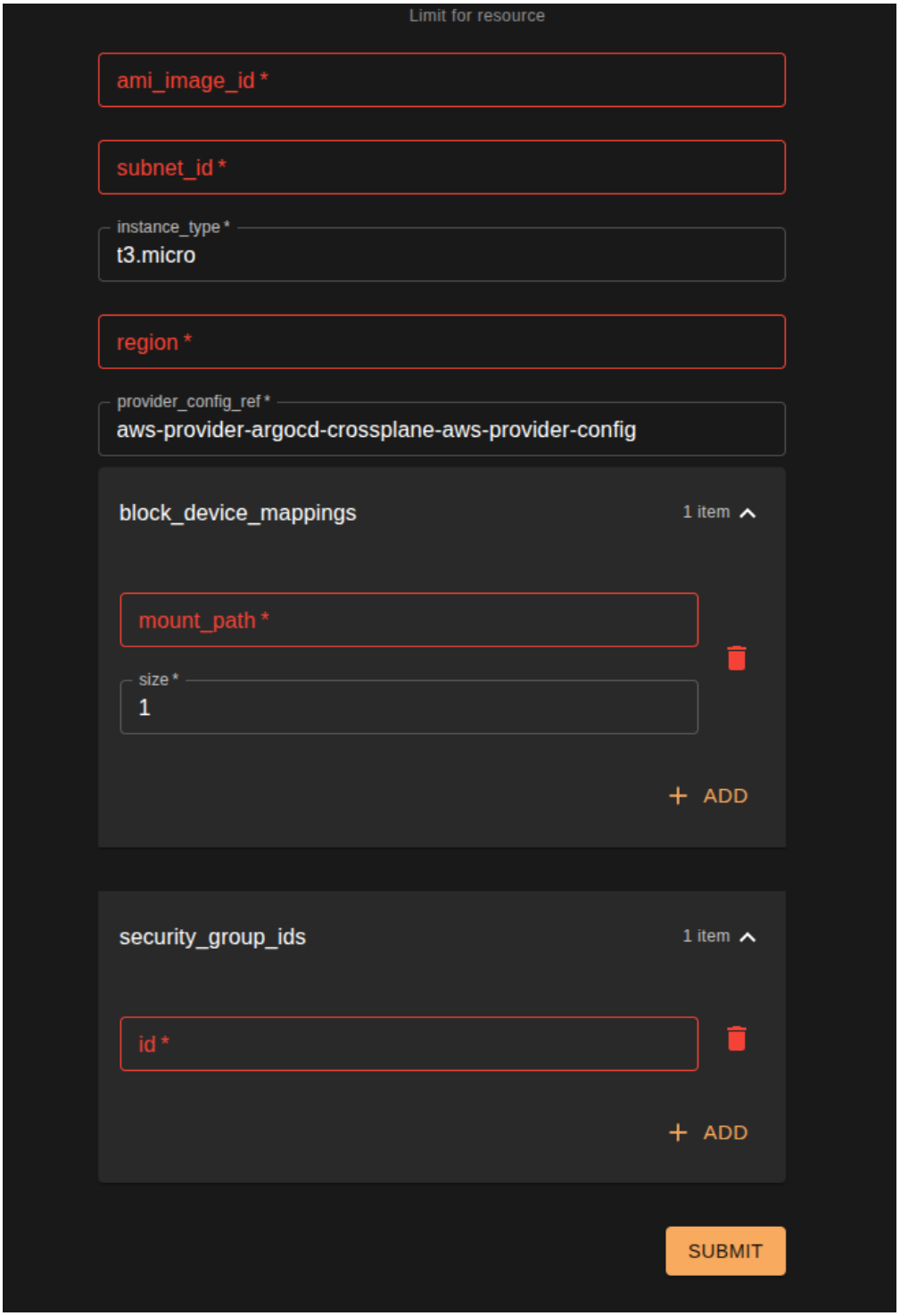
Fill Out the Standard Fields

Fill out the standard workload fields, and ensure you assign the template to a project group.



Fill Out the AWS Specific Fields

Fill out the AWS EC2 specific fields. Once complete click submit



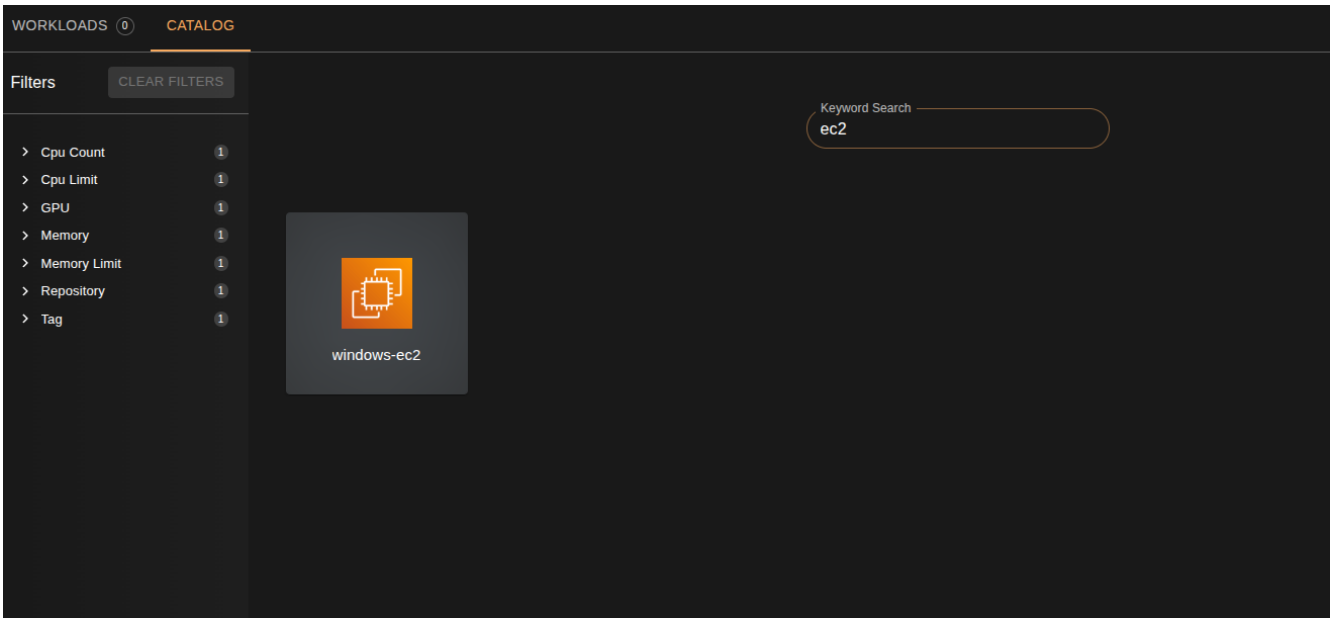
Important

Please ensure your AWS security groups allow connections from a user to the EC2 instance

STEP 5: LAUNCH YOUR WORKLOAD

Launch Workload

1. Navigate to the project you have assigned your new EC2 Workload.
2. Navigate to the **Workloads** table in Hubble
3. Click **Request** to open the catalog
4. Find your EC2 workload and click **Create**



STEP 6: CONNECT TO THE EC2 INSTANCE

Once the workload shows as ready, click the Connect button. For this particular workload you will be presented with dialog box showing you the connection information.

3.4 Recommended Terra Plugins

3.4.1 NVIDIA GPU Operator

Juno integrates with NVIDIA's official GPU Operator to manage GPU resources in Kubernetes.

Step 1: Install the NVIDIA GPU Operator

stall GPU Operator using the latest version

- Navigate to **Genesis → App Store**
- Install the official **NVIDIA GPU Operator**
- Use latest stable CRD and kernel modules

Step 2: Configuring Nodes for GPU

We will want to configure our nodes to handle our GPU. This can be done via our nodes labels. We can adjust our labels per node to specify if the GPU should be utilized for one of the following:

- Virtual Machine Passthrough
- MIG
- Timeslicing

Tip

For virtual machine passthrough, ensure your gpu has been configured properly. You can follow our guide here

Navigate to Node Labels Table

1. Go to **Networking** tab
2. Click **Node Labels** (top right)

Update Node Labels

1. Create a new label:
- **Key:** `nvidia.com/gpu.workload.config`
- **Value:** `vm-passthrough` || `timeslicing` || `mig`
2. Click **Select Servers**
3. Select all servers with GPUs already configured

Critical

If utilizing vm-passthrough ensure the GPU has been configured for VFIO passthrough. If you haven't done this you can see our guide for setting up VM GPU passthrough here

VERIFY GPU OPERATOR DEPLOYMENT

Ensure Sandbox Components are Running

Gather our sandbox pods:

```
kubectl get pods -A | grep sandbox
```

Expected output:

```
testing-argocd-gpu-operator   nvidia-sandbox-device-plugin-daemonset-67mvp    1/1   Running   0   19h
testing-argocd-gpu-operator   nvidia-sandbox-validator-jqw9d                  1/1   Running   0   19h
```

Confirm GPU Detection

Verify the GPU is listed as an allocatable resource:

```
kubectl get nodes <node-name> -o json | jq '.status.allocatable'
```

Expected output:

```
{
  "cpu": "16",
  "devices.kubevirt.io/kvm": "1k",
  "devices.kubevirt.io/tun": "1k",
  "devices.kubevirt.io/vhost-net": "1k",
  "ephemeral-storage": "442866475890",
  "hugepages-1Gi": "0",
  "hugepages-2Mi": "0",
  "memory": "32790116Ki",
  "nvidia.com/TU104_GEFORCE_RTX_2080": "1",
  "nvidia.com/TU104_HD_AUDIO_CONTROLLER": "0",
  "nvidia.com/gpu": "0",
  "pods": "110"
}
```

Key indicator: `"nvidia.com/TU104_GEFORCE_RTX_2080": "1"` confirms the GPU is registered and available for passthrough.

Info

Juno's UI doesn't currently list all allocatable resources in the dashboard. This feature will be released soon. Use the command line in the meantime.

If coming from our KubeVirt VM GPU passthrough guide, you can continue to the next step here

3.4.2 Certificate Manager

Certificates

The default installation of Juno products are deployed with a self signed certificate and will be flagged as untrusted by most browsers. We recommend replacing this certificate and installing the Cert Manager plugin available through the Terra plugin app store.

The Cert Manager plugin installs the third party software cert-manager which provides a easy to use service to automate certificate creation for your configured ingresses.

Setup

The below is an example setup of cert-manager in Orion using a custom CA, see the cert-manager website for further more detailed information on various deployment settings. We will be using a CA file and key in the example, if you wish to use an external ACME provider such as Lets Encrypt then please refer to the cert-manager docs themselves.

REQUIREMENTS

- A CA certificate and corresponding private key in separate files locally
- Certificate in a file called `ca.crt`
- Key in `ca.key`
- See here for info on generating these files
- Access to your install's `kubectl` command
- A deployed project to use the cert-manager with
- Cert Manager plugin installed

METHOD

ploy CA to the cluster

The following command will deploy your certificates located in your current folder to the cluster under the default cert-manager namespace

```
kubectl create secret tls ca-key-pair \
  --cert=ca.crt \
  --key=ca.key \
  -n cert-manager
```

tup the issuer

We now need to create an issuer pointing at the previously deployed certificate pair. The following command will create a non-namespace scoped deployment, to restrict the namespace simply include which namespace in the metadata e.g. `namespace: test-deployment`

```
kubectl apply -f - <<'EOF'
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: my-ca-issuer
spec:
  ca:
    secretName: ca-key-pair
EOF
```

rify

Once the above is deployed we can verify it's working by running the following commands

Firstly ensure the issuer `my-ca-issuer` is ready to provide certificates, the following command should report back `READY: True`

```
kubectl get clusterissuer my-ca-issuer
```

Next we can manually issue a certificate by deploying a `Certificate` and referencing the issuer. Replace the values identified with values pertinent to your deployment of Orion and CA certificate

```
kubectl apply -f - <<'EOF'
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: my-service-cert
  namespace: <YOUR NAMESPACE>
spec:
  secretName: my-service-tls
  issuerRef:
    name: my-ca-issuer
    kind: ClusterIssuer
  commonName: my-service.<YOUR DOMAIN>
  dnsNames:
  - my-service.<YOUR DOMAIN>
EOF
```

pply to a project

We have our CA setup with the issuer so now we can apply it to our deployed project and have our CA provide a TLS certificate for the hubble ingress.

Edit the project's ingress controller using kubectl

```
kubectl edit ingress hubble-ingress -n <YOUR PROJECT NAMESPACE>
```

Under annotations remove the `cert-manager.io/issuer: letsencrypt-prod` line and replace it with

```
cert-manager.io/cluster-issuer: my-ca-issuer
```

Then append the following to the `spec` section

```
  tls:
    - hosts:
        - <PROJECT NAME>.<YOUR DOMAIN>
      secretName: hubble-tls-certs
```

Save and exit.

Allow some time for the secret to be created and try to access your applications web interface, you should see the certificate provided is now one provided by cert manager

```
$ kubectl get secrets -n <YOU PROJECT DOMAIN>
NAME               TYPE                DATA   AGE
hubble-tls-certs   kubernetes.io/tls   3      1m
```

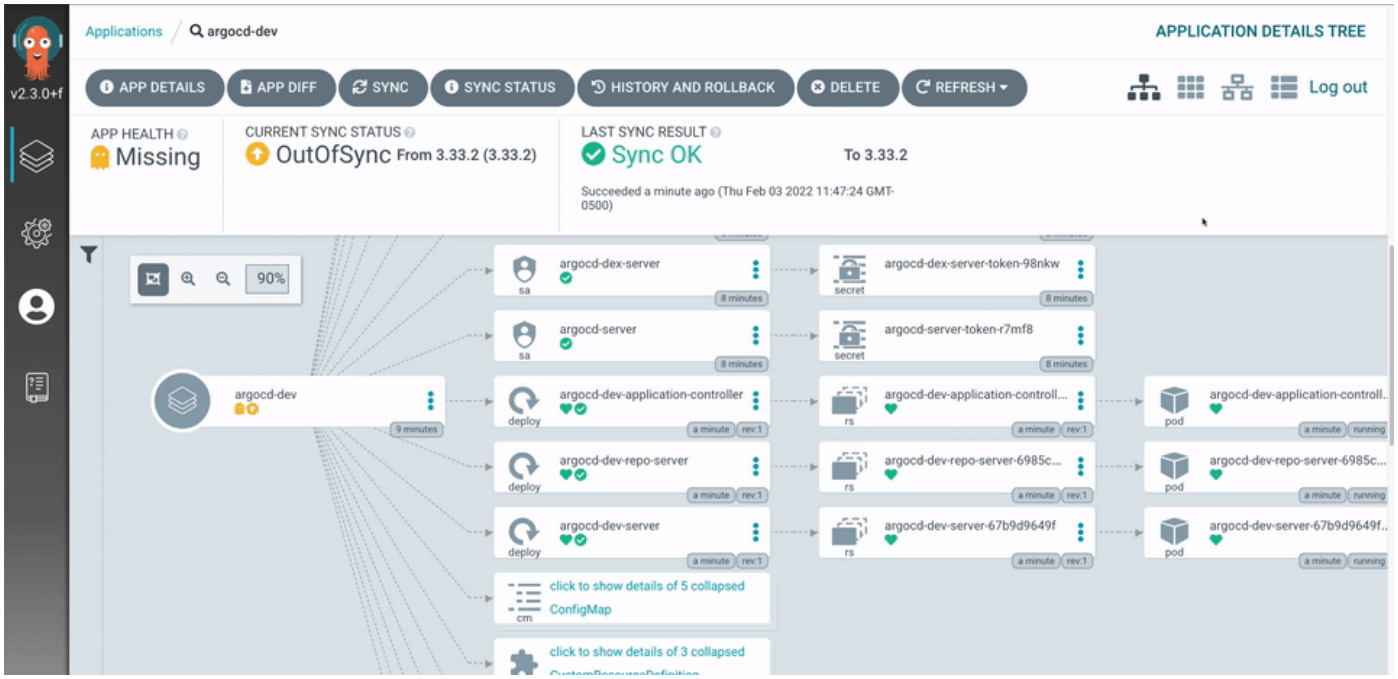
This change to the project ingress will need to be made to all active and future projects deployed.

3.5 Argo CD Overview

3.5.1 Introduction

Argo CD is a declarative, GitOps continuous delivery tool for Kubernetes. It is with this tool admins can see all the deployed Kubernetes clusters and services. This is useful for updating, refreshing, syncing, and reviewing/debugging each service's logs.

If you are new to Argo CD we recommend reviewing the official documentation.



Operation with ArgoCD

We highly recommend following the operators guide.

Validating your Argo deployment

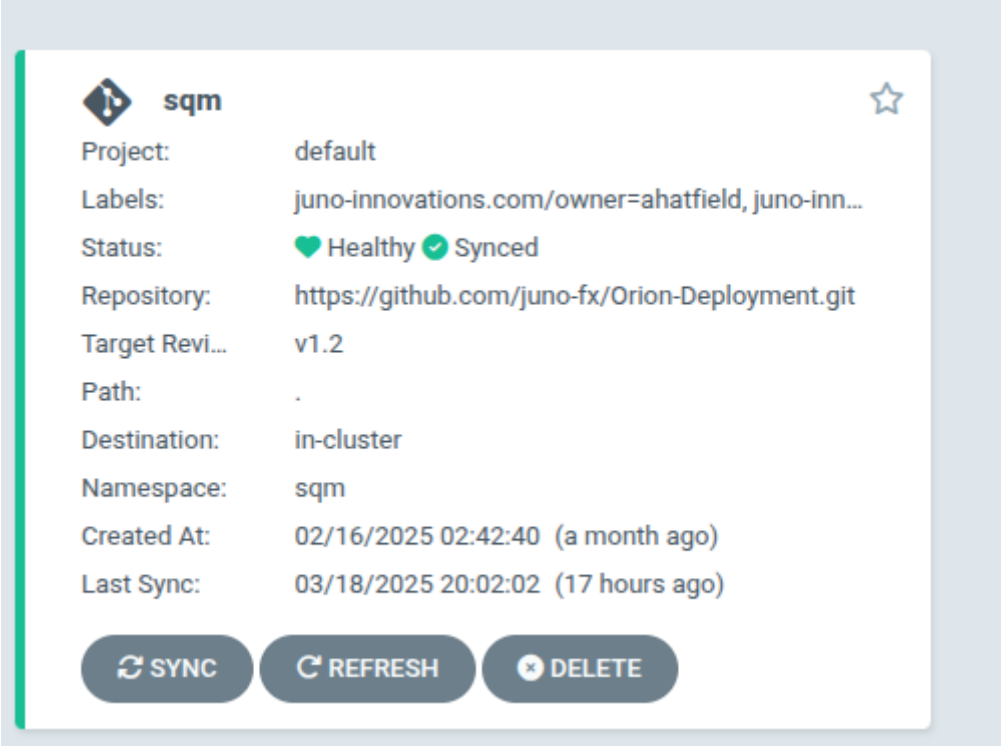
In case you need to validate your ArgoCD deployment is healthy, you cannot rely on the web UI itself - Argo does not manage its own resources.

To do that, you can use `kubectl logs -n argocd <pod name>` to investigate the state of each individual component. A good sanity check for your ArgoCD deployment is validating its pods are up, using `kubectl get pods -n argocd`

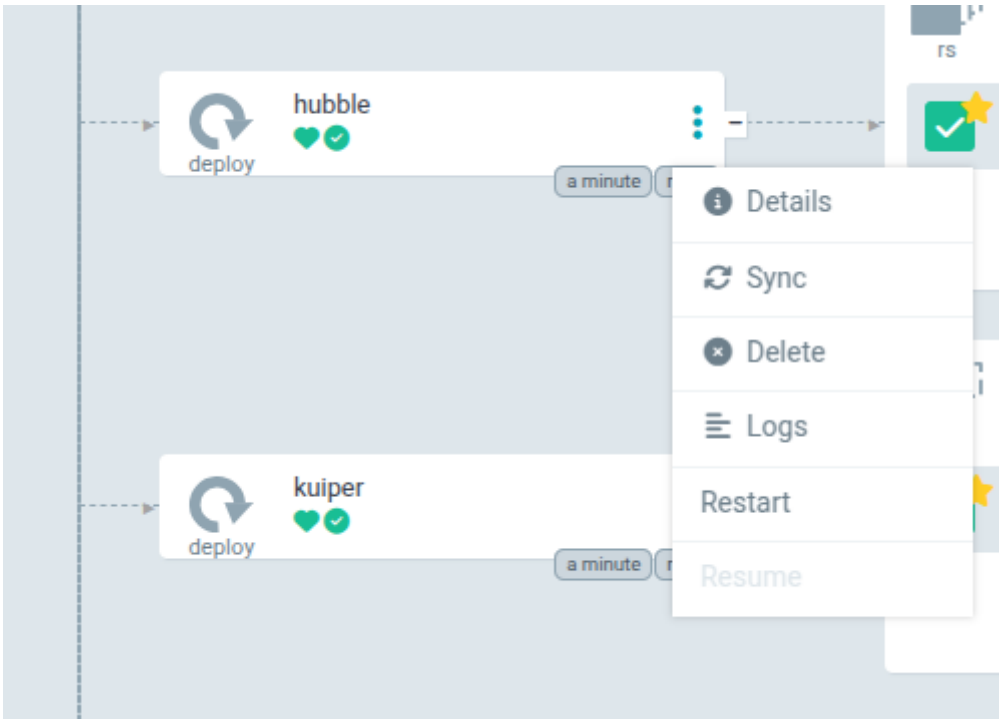
3.5.2 Managing Deployments

Admins can quickly and easily manage their Orion deployments for each project as needed. Whether that's restarting a deployment to ensure it picks up the latest version updates, checking logs for potential errors, or syncing. It can all be easily accessed via Argo CD and the projects application.

Click the project application which has the deployment you are looking to manage.



Find the deployment you wish to manage. Selecting the 3 dots you will see the available options. While clicking the deployment directly will provide you the full summary view.



Options

1. Logs: review/debug any potential errors for the selected deployment via the logs.
2. Restart: restart the deployment. This will force the parent project to pick up the latest updates for this deployment.
3. Details: see the full manifest/summary for this deployment.

3.6 Monitoring and Logs

3.6.1 Logging

Each workload you deploy to your cluster, whether through Juno products or on your own will eventually run in a container.

For workloads within Juno, we provide you with the ability to view workload logs directly in the web console, as well as related Kubernetes events. Additionally, services that make up Juno can all be viewed in ArgoCD and their logs inspected within it. To get a quick overview, check out this guide

For workloads you run outside of Juno, you can use Kubernetes to inspect the logs.

A container is simply a process that gets confined & contained by the runtime. All of the logs it outputs are gathered and available via the Kubernetes api, using the `kubectl logs` command. To inspect a workload not managed by Argo or Juno, you would look up the name of the pod and run `kubectl logs -n <namespace> <pod name>`

This is useful when troubleshooting ArgoCD itself.

3.6.2 Monitoring

To better operate your cluster in production, you can leverage a monitoring stack. It will be able to give you insight into performance of your cluster hardware, as well as deployed workloads.

We are working on integrating a full monitoring stack into Juno's Terra store, to provide you an opinionated, out-of-the-box install.

While that's on the way, the Kubernetes ecosystem provides plenty of options - though they do require some setup and each have a learning curve. For workloads outside of Juno, we recommend the kube-prometheus-stack as a good starting point.

3.7 Orion - Upgrade and Rollback

3.7.1 Overview

This guide covers upgrading and rolling back the version of your deployment of the Orion platform.

In general, there are 2 components you might want to upgrade: - Genesis - the management layer of your cluster - Environments / projects

Both Genesis and individual projects also consist of individual services.

3.7.2 Assumptions

This guide assumes that:

- you already have basic familiarity with the product and its core concepts.
- you are familiar with core ArgoCD concepts. If you are new to this tool, you can find a quick reference on it in our ArgoCD Introduction.
- you can perform simple kubectl operations. If you are new to kubectl, you can find a good starting point here

Moreover, if you run an airgapped deployment it assumes that:

- you have a proxied or forked version of all OCI images available
- you have up-to-date deployment charts available

3.7.3 Versioning strategy - Genesis management layer and projects

For Genesis, Orion's management layer, you can safely assume upgrades of minor and patch versions will not break existing projects. The minimum supported version of the Genesis management layer is listed in the technical changelog.

When an upgrade is required, you would always upgrade first the Genesis management layer and only then the projects.

For the management layer, you can freely upgrade it through minor and patch versions without disrupting running projects. The aim here is to avoid downtime to heavily-used environments and projects, letting you upgrade at your own pace.

While projects can stay on a previous version until it is convenient to upgrade, we recommend keeping your management layer up-to-date to make use of the latest enhancements.

3.7.4 Changelogs

We maintain two release changelogs:

- One contains new features where you can review the latest & greatest Orion has to offer.
- The technical changelog provides guidance for technical teams operating Orion deployments. This is intentionally kept brief and only informs you of any deprecations, migration steps between major versions or addressed security vulnerabilities. If a release has nothing extra to consider, it is explicitly listed as such for you to have confirmation.

We recommend you link the technical changelog within your internal documentation and review it prior to each upgrade. Any major release requiring special upgrade steps also contains a detailed migration guide linked under the changelog.

3.7.5 Project upgrade workflow

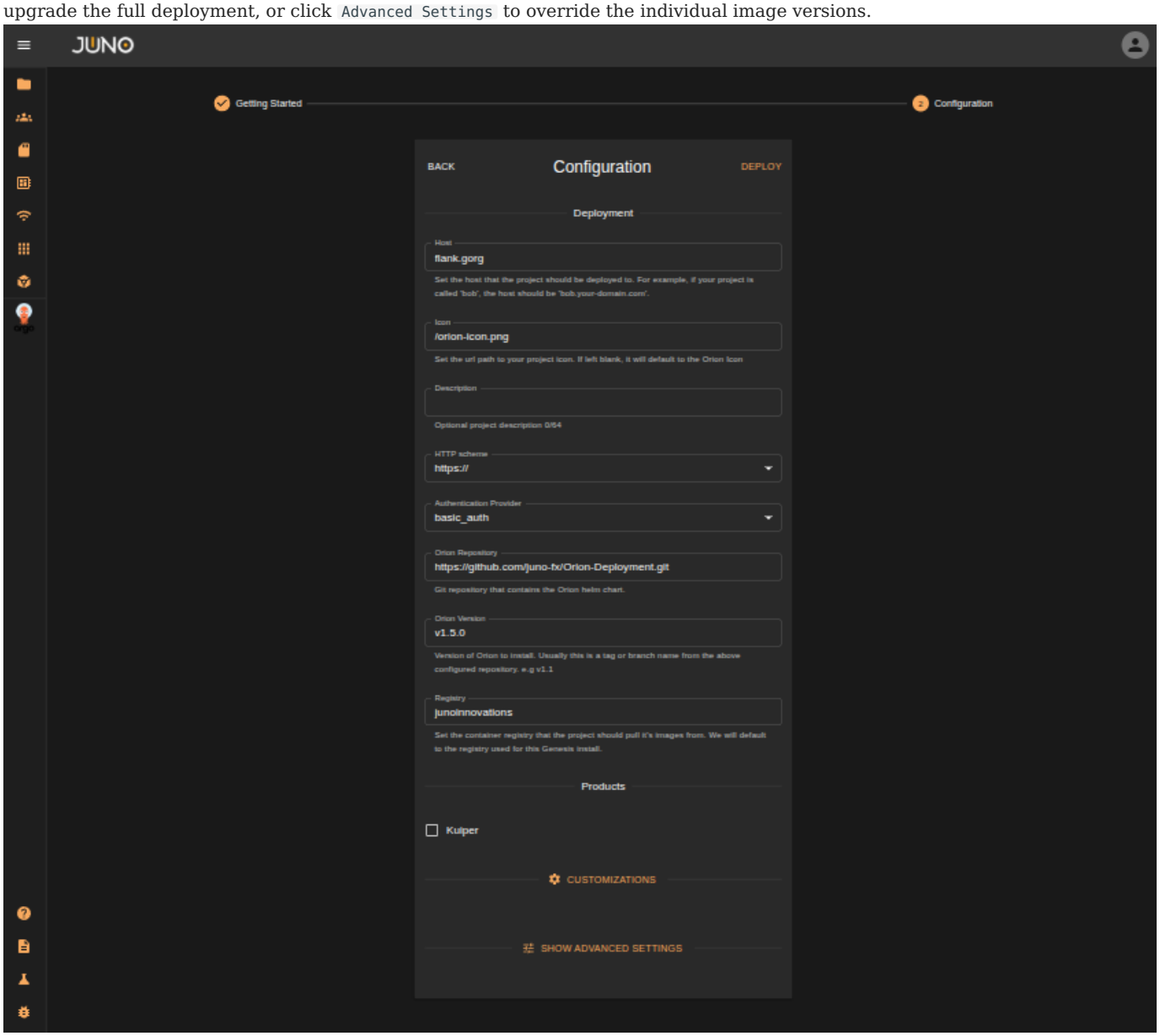
Project upgrades can be carried out either via the Genesis UI or the kubectl commandline. For debugging, having both at hand is useful.

To debug a project's upgrade status, go to ArgoCD either via the dashboard in Genesis, your existing ingress or by port-forwarding it.

3.7.6 Projects - adjusting individual image versions

In day-to-day operations, it is recommended you stick to predefined image versions in the deployment chart. When debugging or inquiring about early access/unstable releases, you might need to upgrade or roll back an individual component.

For projects, this is done by clicking "Manage" -> "Configure". Simply pass in the chart version to the `Orion Version` field to upgrade the full deployment, or click `Advanced Settings` to override the individual image versions.



Alternatively, you can update the Orion version deployment via kubectl:

```
kubectl patch app -n argocd <project name> --type='json' -p='[{"op": "replace", "path": "/spec/sources/0/targetRevision", "value": "v1.4.0"}]'
```

The release version number corresponds to a target branch of the environment/project deployment repo. Available releases can be seen both in the changelogs and as branches on the project deployment repo found here.

3.7.7 Genesis management layer upgrade workflow

To upgrade Orion's management layer, Genesis, you would edit the target branch.

The correct image versions will be automatically defaulted to, assuming you have no overrides in your Genesis settings accessible by clicking on your profile icon. If you do have overrides, make sure to remove them before the upgrade!

To edit the target revision, you can either utilize the Genesis UI or kubectl. To update via the Genesis UI, simply navigate to the admin settings page. Click the deployment settings drop down and pass in the deployment charts branch version into the `targetRevision` field. Available releases can be seen both in the changelogs and as branches on the deployment repo found here.

To perform the same with kubectl:

```
kubectl patch app -n argocd <project name> --type='json' -p='[{"op": "replace", "path": "/spec/sources/0/targetRevision", "value": "v1.4.0"}]'
```

From there, you can observe the upgrade status in ArgoCD's UI.

Genesis management layer upgrade workflow - individual components

Similarly to projects, it's recommended to not adjust the management layer image versions unless requested so by the support team.

For the management layer, you can adjust:

- Genesis (Orion's central management component)
- Titan (user management)
- Terra (plugin manager)

You can adjust those either directly in the web UI by once again navigating to the admin settings page. Click the service settings dropdown, and pass in the image versions you wish to override. Please note if you have overridden these versions and wish to roll back to the chart defaults. You can simply clear out the values and click submit. This will automatically roll the versions back to the original chart values.

```
$ kubectl edit -n argocd genesis
```

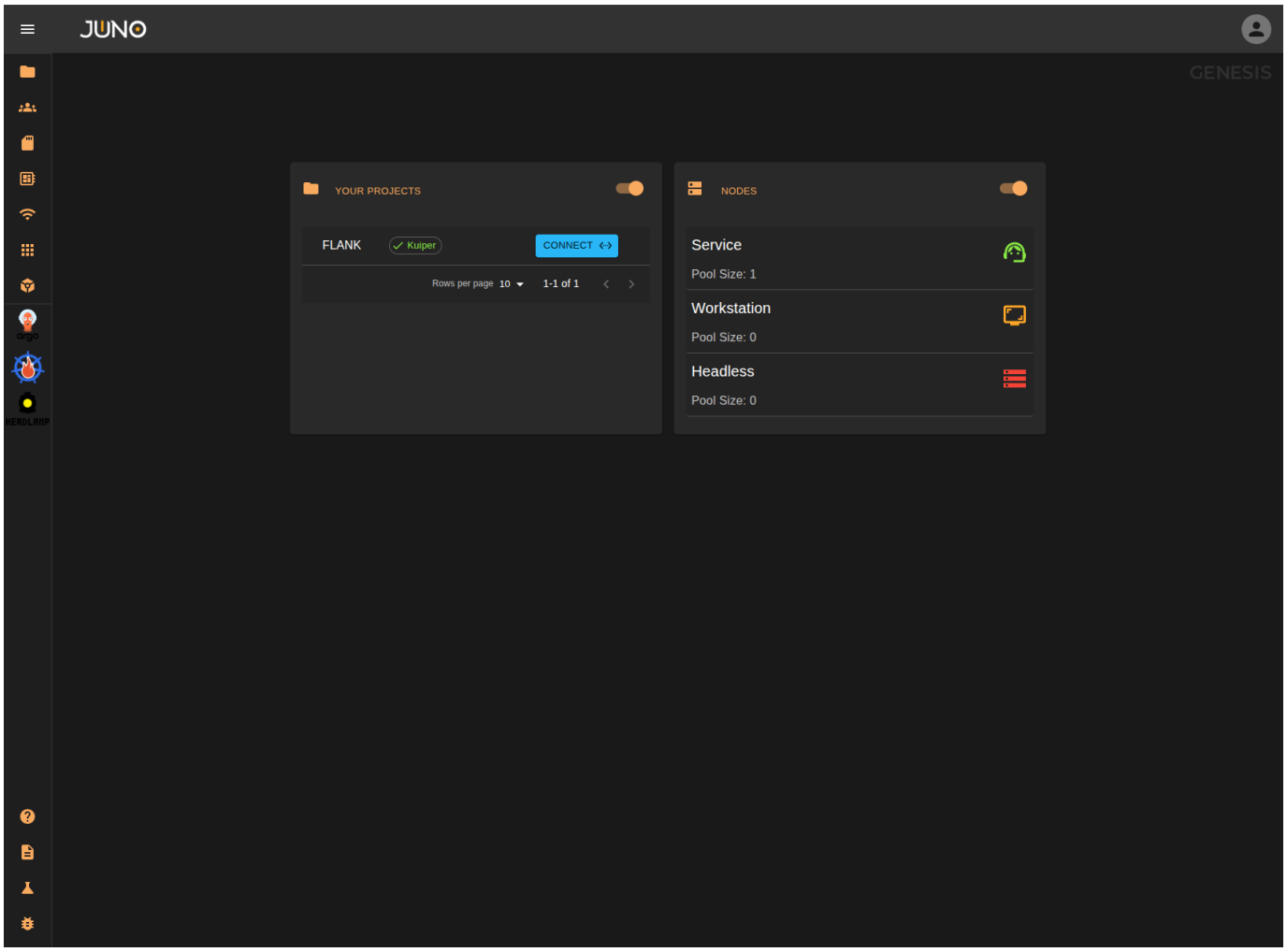
```
      parameters:
        - name: image.tag
          value: X.Y.Z
        - name: titan.image.tag
          value: X.Y.Z
        - name: terra.image.tag
          value: X.Y.Z
```

4. Products

4.1 Genesis

4.1.1 Introduction

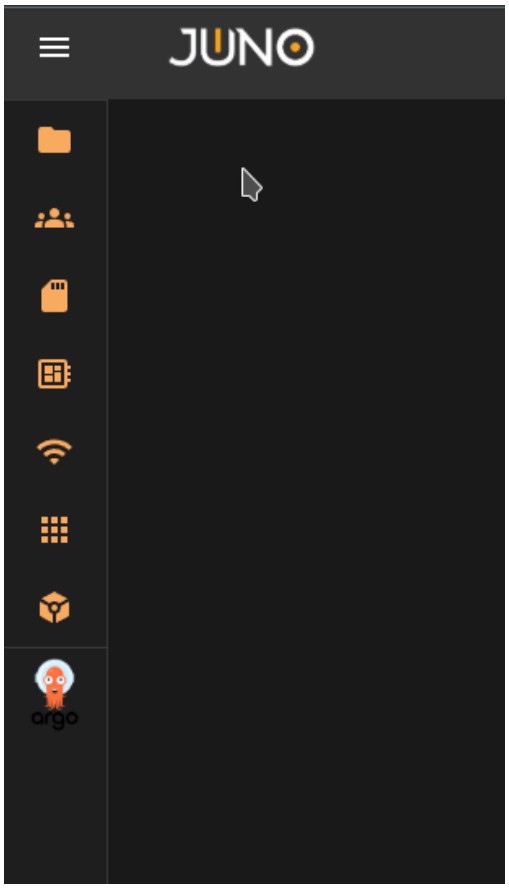
Genesis is the official dashboard for managing your entire Orion ecosystem. From here, users can create, manage, and connect to projects. It provides tools for user and group management across projects, services, and roles. Storage, network capabilities, and custom workloads can all be configured on a per-project basis. Additionally, apps can be installed via Terra, and the Orion license can be managed directly from the dashboard.



Navigation

MAIN LINKS/INSTALLED CLUSTER APPS

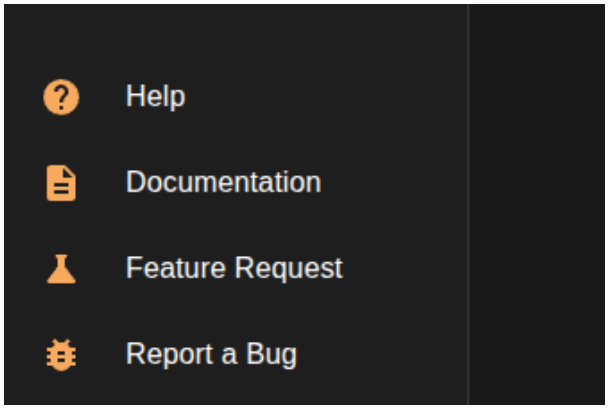
On the left side of the screen, users will find a sidebar. To expand it, click the icon in the top-left corner of the page. To collapse the sidebar, click the arrow at the top. Here you will find the main links, Cluster Installed apps (If you have any), and help links which are located at the bottom of the side bar



HELP LINKS

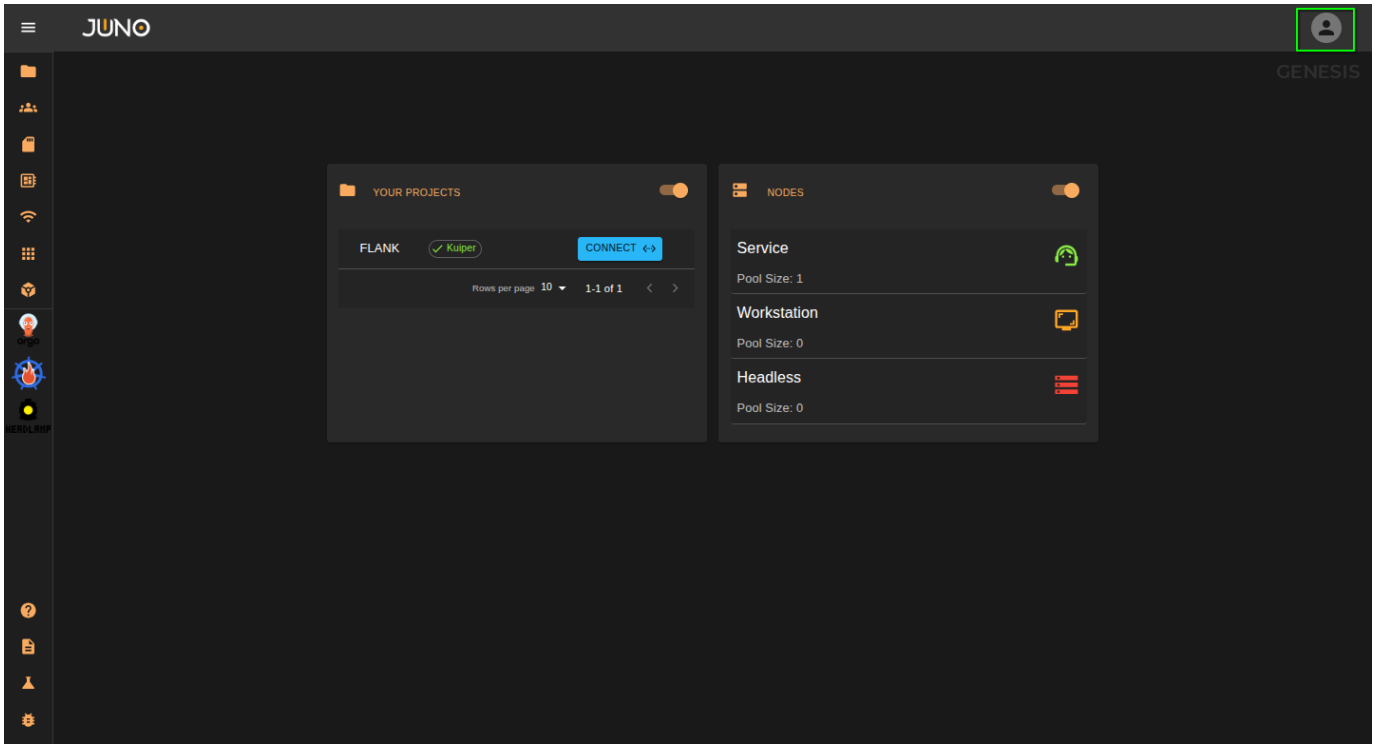
These links are located at the bottom of the side bar.

Link Name	Definition
Help	Starts a help stepper that will guide you on how to use the main functionalities of the genesis UI.
Documentation	Links you to our Official Orion Documentation.
Feature Request	Links you to our Github report repo where you can open a ticket for a new feature.
Report a Bug	Links you to our GitHub report repo where you can open a ticket for an issue you encounter.



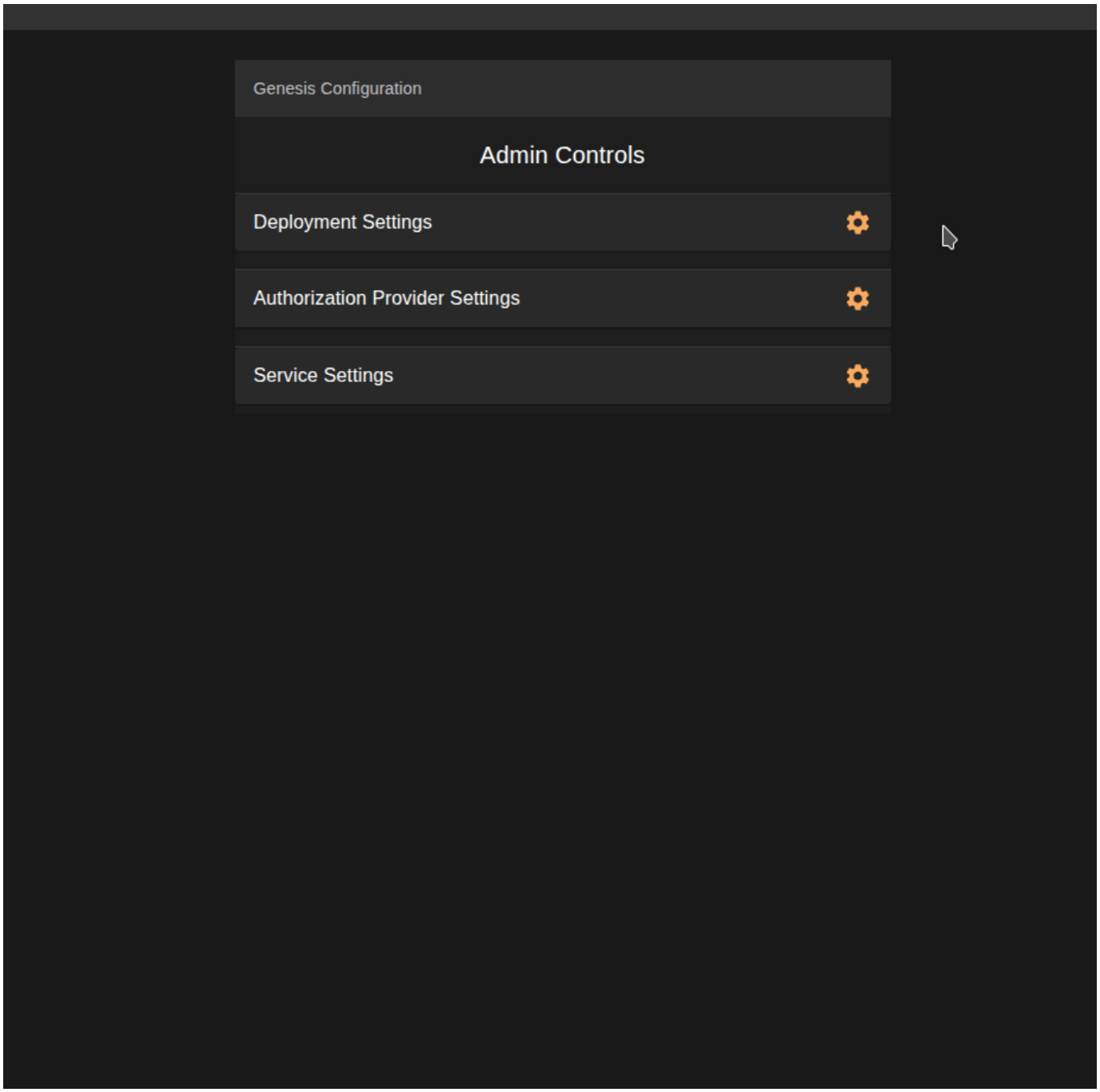
User Menu

Click the user icon located in the top right of genesis to open the user menu. Here you can access the settings page, create an api token, development page and log out.



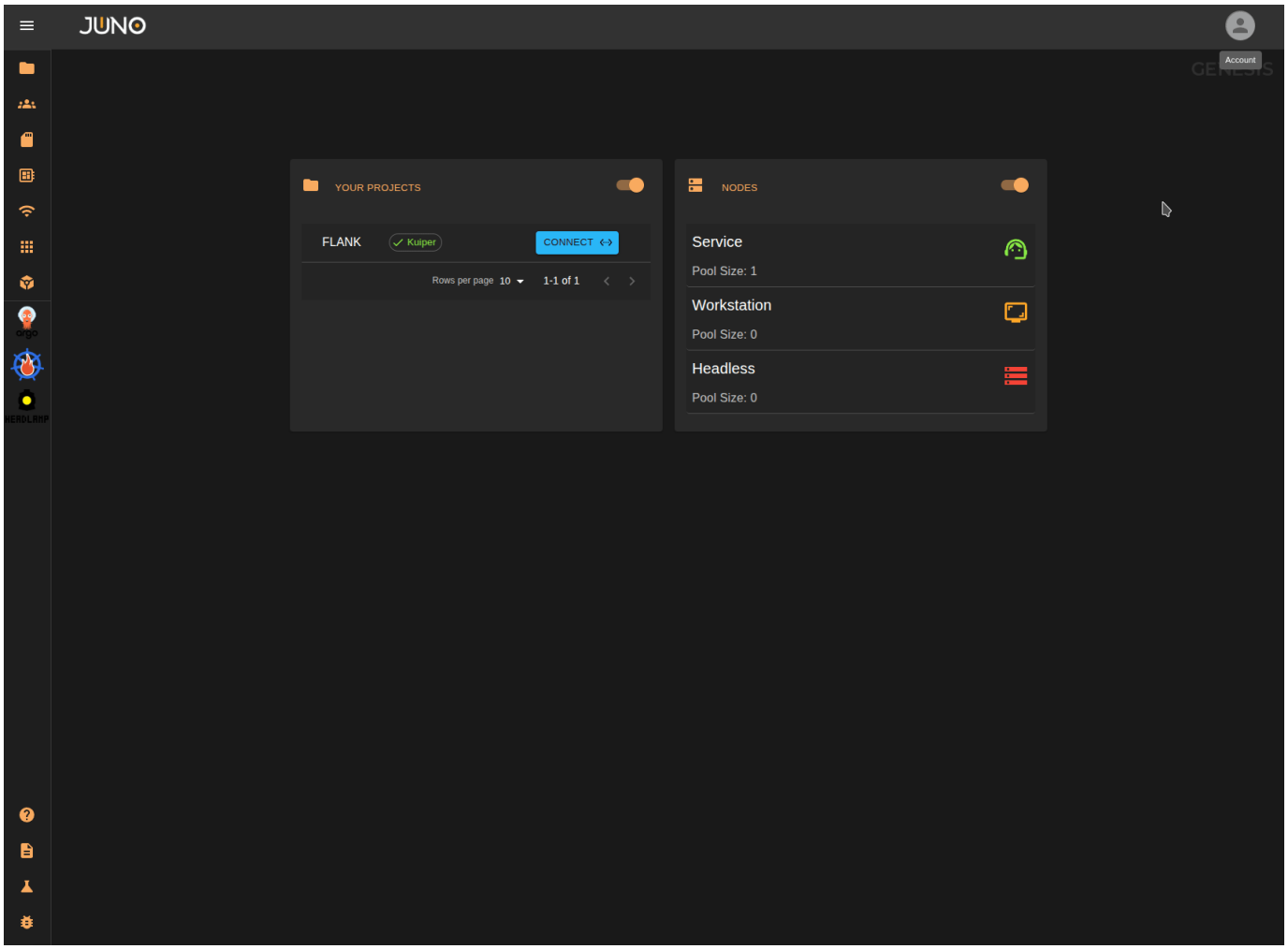
SETTINGS PAGE

Here you can update your deployment targetRevision, authorization providers, and service images. You can also restart services.



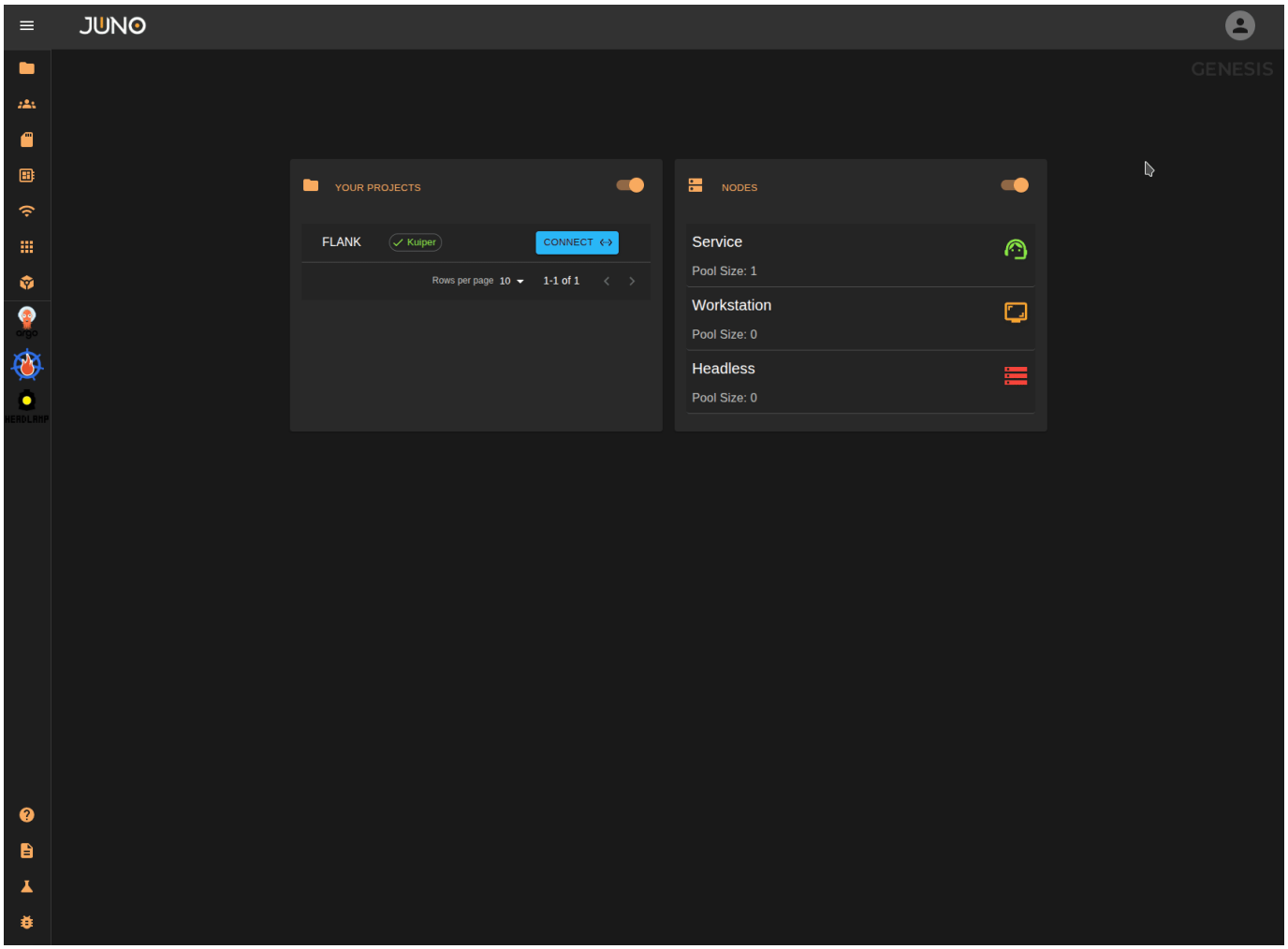
CREATE API TOKEN

If you want to hit our service API's you will need to generate an API token. Open the user menu and click Create API Token.



DEVELOPMENT

We use FastAPI under the hood for our services. You can navigate to our development page to directly use the FastAPI Swagger UI to hit our services. Each service is available specifically to admin users or users assigned to the respective group. This can be used for testing or development purposes.

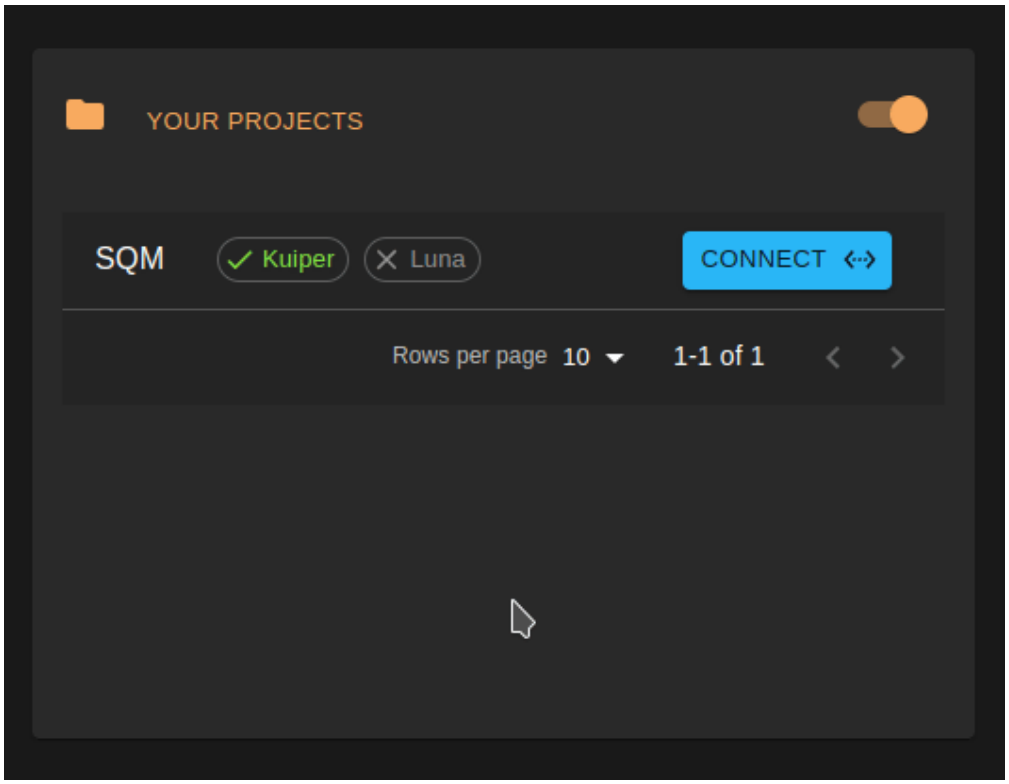


Dashboard widgets

The homepage of genesis will have two widgets displayed. These are used for quick access/viewing.

YOUR PROJECTS

This widget displays your active projects. For a project to appear in this widget, your user must be assigned to the project group, and the project must be active. Learn more about user group assignment here.

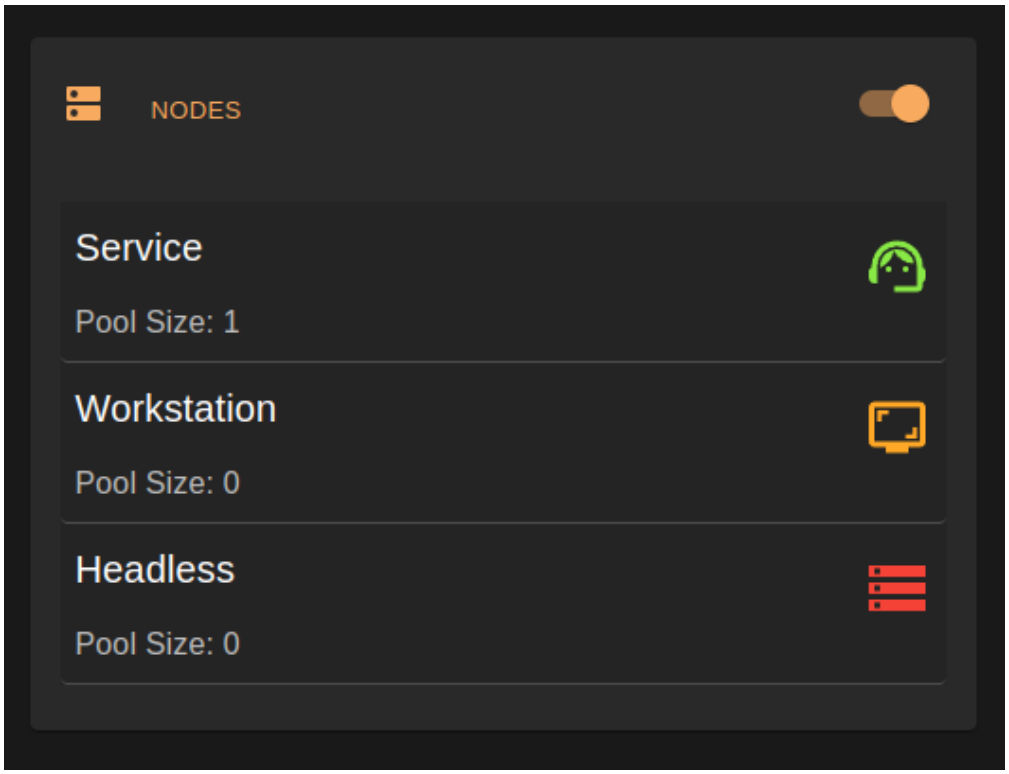


NODES

This widget displays the following types of nodes

Node type	Description
Service nodes	Represent projects.
Workstation nodes	Show all active running workloads.
Headless nodes	Represent workloads running without a user interface (e.g. render node).

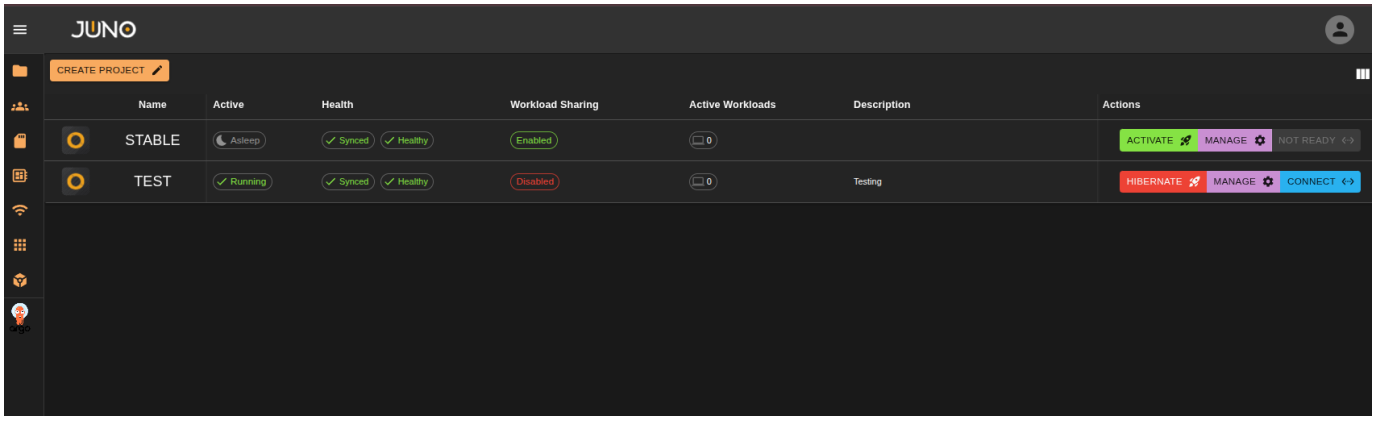
Learn more about the nodes and network page here.



4.1.2 Project Management

> **What is a project?**
>
> A project is an isolated environment for Hubble and Kuiper with its own network policies; connecting to a project lets you launch workloads.

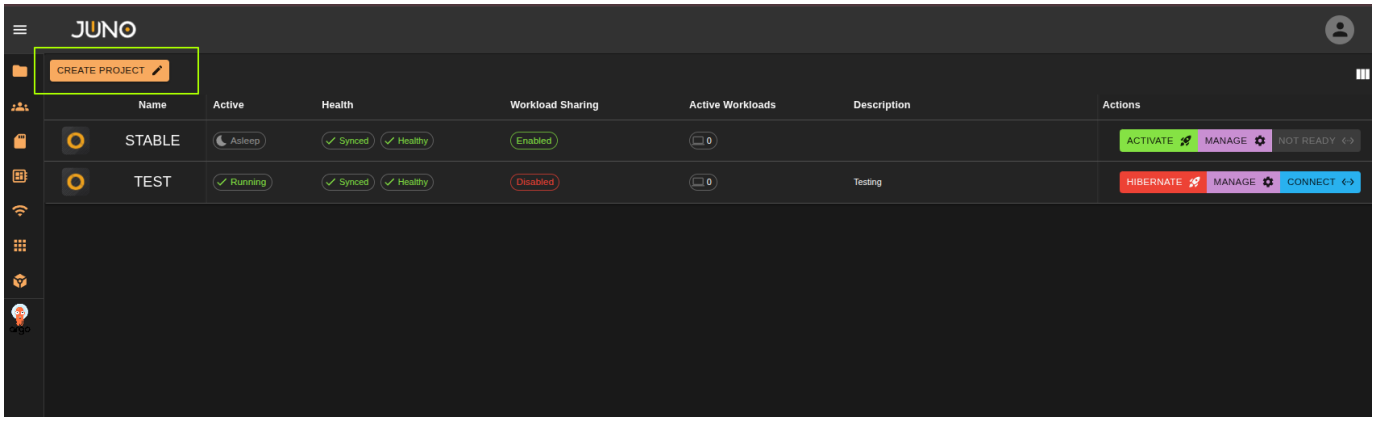
The project management page allows admins to create, configure, and activate projects. While also allowing users to connect to their assigned projects. See user group assignments on our User Management Page for more information. Users can also see which projects are active, the status of each project, which products are active on each project, the number of active workloads currently running on a project, and the description of the project.



Project Creation

At the top left of the project table, there is a button labeled "CREATE PROJECT" clicking this button opens a project creation wizard.

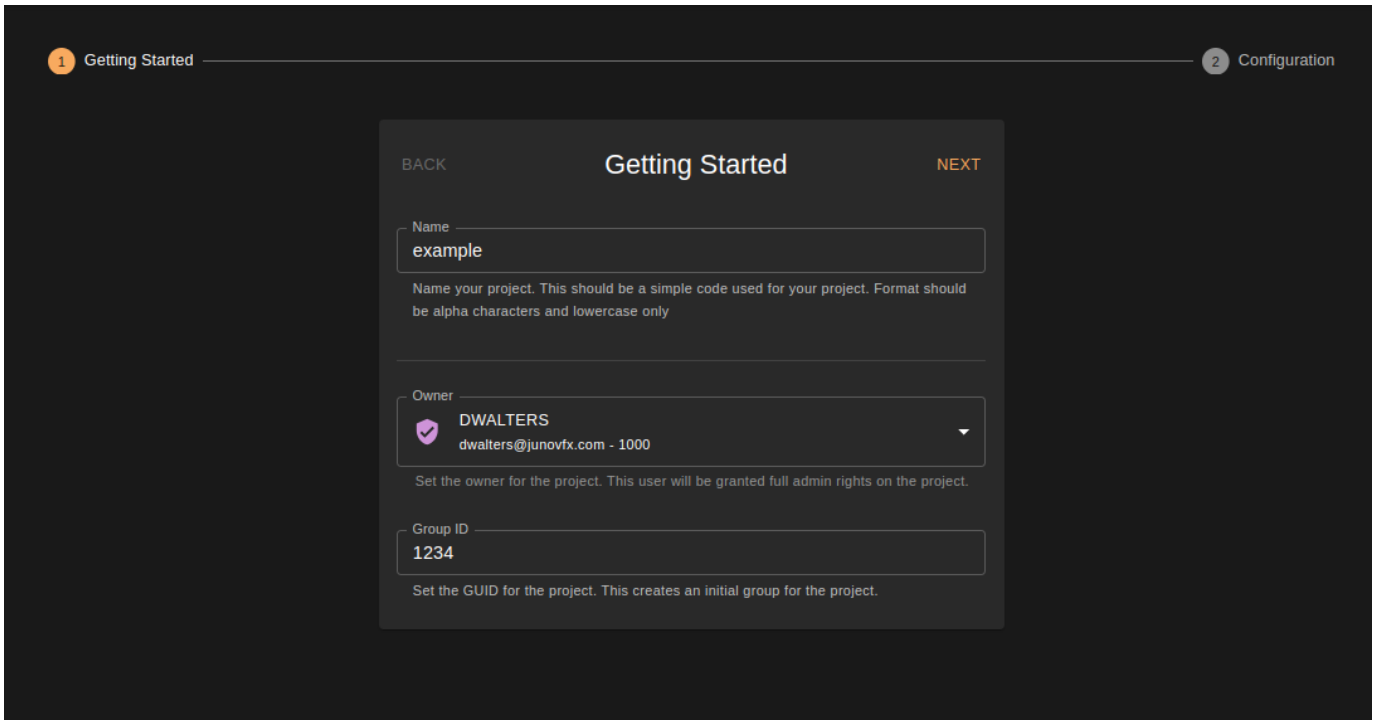
> **Note**
>
> Admins are the only users that are able to create new projects. Learn more about user management here



GETTING STARTED

The first three fields in our getting started section must all be filled out before proceeding to the configuration section.

- Project names must be alpha and lowercase characters only.
- Only admin users can be selected as an owner of the project
- A GUID must be given. This will be used when creating the project group for user assignment.
- Once all fields are filled in click the NEXT button to continue to the configuration section.



CONFIGURATION

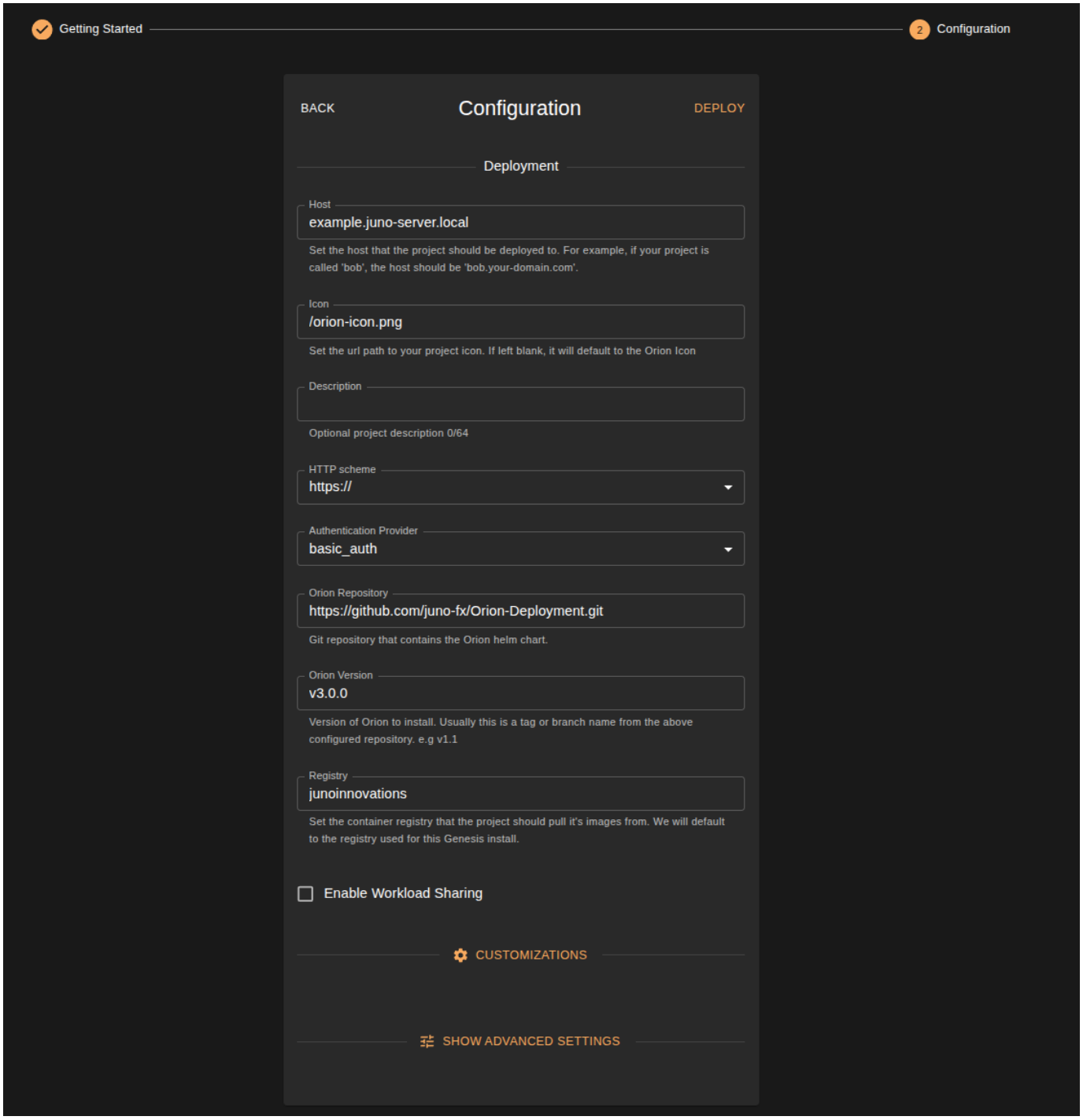
The configuration section, is where you can set your projects configuration. Once completed, simply click DEPLOY to finish the project creation wizard.

REQUIRED FIELDS

- The host url
- HTTP scheme
- Authentication Provider
- The repository for the orion deployment helm charts
- The Orion version to install
- The registry where the project will pull the images from
- Which products you wish to activate for the project.

OPTIONAL FIELDS

- Project Icon
- Project Description
- Enable Workload Sharing. Checking this box, will allow users to share their workloads with one another for this project environment.
- You can add additional customizations to the project. These will be additional helm charts to install alongside the main Orion deployment charts. Example: NVIDIA GPU Operator
- You also have access to advanced settings for the project. Where you can choose specific product versions to install. Doing this is not recommended as each Orion deployment has these versions pinned to ensure compatibility

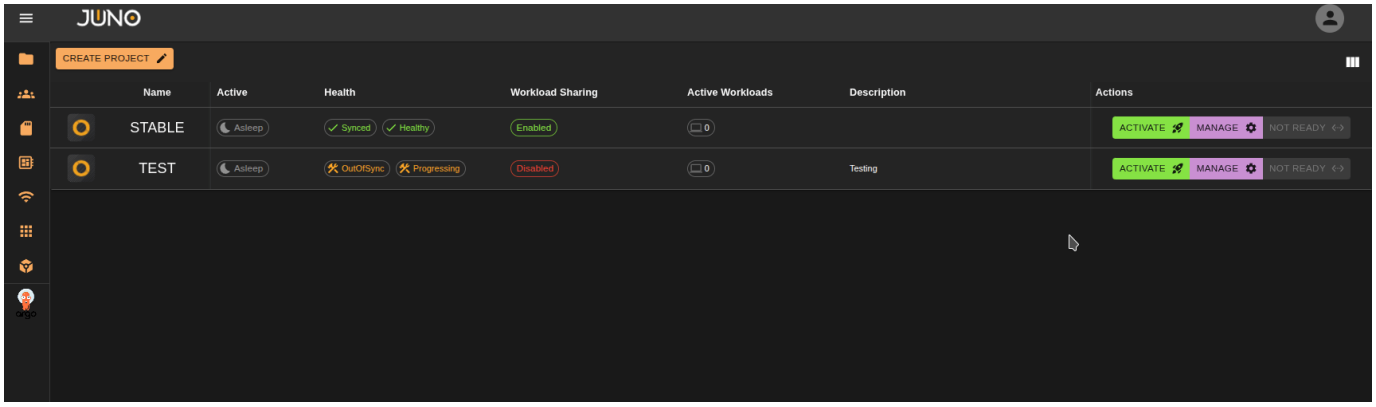


Project Actions

> **Non-admin vs admin privileges**
>
> Admins are the only users that are able to activate/hibernate, manage and configure the projects. Non-admins are only able to connect to their assigned project. Learn more about user management here

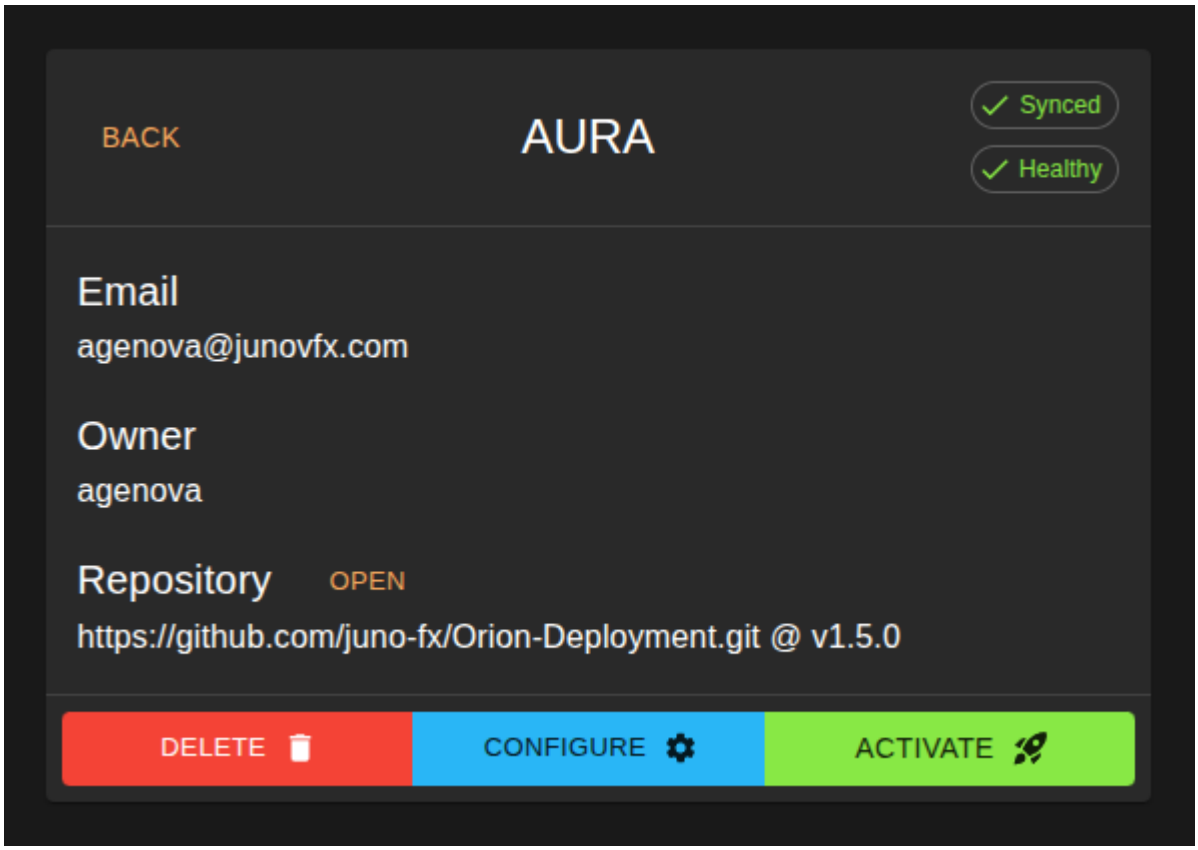
ACTIVATE/HIBERNATE

From the main project table, you will see your available actions for each project. Clicking Activate will begin scaling up the compute resources for the project. Once it's ready users can then connect to the project via the connect button. To hibernate a project and scale the compute resources back down, simply click the hibernate button. You will be prompted with a confirmation pop up before proceeding with the hibernation process. Please note all active workloads on a project must be shutdown prior to hibernating.



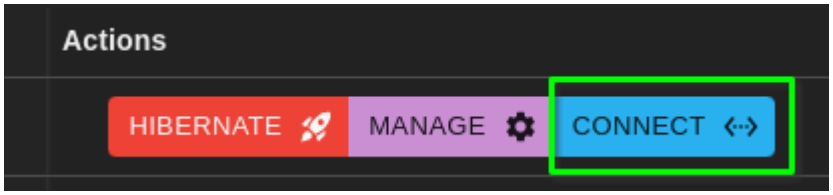
MANAGE

The manage button is only available to admins. Currently, only the owner of the project can manage the project. Owners, can delete the project, go back through the projects configuration to reconfigure the project, and activate/hibernate.



CONNECT

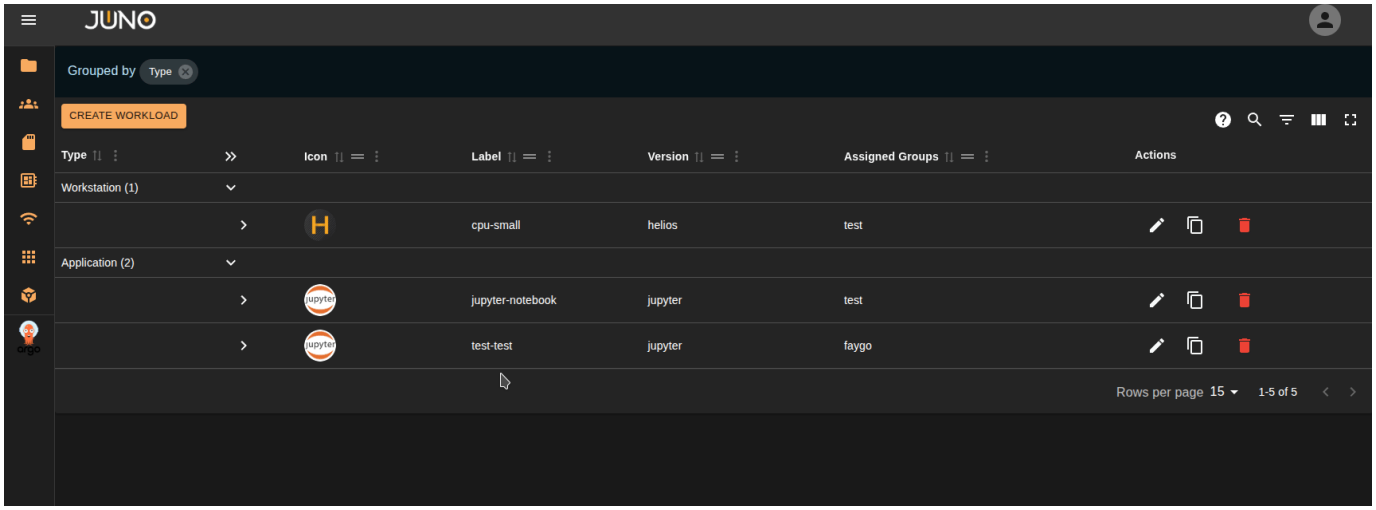
Once a project is active and ready, users that are assigned that project can click the connect button, to connect directly to that project deployment. Read more about hubble here



4.1.3 Workload Management

> **Note**
>
> Workload management is restricted to members of 'admin' and 'kuiper' user groups

The workload management page allows users to create, edit, and delete workload templates, including assigning templates to specific groups and projects. Each template row in the table can be expanded to show additional details for the workload.

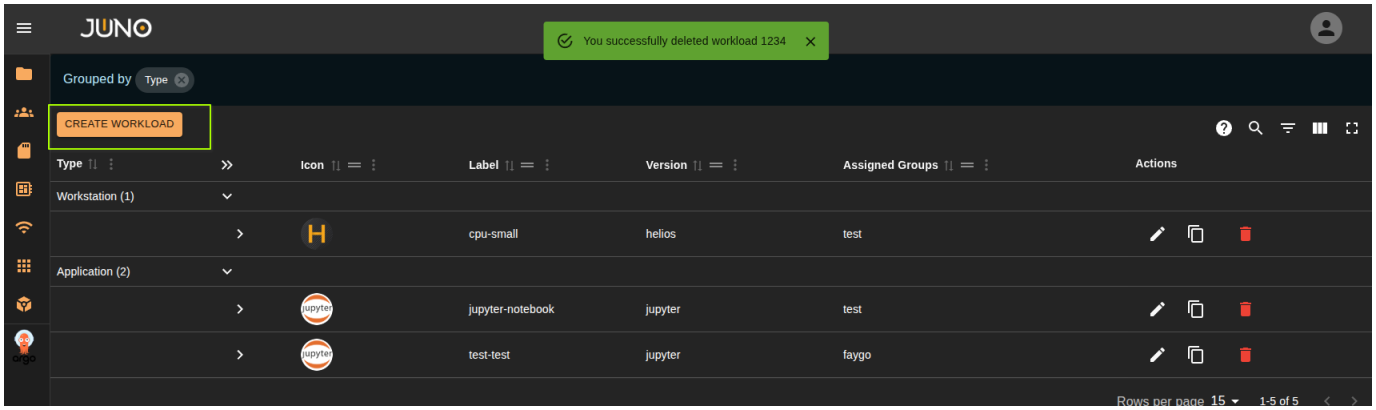


Pre-requisites

Before continuing, ensure you have installed a workload template schema from Terra. You can find the instructions for this in the Terra Plugin Installation documentation.

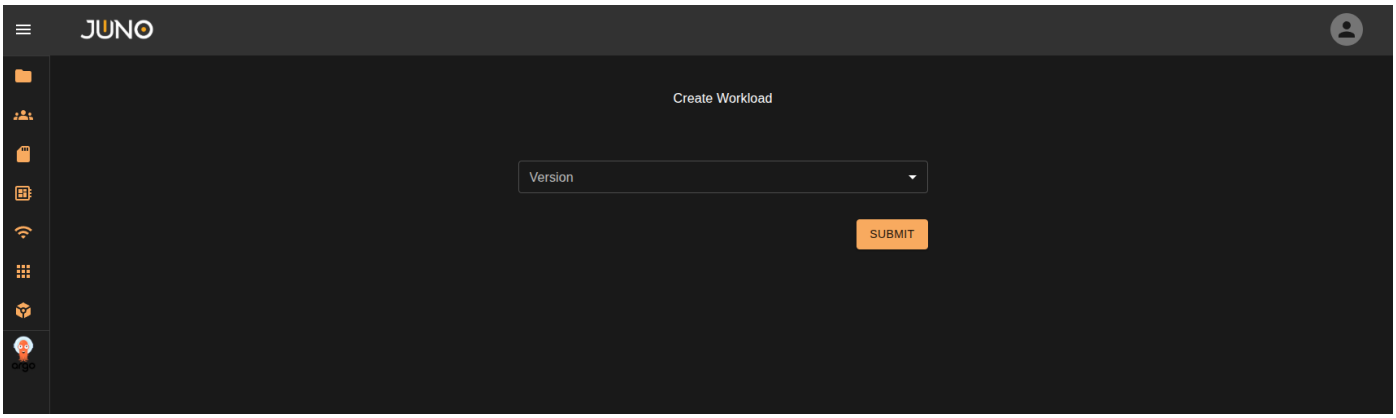
Template Creation

Use the Create button at the top left of the table to open the template creation wizard.



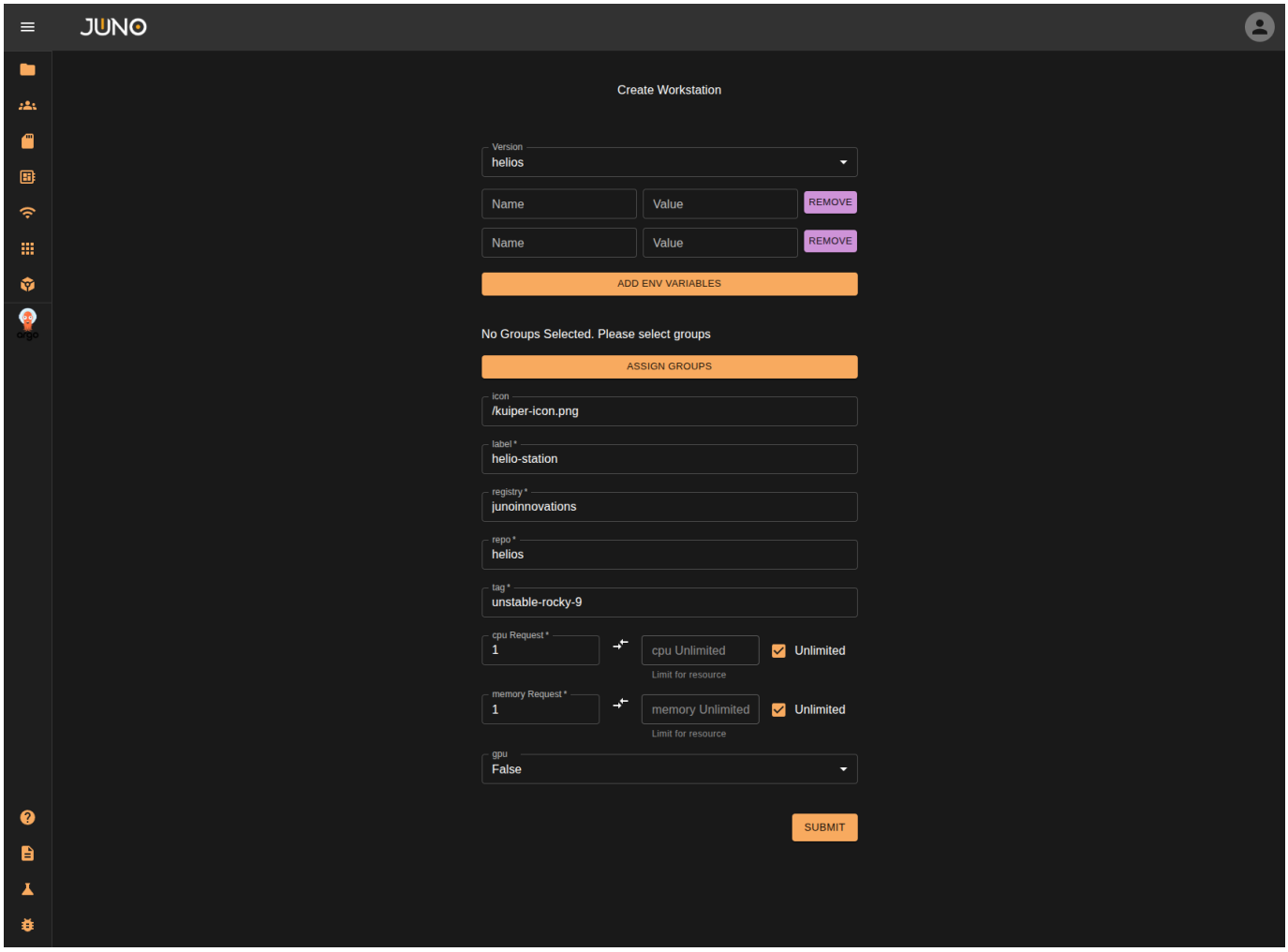
SCHEMA VERSION

The first drop down will list the available schema template versions. These will be grouped by the workload type you are creating the template for. Once selected, the template schema fields will appear for you to fill out.



SCHEMA FIELDS

The schema fields available may differ between the schema version selected. The UI will prompt you for all required fields for the schema. Once the fields are filled out simply click submit at the bottom right, and the workload template will now be created.



SCHEMA FIELD TYPES

When authoring a workload template via Terra the author of the plugin can define additional fields within the configMaps data. These are then populated into the Genesis front end UI.

> **Please note**
>
> Not all Genesis release versions have full field support. Field type and support is continuously added in newer release versions. See Terra's Plugin Fields documentation for the full list of supported field types and properties.

TEMPLATE ENV VARIABLES

Add Environment Variables to a template, which can be used and accessed directly from within a workload.

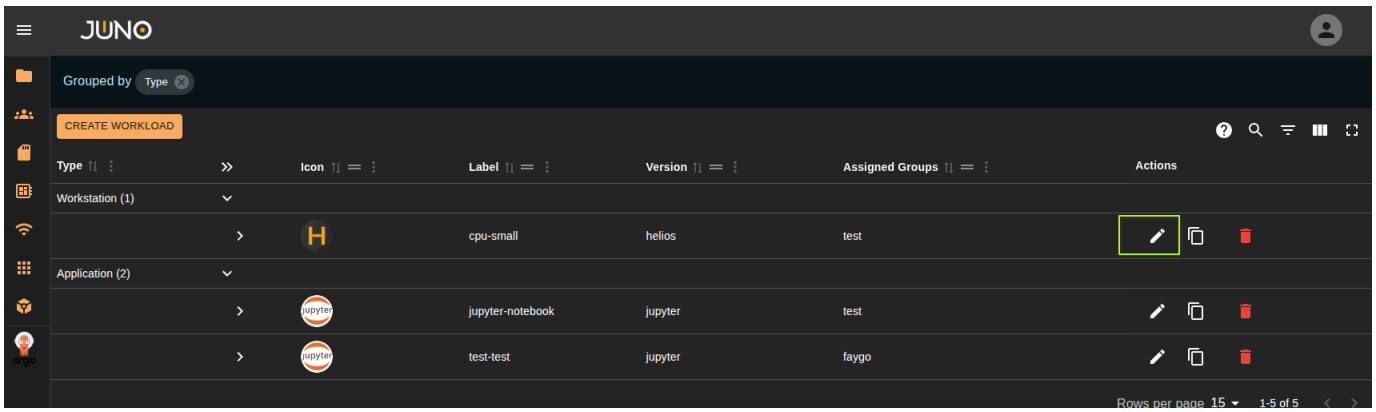
TEMPLATE GROUP/PROJECT ASSIGNMENT

During template creation you must assign the template to at least one group. Assigned templates are available only to the selected group(s); you can scope them to a project or to a specific user group. Learn more about user management here

Template Actions

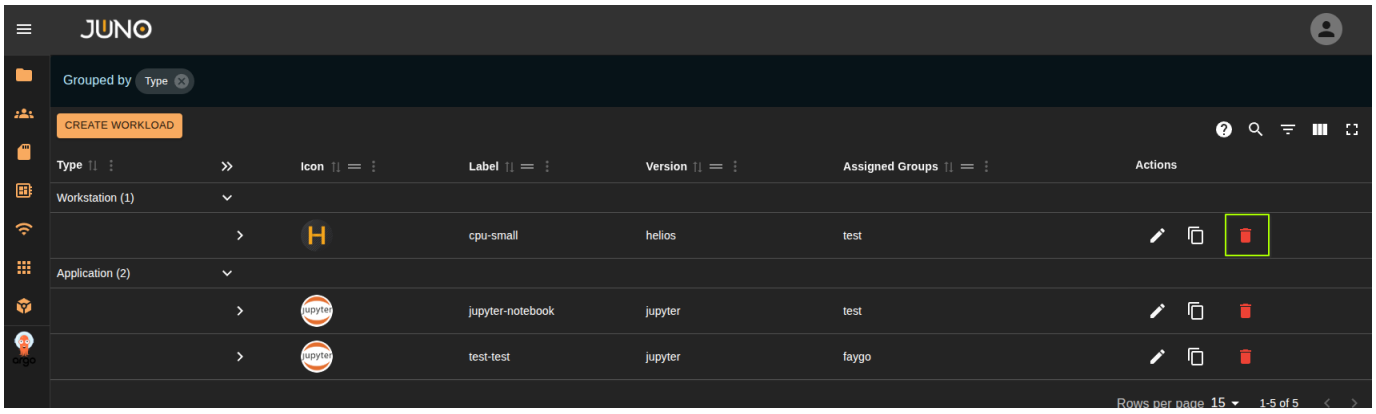
EDIT

From the main table, users can choose to edit a workload template by clicking the pencil icon on the template's row. Doing so will pull up the same creation form, pre-filled in with the current template values. Users can edit any field as needed. The schema version will default to the current template's version, and is not editable. If a new version is needed, users can create a new workload via the **CREATE WORKLOAD** button.



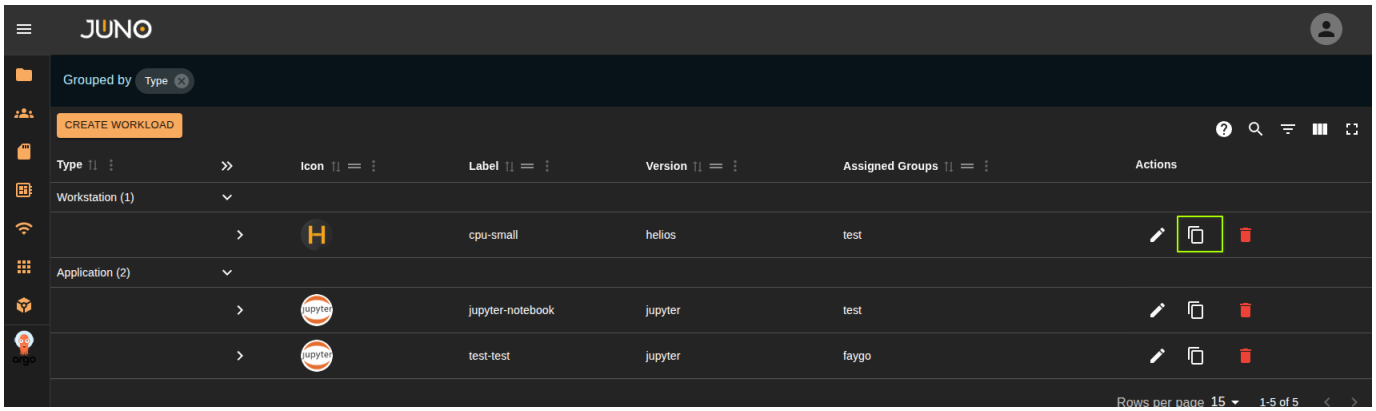
DELETE

Users can also choose to delete a workload template by clicking the red trash can icon on the template's row. The user will be prompted with a confirmation before proceeding.



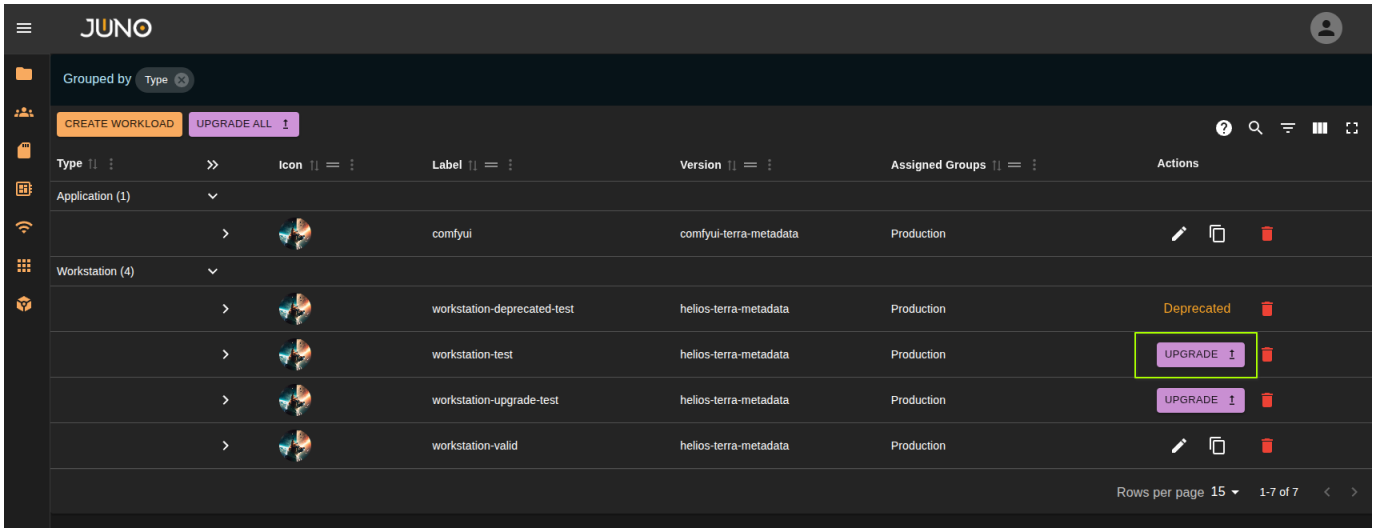
DUPLICATE

Users can duplicated a template by selecting the duplicate icon on each row.



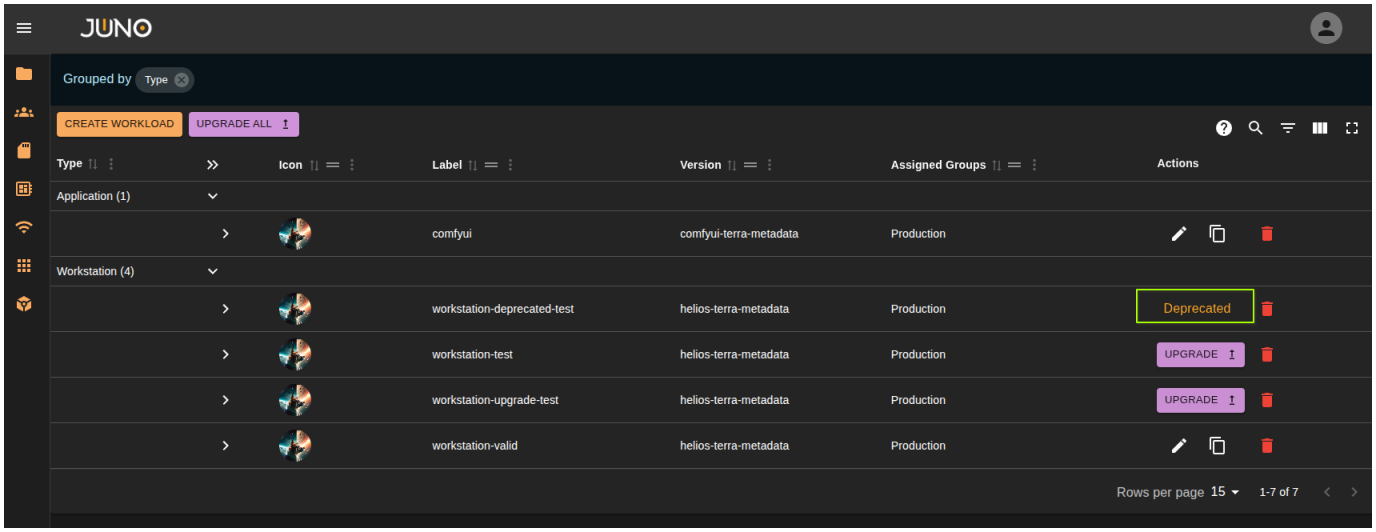
UPGRADE

When a chart for a template changes, you will be able to upgrade the template so it matches what changes the chart has. You can upgrade all templates in the table or upgrade a single template by selecting the upgrade button on the row.



DEPRECATED

A template becomes deprecated when the template matching schema is no longer found or if the template associated schema has changed, and it now has required fields the original template never had. You may continue using the template but you will no longer be able to edit the template.



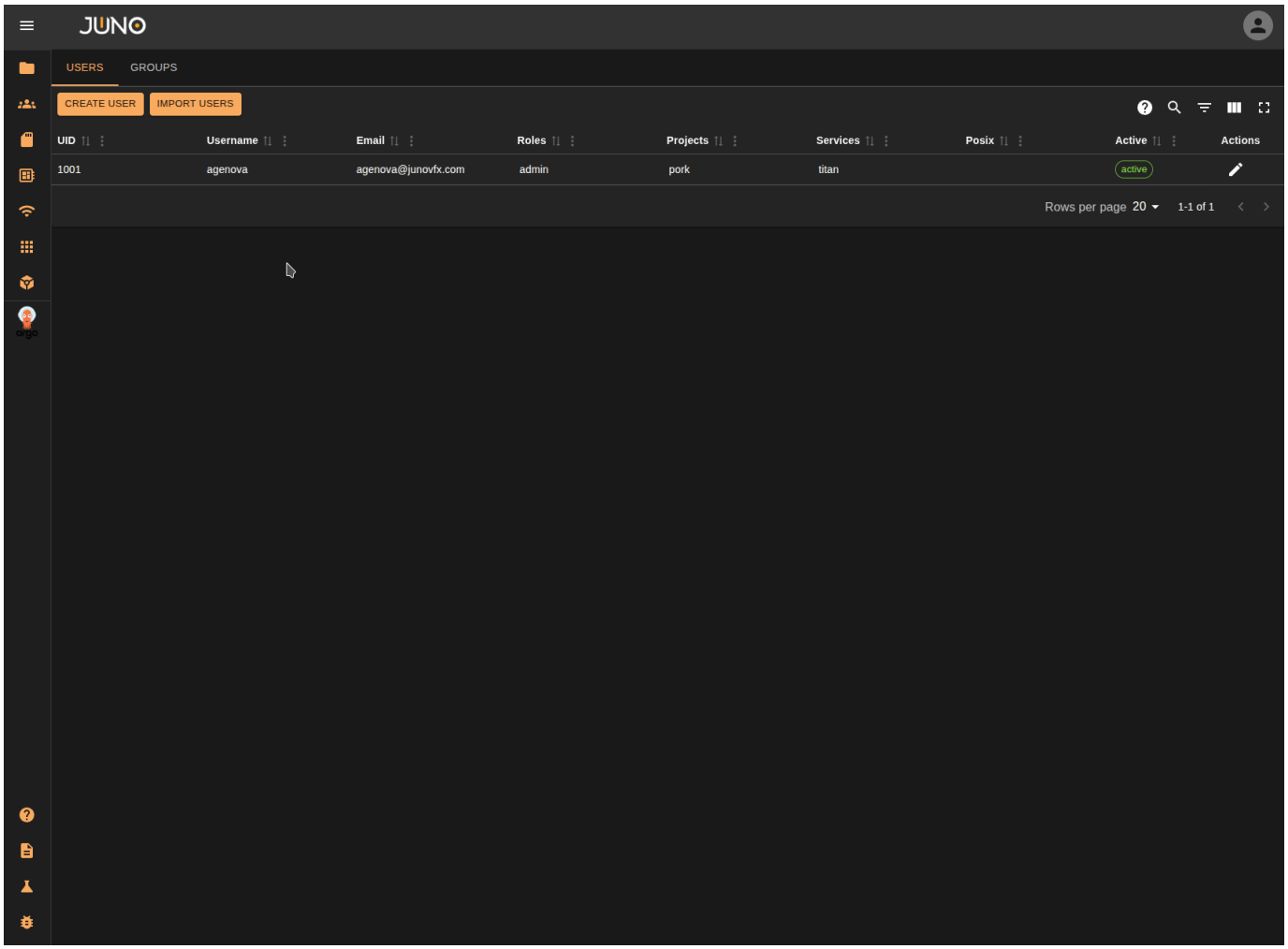
4.1.4 User Management

> **Note**
>
> User management is restricted to members of 'admin' and 'titan' user groups. Each table will have a help icon located on the top right of the table. Click this icon to start the help stepper.

The user management page allows users to oversee and manage all users and groups.

User Creation

At the top left of the user table, there is a button labeled "CREATE USER." Clicking this button opens a modal with three fields that must all be filled out to create a user. Usernames must only contain lowercase letters, numbers, or hyphens, and cannot include capital letters or special characters.



Import Users

You can import users via a CSV file or a JSON file.

CSV FILE REQUIREMENTS

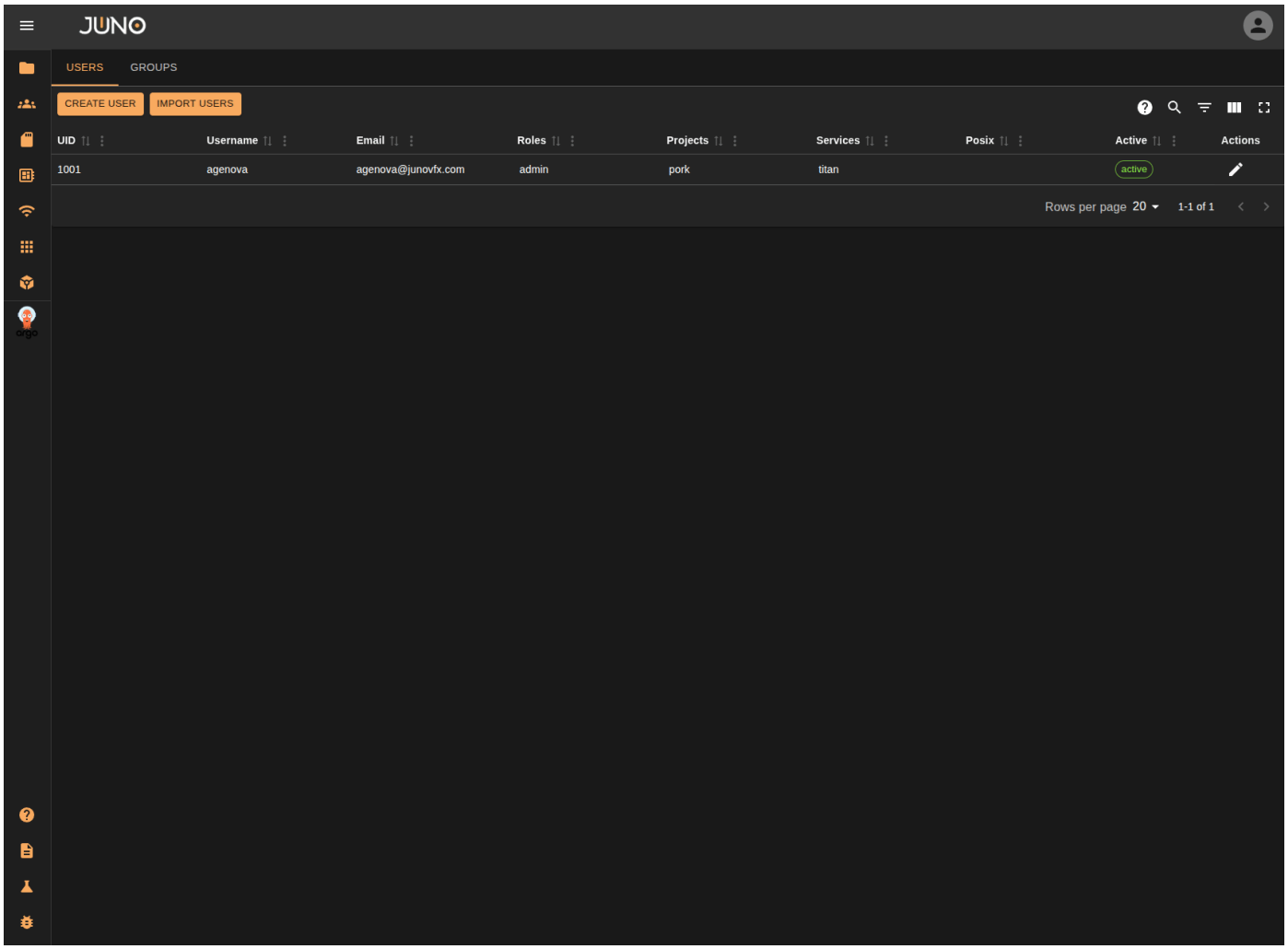
- The file must include three headers: `uid`, `email`, and `username`.
- An optional active header can be included. If omitted, all users will be created as active by default.
- `Usernames`, `emails`, and `uid` values must be unique within the file.
- All rows must have non-empty values for `uid`, `username`, and `email`.

JSON FILE REQUIREMENTS

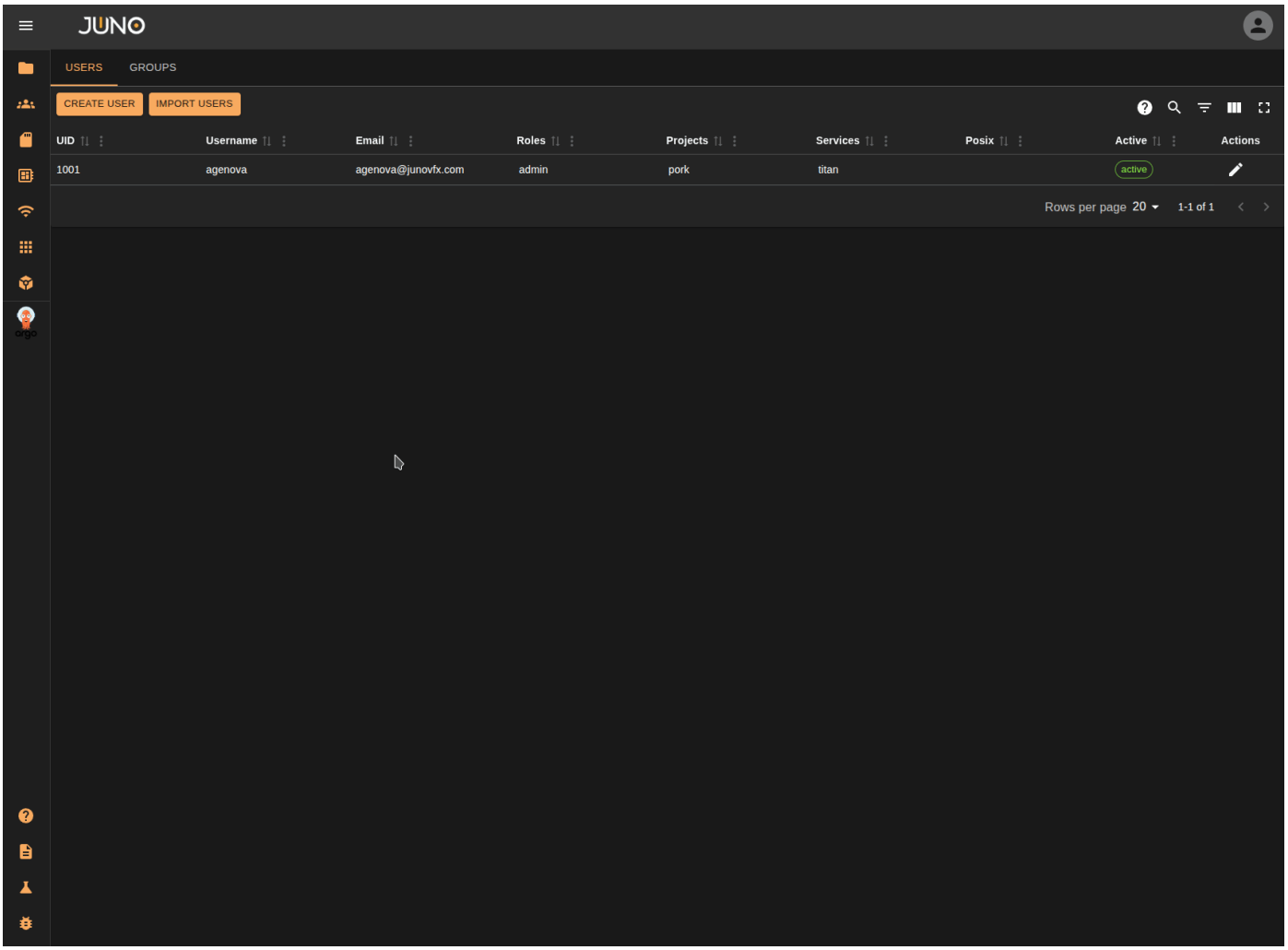
- The JSON file must be a list of user objects. **(See code snippet below for the JSON structure)**
- The active field is optional and defaults to true for newly created users.
- All fields must have non-empty values; no blank fields are allowed.
- Duplicate values are not permitted in the JSON file.

```
[
    {
        "username": "newuser",
        "email": "newuser@newuser.com",
        "uid": 1234,
        "active": true
    },
    {
        "username": "newusertwo",
        "email": "newusertwo@newuser.com",
        "uid": 6789,
        "active": false
    }
]
```

CSV Import



JSON Import

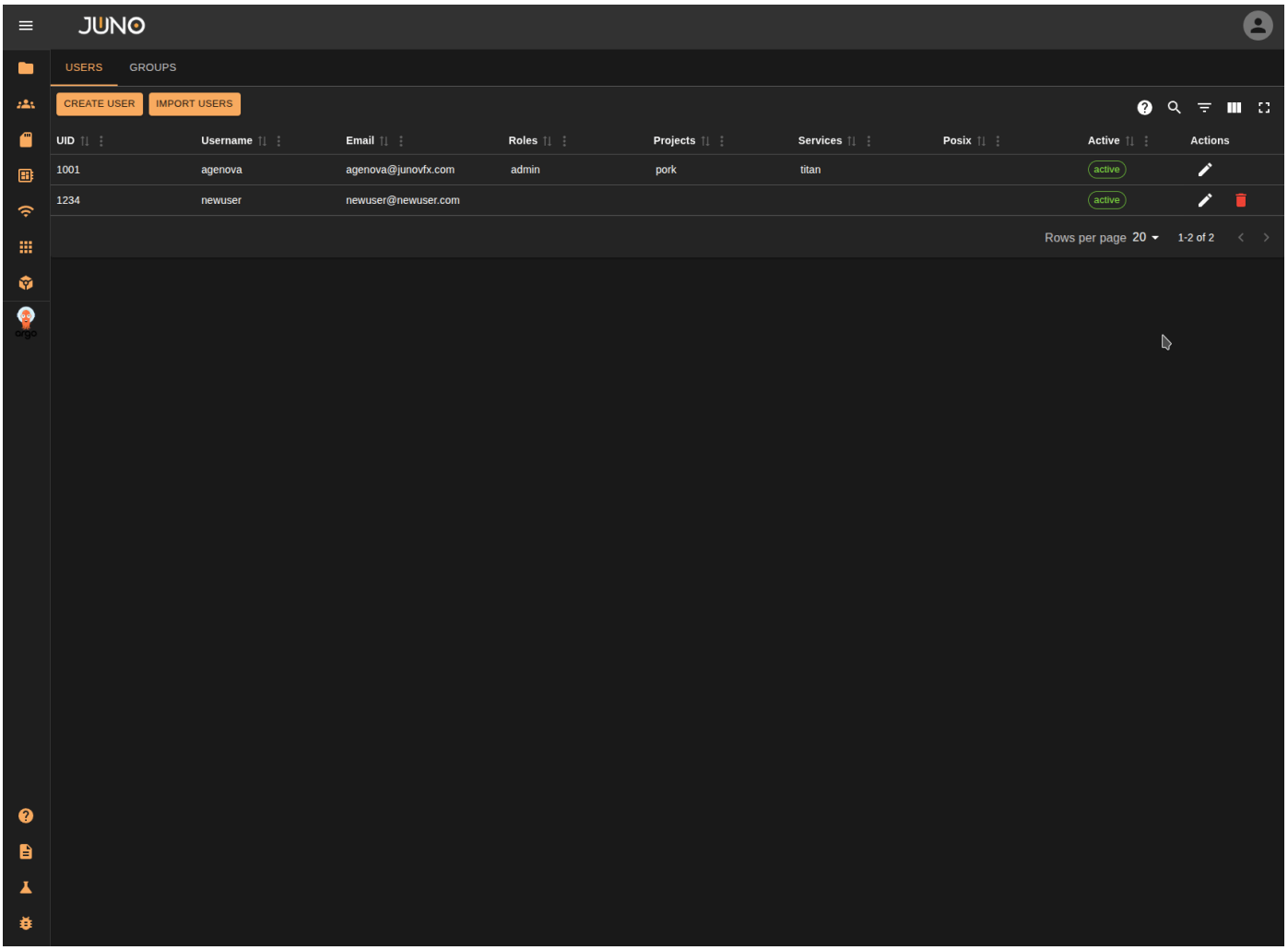


Editing a User

Each user can be edited by clicking the pencil icon at the end of their row.

> **Warning**
>
> You cannot modify the active status of your own user account; this must be done by another admin or titan role.

Edit User



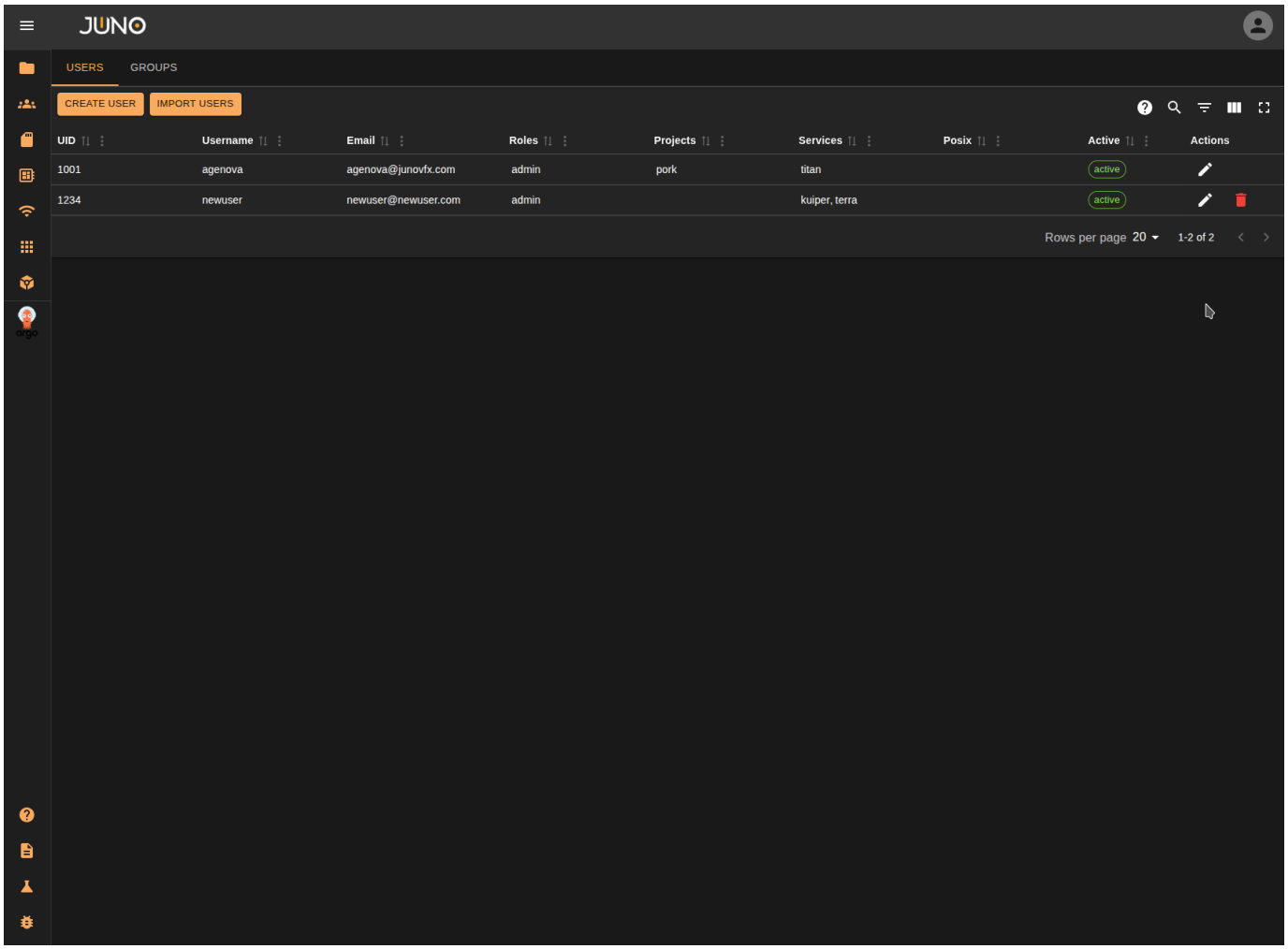
USER GROUP ASSIGNMENT

There are four group types that a user can be assigned to. Once a user is created you have the ability in the user table to assign or unassign that user to existing groups. You can also assign multiple users to one group at a time in the group table

SELECTING AND DESELECTING GROUPS

Clicking the edit user pencil and choosing the desired group type opens a popup showing all available groups for that type. Highlighted groups indicate the ones the user is currently assigned to, while non-highlighted groups represent those the user is not assigned to. To select or deselect groups, hold down `Ctrl` and left-click on the desired groups in the popup. Once you have made your selection, click the green checkmark button to confirm the changes. Finally, ensure you click the save button at the end of the row to apply all changes to the user.

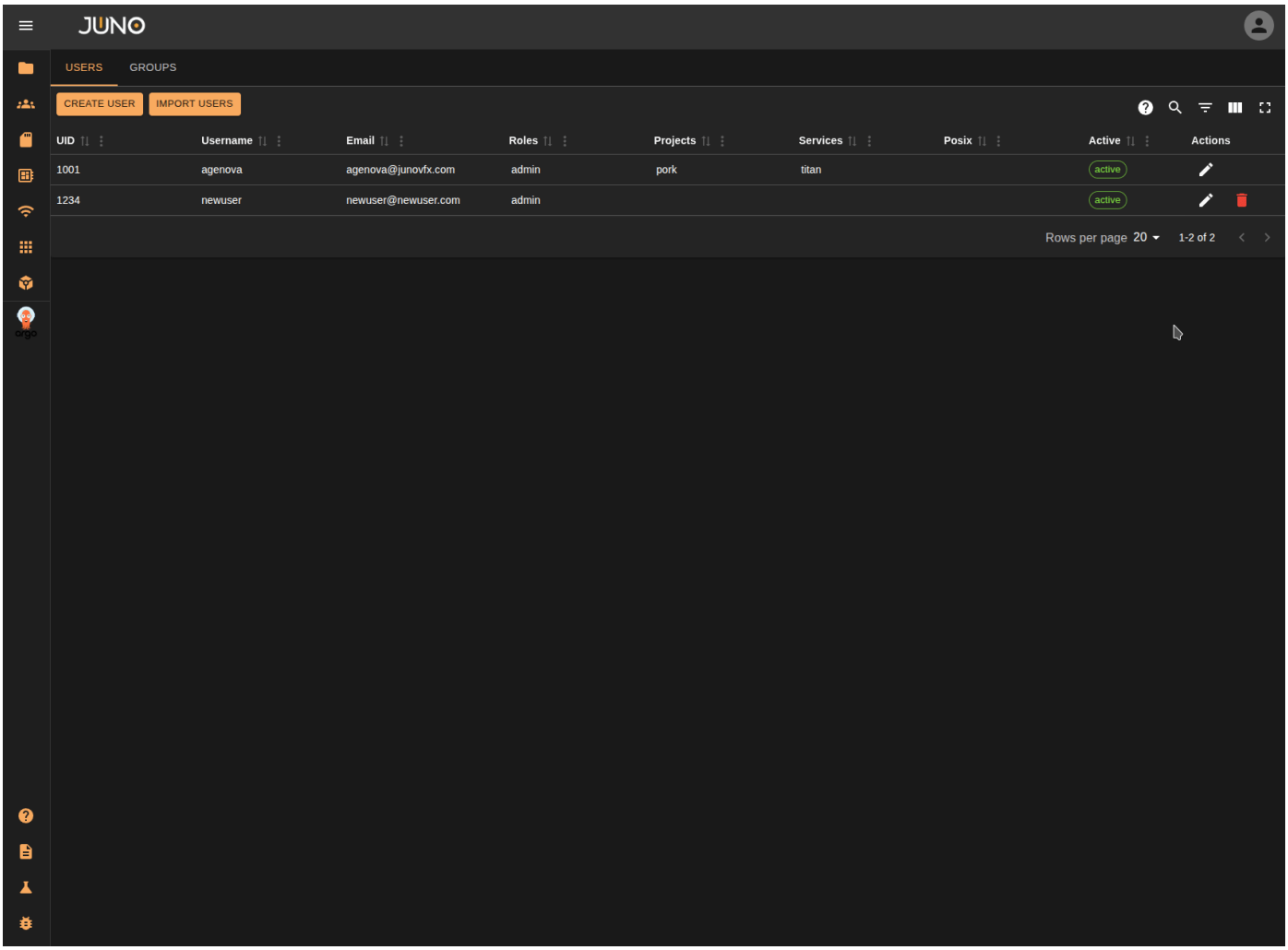
Using CTRL click to select/deselect



SELECTING MULTIPLE GROUPS AT ONCE

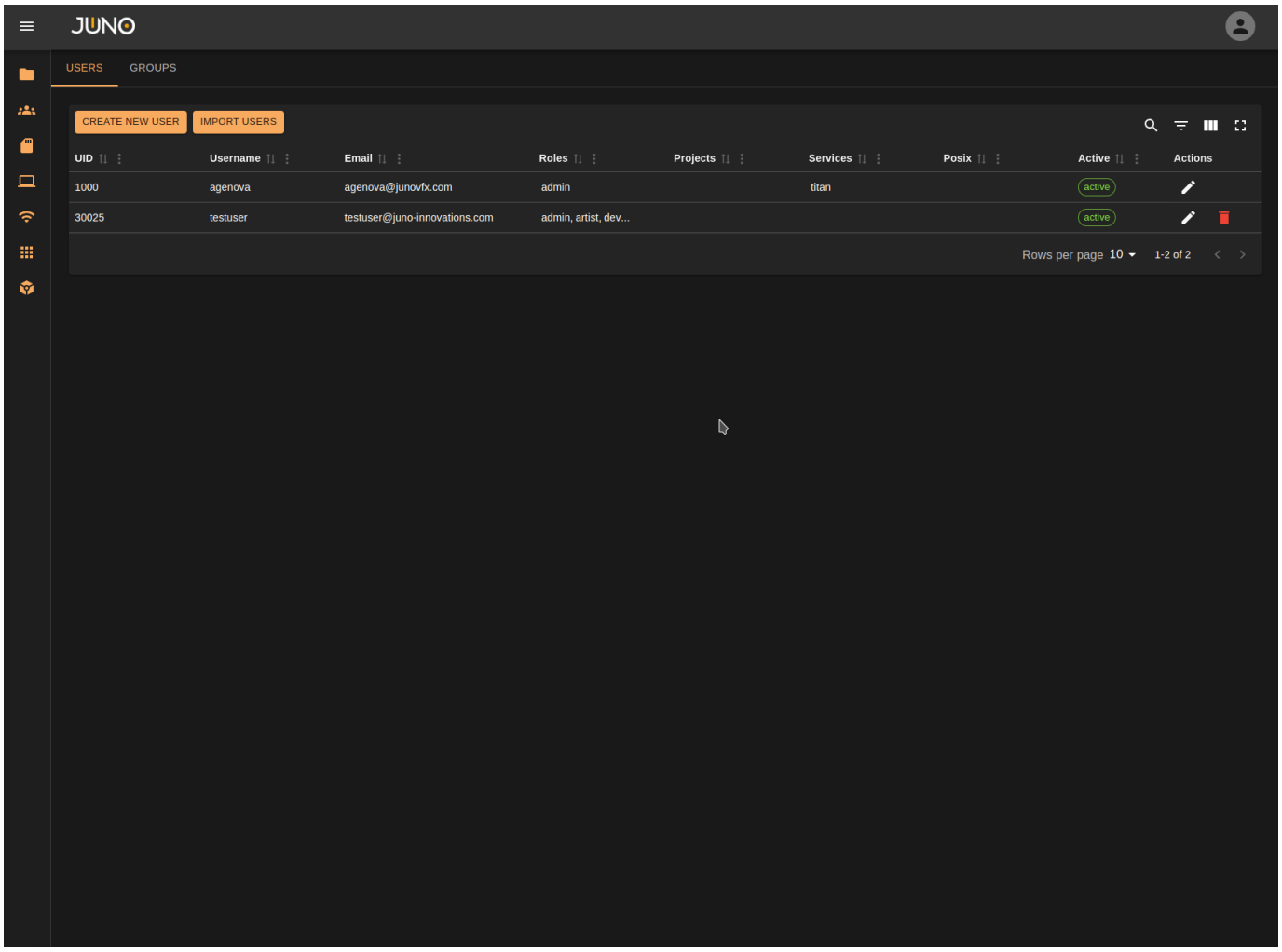
Hold down Shift, click the first group in your batch selection, then click the last group. All groups between the first and last will be selected.

Using SHIFT click to multi-select



Group Creation

To view the groups table, select the groups tab in the top left of the usermanagement page. Click the CREATE NEW GROUP button, a pop up will appear with a drop down and group name input. You are only able to create the group type role or posix. Group names can only contain lowercase letters and hyphens, no special characters allowed



Group Types

Group Type	Description
Role	User defined roles; no UID.
Posix	Allows users to have file/directory ownership.
Service	Created programmatically when a service is enabled; membership grants access to that service.
Project	Created programmatically during project creation; membership grants access to that project.

Import Groups

You can import groups via a CSV file or a JSON file.

CSV FILE REQUIREMENTS

- The CSV file must contain three headers: `uid`, `type`, and `name`. If all your groups are of type `role`, the `uid` column is not required. For mixed `role` and `posix` groups, leave the `uid` field blank for `role` groups.
- Only two group types are supported for importing: `posix` and `role`.
- All `posix` group types must have a `uid` value.
- Duplicate group names and UIDs are not allowed in the CSV file.
- The `members` column is optional. All listed members must already exist in the system. Usernames should be separated by commas. Both of the following formats are valid:

Valid CSV member list:

```
userone, usertwo, userthree
```

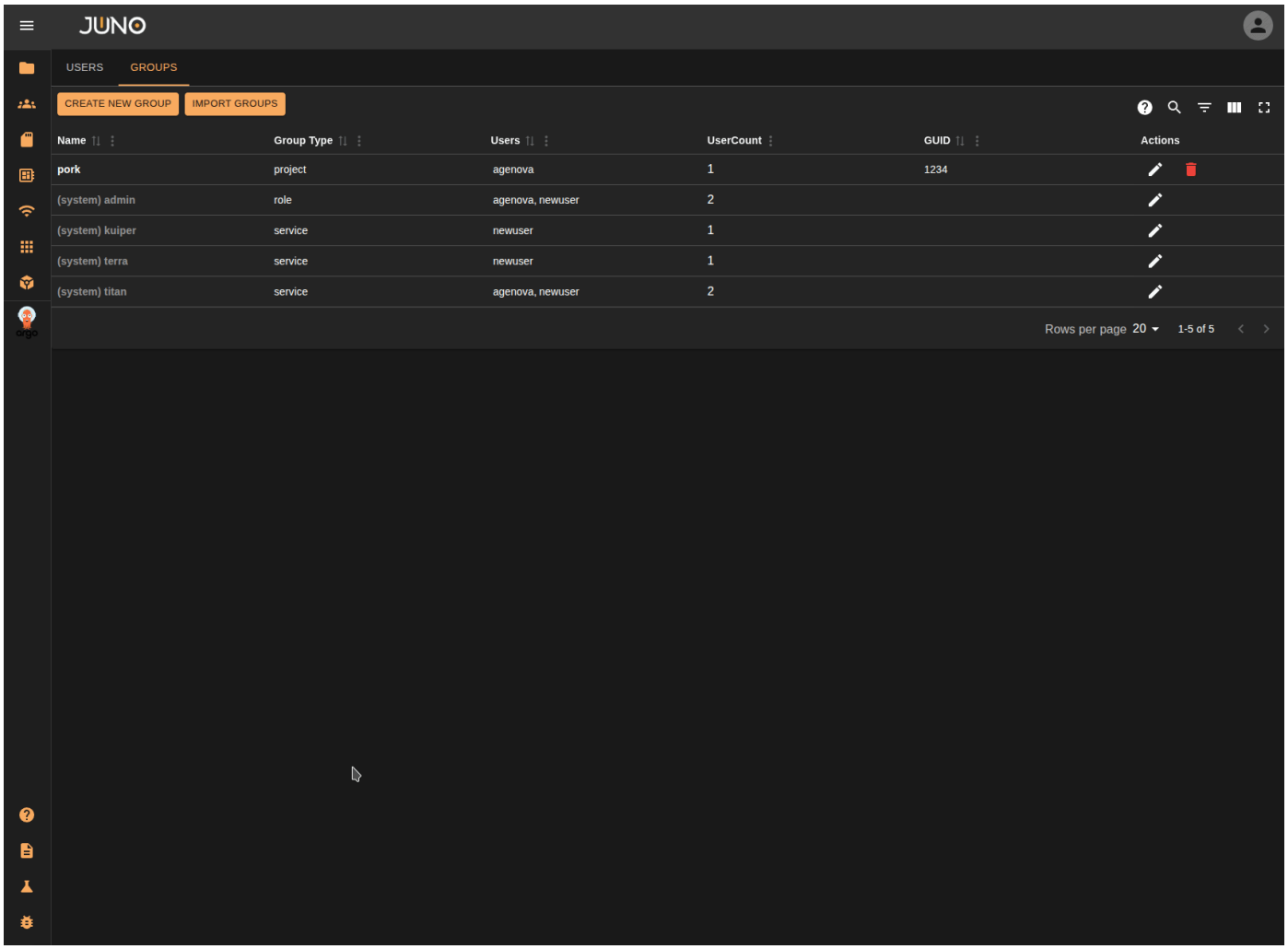
```
userone,usertwo,userthree
```

JSON FILE REQUIREMENTS

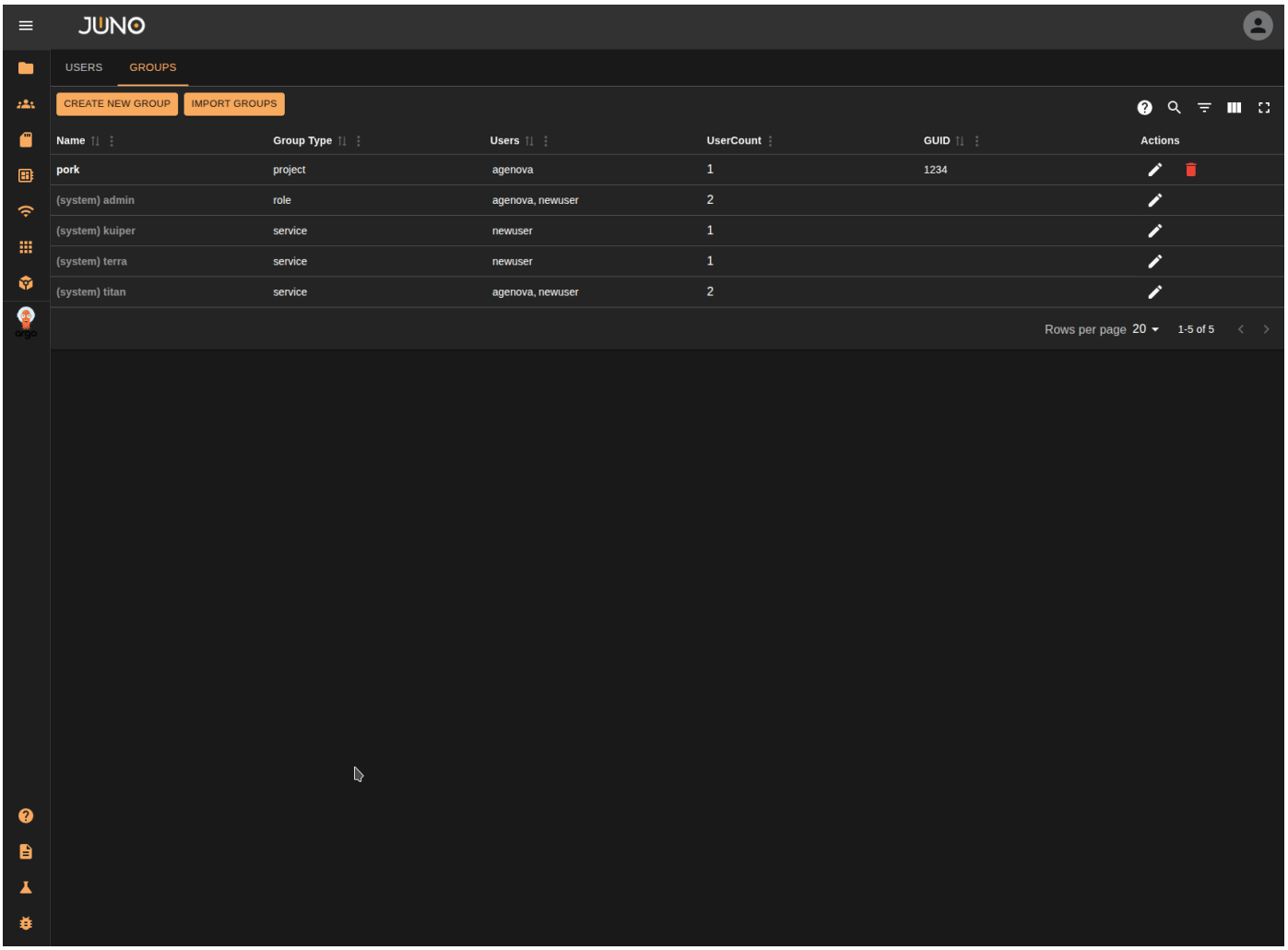
- The JSON file should contain a list of group objects. **(See the code snippet below for the required structure.)**
- The `members` field is optional.
- Groups of type `role` do not require a `uid` field.
- All group names and UIDs must be unique; duplicate values are not allowed.

```
[
    {
        "name": "rolegroup",
        "type": "role",
        "members": ["userone", "usertwo", "userthree"]
    },
    {
        "name": "posixgroup",
        "type": "posix",
        "uid": 1234,
        "members": ["userone", "usertwo", "userthree"]
    }
]
```

CSV Import



JSON Import



Editing a Group

Once a group is created, Click the pencil Icon to edit the group, you then will be able to change the name and user assignment. Click the save icon to save your changes.

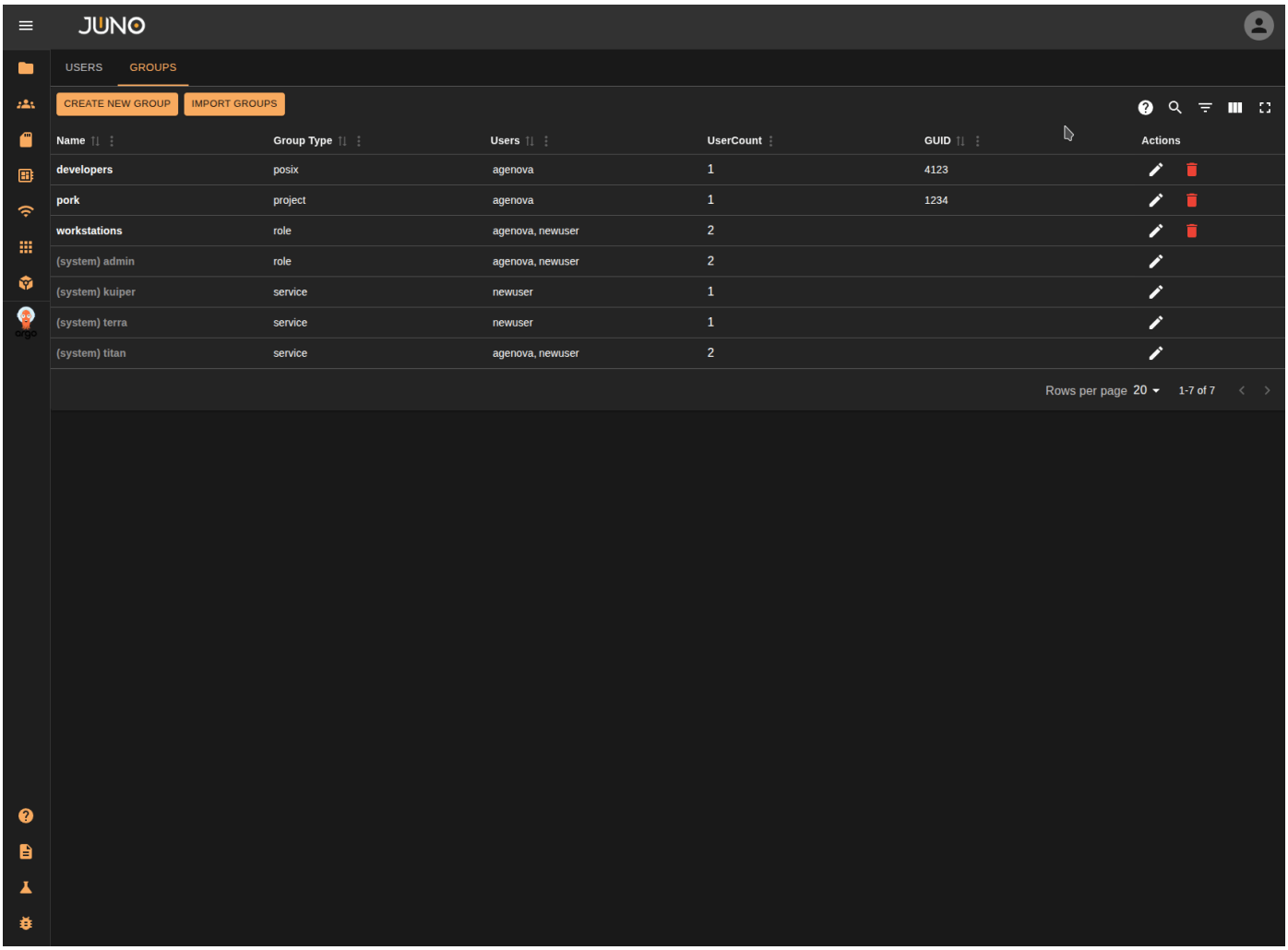
> **Warning**
>
> You are only able to edit the name and the users assigned to a group once it is created.

Assigning users to a group

To change what users are assigned to a group, Click the pencil icon on the group row to enable the editing. Then click on the user column which will display a popup. This pop up shows all available users. Highlighted users indicate the ones that are assigned to the group, while non-highlighted users represent those that are not assigned to the group. To select or deselect users, hold down 'Ctrl' and left-click on the desired user in the popup. Once you have made your selection, click the green checkmark button to confirm the changes. Finally, ensure you click the save button at the end of the row to apply all changes to the group.

SELECTING AND DESELECTING USERS

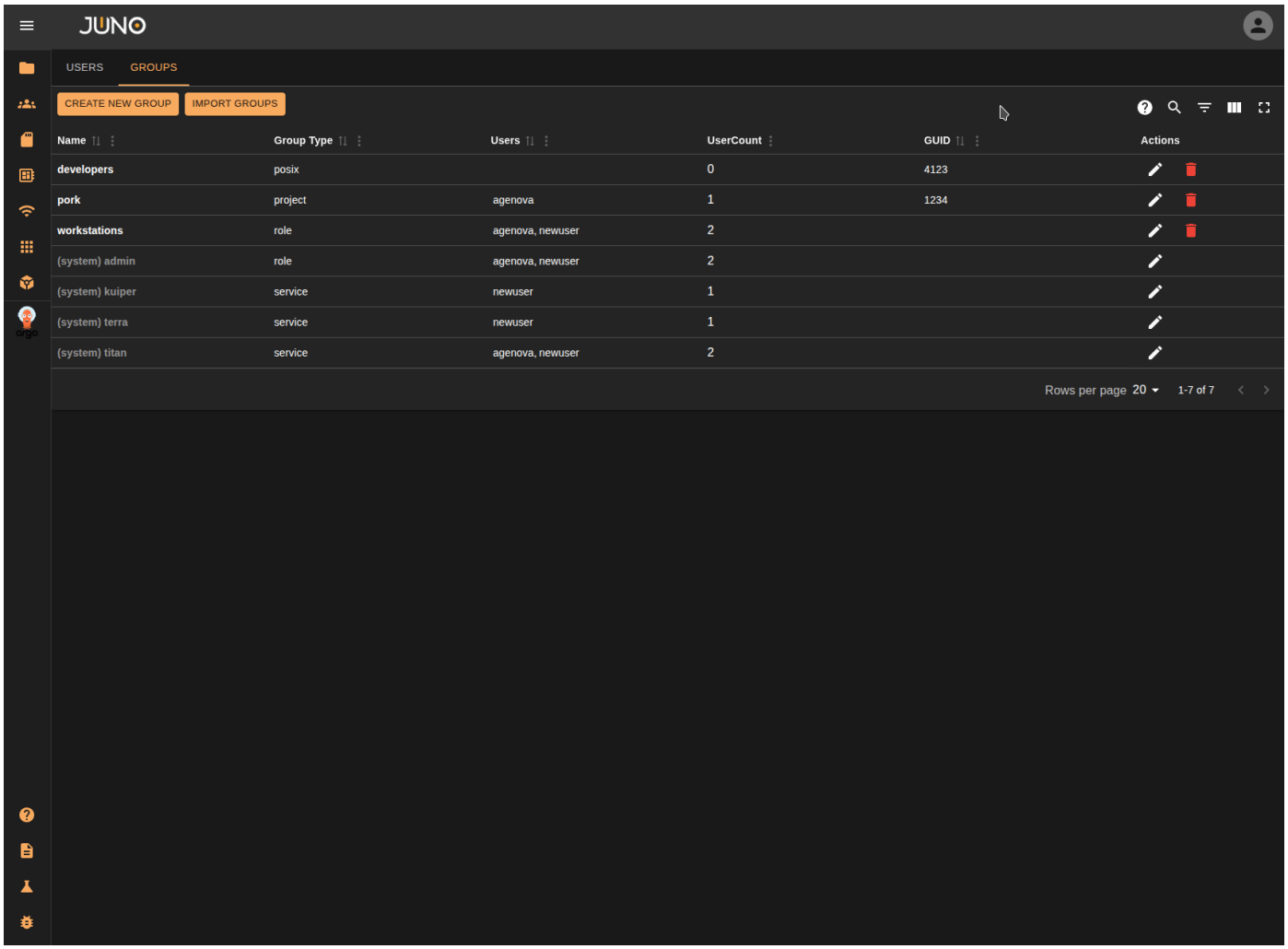
Using CTRL click to select/deselect



SELECTING MULTIPLE USERS AT ONCE

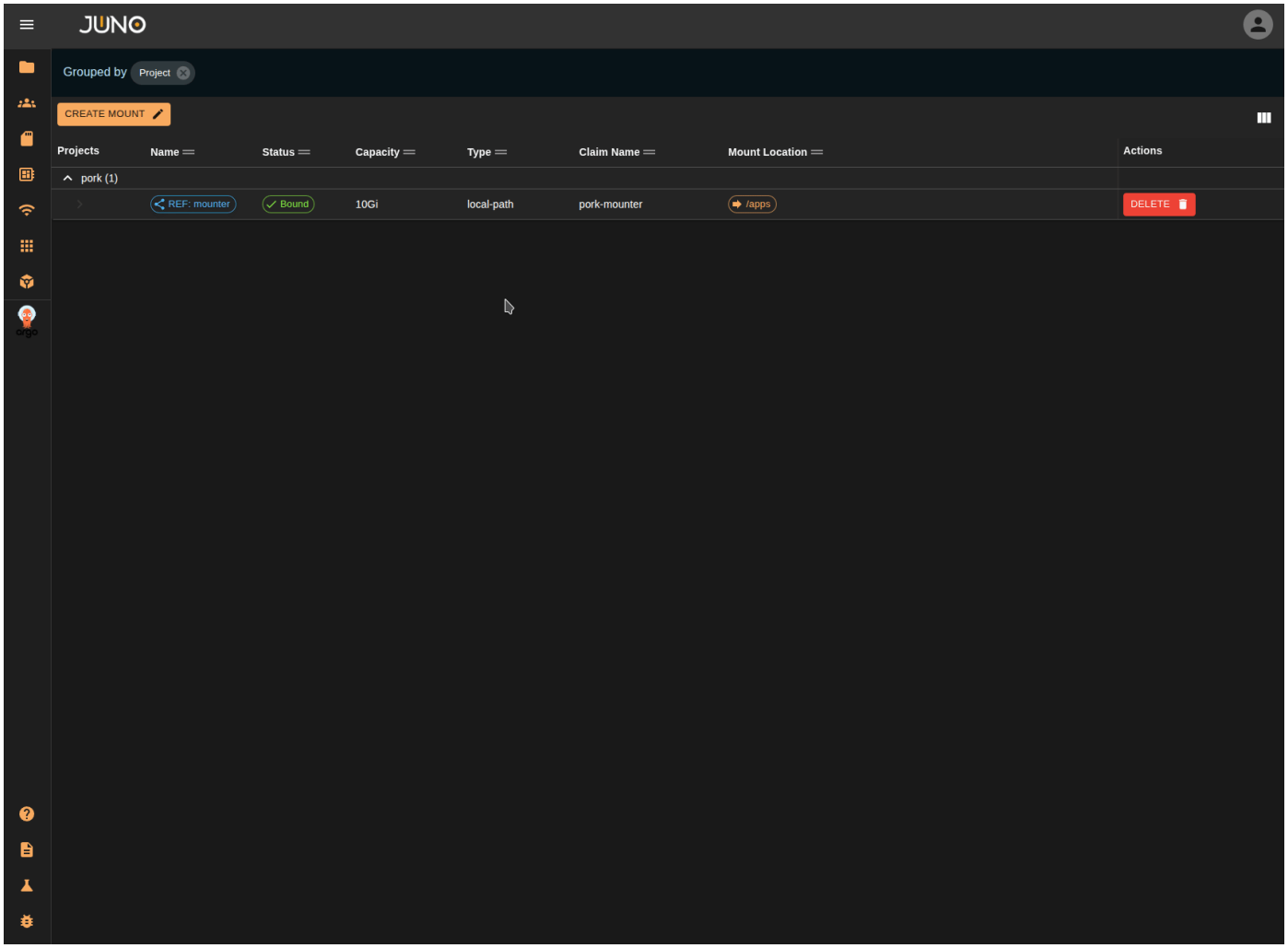
Hold down Shift, click the first user in your batch selection, then click the last user. All users between the first and last will be selected.

Using SHIFT click to multi-select



4.1.5 Storage Management

The storage management page allows admins to create, edit, and delete storage mounts inside each Orion deployment. Storage mounts are used to persist data across Orion deployments, and can be used to store user data, project data, and other files. The underlying engine is the Kubernetes CSI drivers that you have configured on your cluster. Genesis aims to not only make managing this easier, but also add features that are difficult to do with just the Kubernetes API.



Persistent Volume Creation (PV)

Because Juno uses Kubernetes as its backend, you can use any kind of storage that is supported via Kubernetes directly or via a CSI. Here a few examples of Persistent Volumes using common mount techniques.

- NFS
- ISCSI
- HostPath
- Many many more...

Juno only mounts PV's and does not actually manage them directly unless using the built-in "Quick Mount" setup. In our example, we will use the following PV spec and apply it to our cluster using a simple HostPath.

my-pv.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-pv
spec:
  capacity:
    storage: 10Ti
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
  hostPath:
    path: /tmp
```

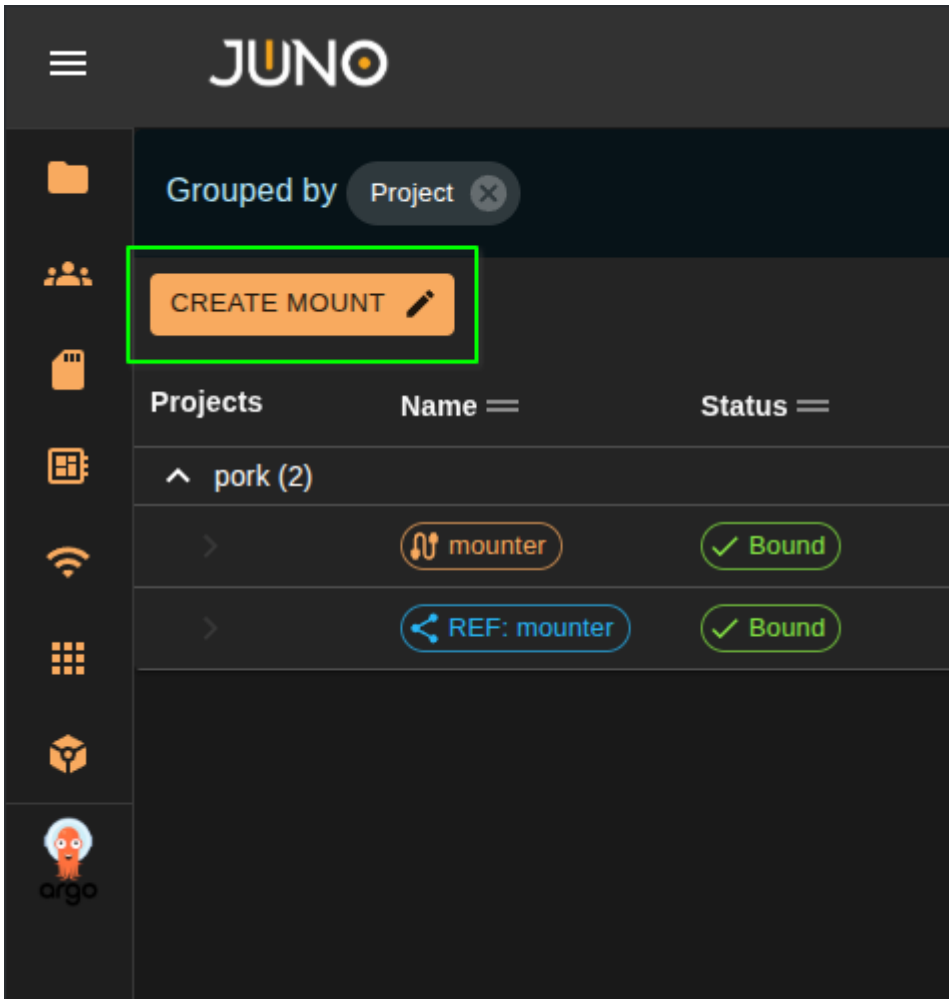
```
$ kubectl apply -f my-pv.yaml
```

Ad-Hoc Volumes

While we can add volumes ad-hoc like this, we recommend pushing your PV's to a GitOps repository in production.

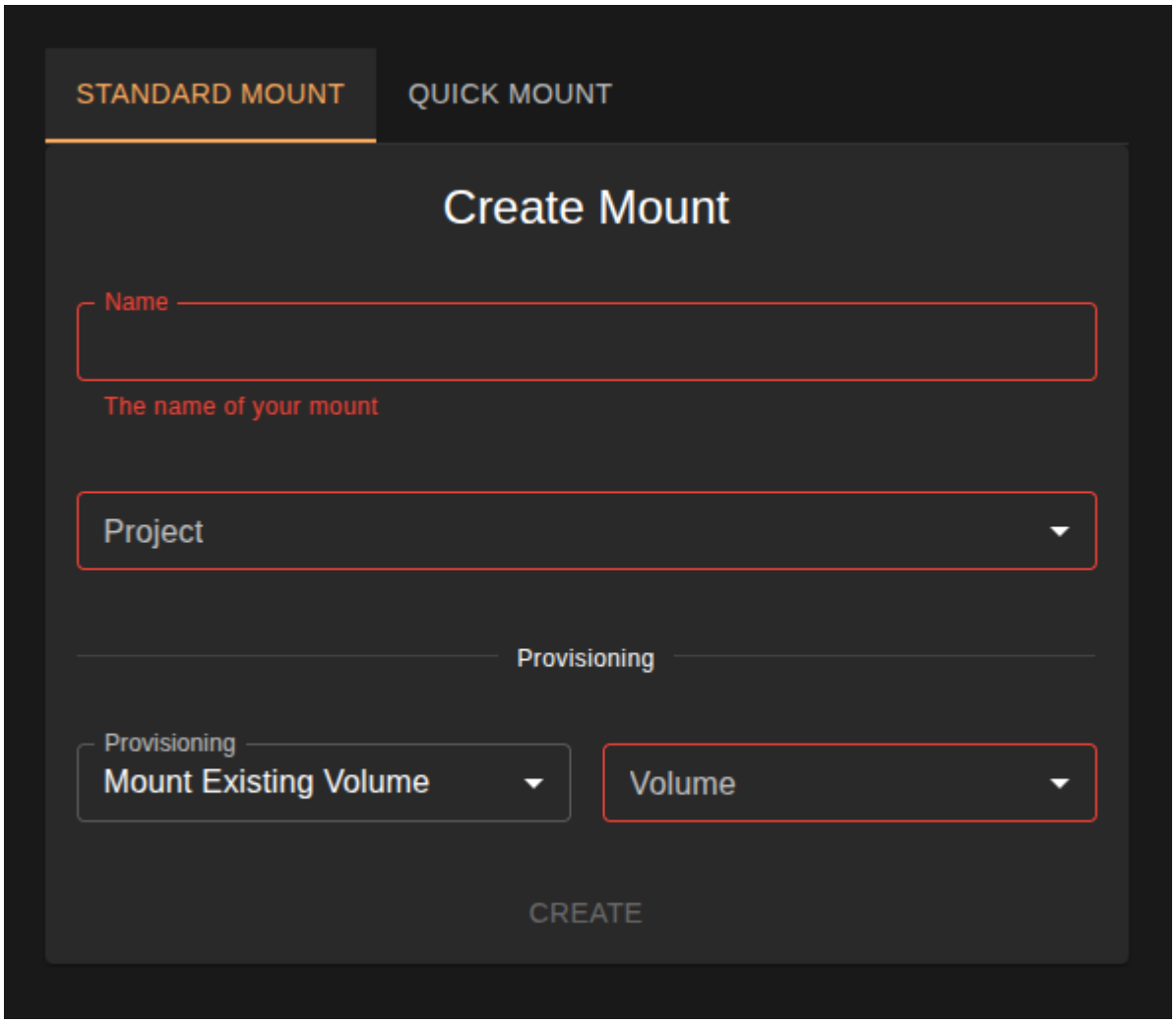
Mount Creation

Click the "Create Mount" button to create a new storage mount.



MOUNT FORM

You will be directed to the storage form where you can fill out the details for the new storage mount.



NAME FIELD

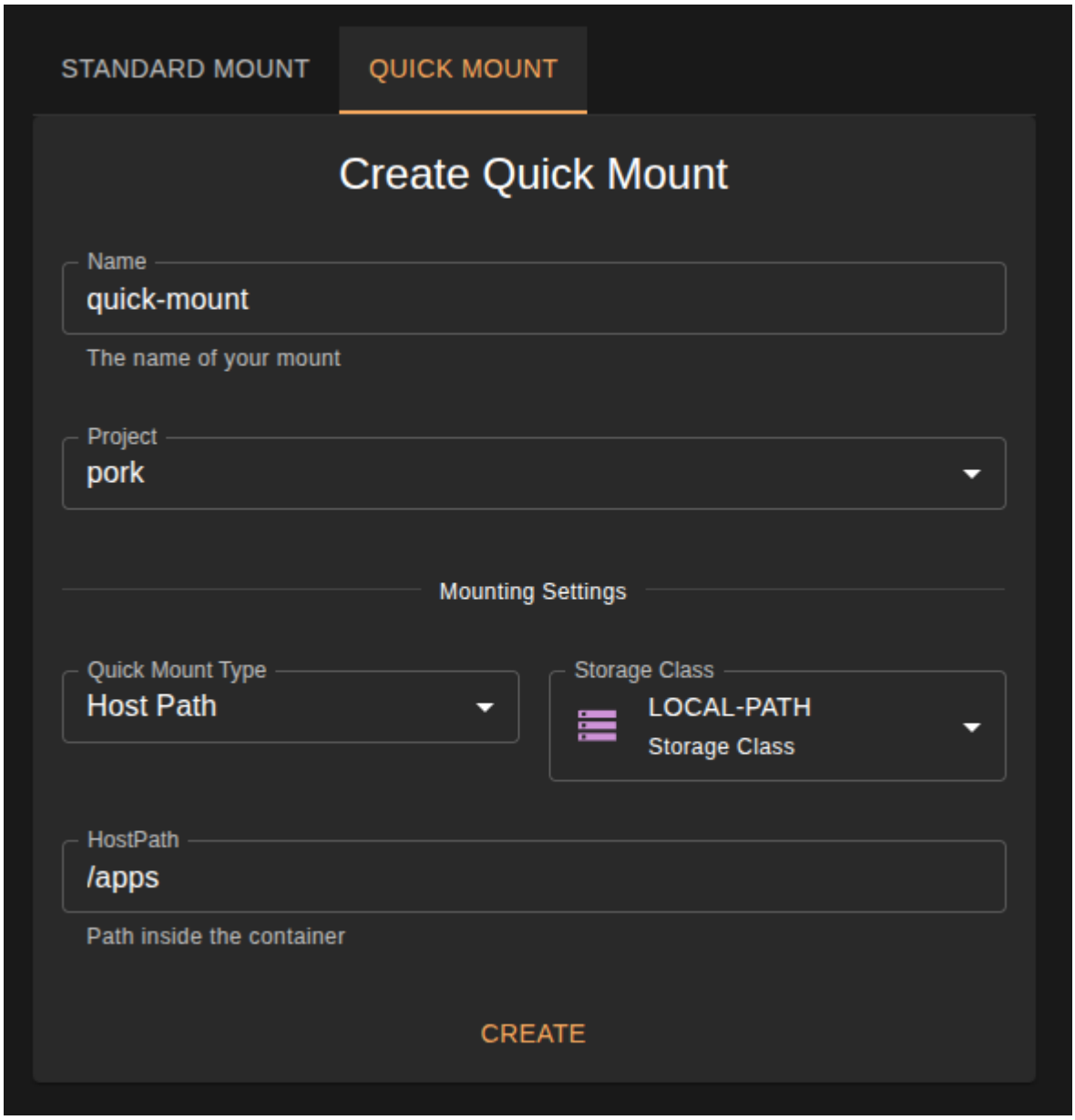
Juno will create a Kubernetes Persistent Volume Claim (PVC) for you, so you will need to provide a name for the PVC. The form will validate the name you put and ensure it is a valid Kubernetes name.

PROJECT FIELD

Mounts are created per-project. This isolates the mounts across namespaces and will attach the projects code to the name you provided above. This allows the PVC's to be unique.

QUICK MOUNT CREATION

A quick mount is an exclusive mount with a host path.



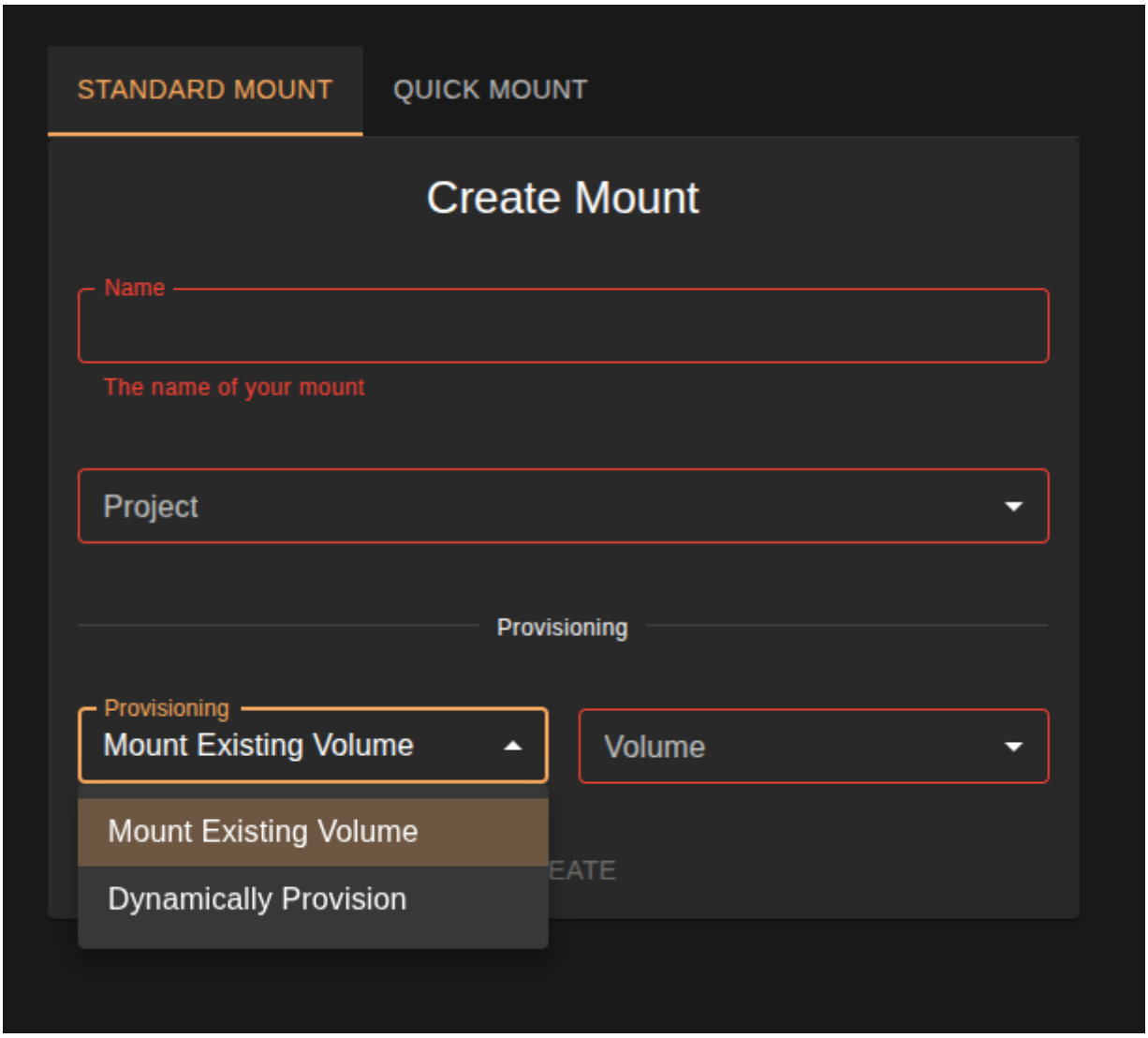
Provisioning

PROVISIONING TYPE

Provisioning type specifies how Juno should create the PVC. The options are:

- **Mount Existing Volume** - Create a bound mount to an existing Volume. i.e. the HostPath we created above
- **Dynamically Provision** - Use a StorageClass to dynamically provision storage on the fly. This is very common in cloud environments. i.e. `gp3`

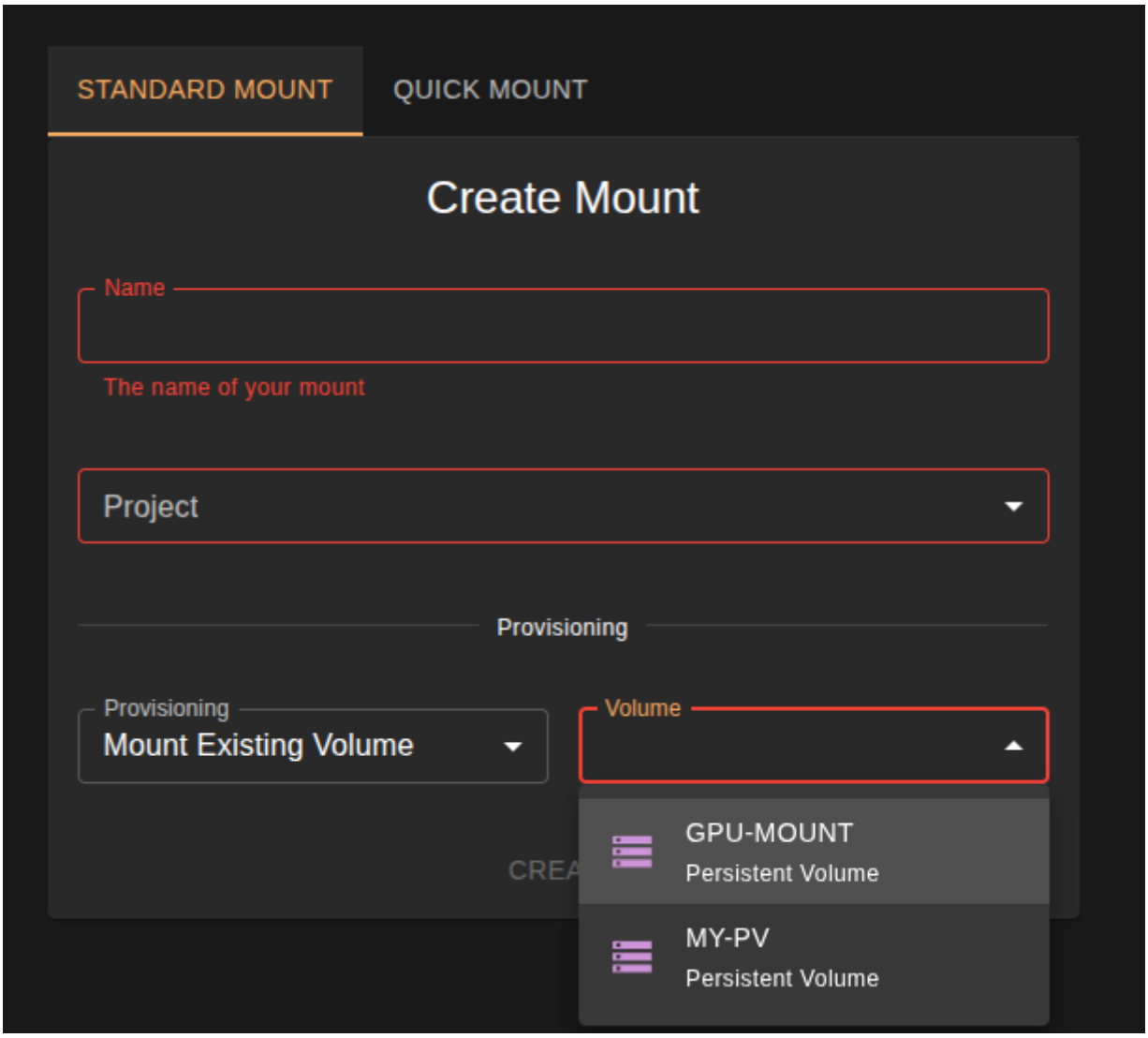
In most on-prem environments, you will want to use the "Mount Existing Volume" option and specify the Persistent Volume you created to be mounted.



MOUNT EXISTING VOLUMES

Volume Selection

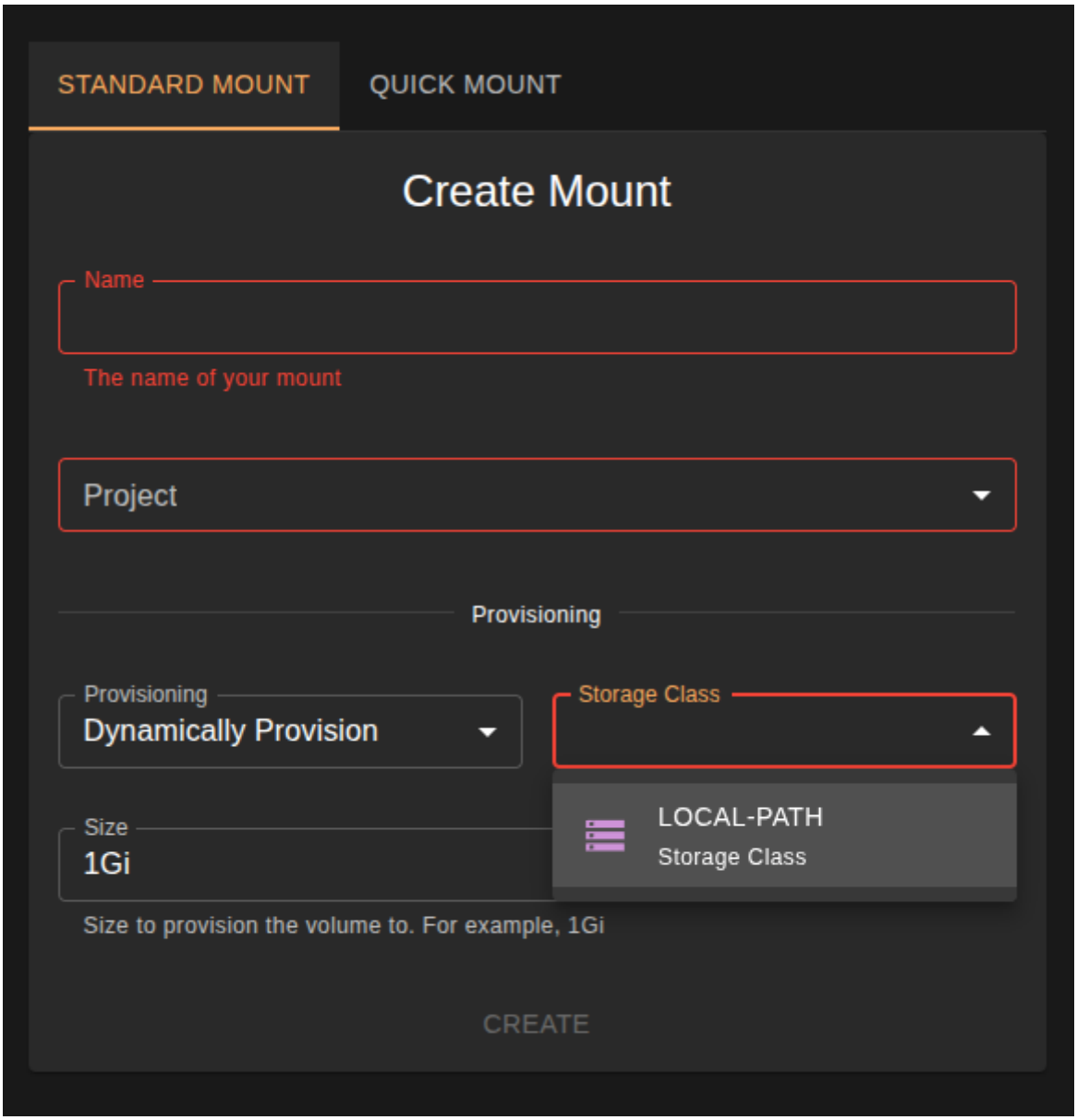
Juno will detect all unbound Persistent Volumes in your cluster and display them in a dropdown. You can select the volume you want to mount.



DYNAMICALLY PROVISIONED VOLUMES

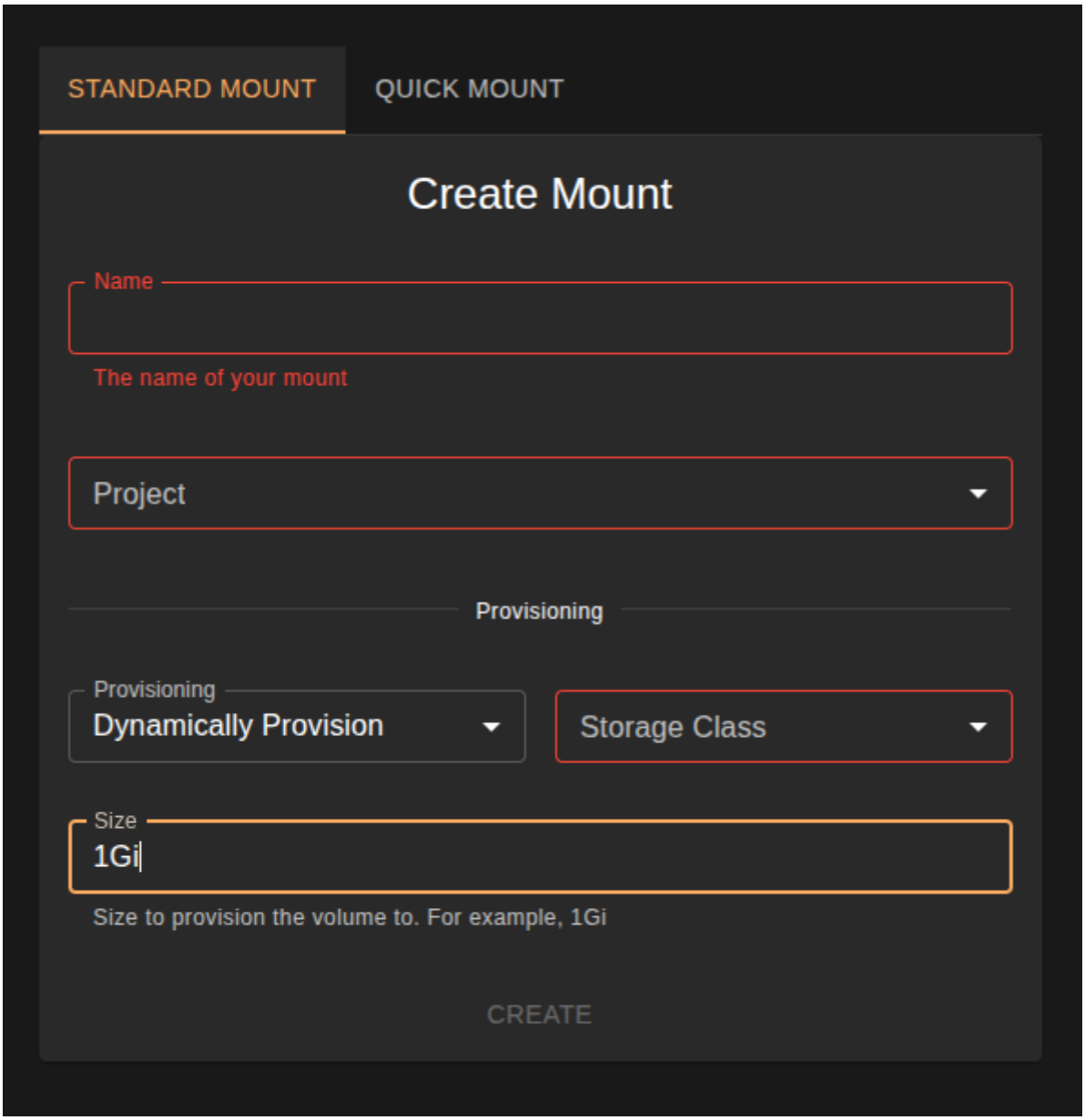
Storage Class Selection

Dynamic Provisioners will use a StorageClass to provision the PVC. Genesis will display all available StorageClasses in your cluster and you can select the one you want to use. If you do not have any StorageClasses, you will need to create one first. For common cloud providers, this is usually included in the Kubernetes distribution. For EKS on AWS, this can be `gp3` for example. To learn more about the storage classes available in your cluster, you can refer to the CSI documentation for that Storage Class from the provider.



Size Field

The size field specifies the size of the PVC to be created. This is required for creating the volume size for the storage class. This is only used if the storage provider uses it.



Mounting Options

EXCLUSIVE

By default, mounts are created to be "exclusive". Meaning, they are expected to be consumed by a single workload in the project. You can also create an exclusive mount by applying a PersistentVolume Manifest via kubectl to your cluster. Make sure the accessModes has ReadWriteOnce

pv-exclusive.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: test-pv
spec:
  capacity:
    storage: 2Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
```

```
  hostPath:
    path: /juno-apps
```

Data Protection

Exclusive dynamically provisioned volumes are deleted as well as their data in most cases. Juno will inherit whatever the CSI driver sets upstream. If you need to customize this behavior, we recommend you manually create the Persistent Volume and then use the "Mount Existing Volume" option to mount it. This will allow you to set the `persistentVolumeReclaimPolicy` to `Retain` and ensure that the data is not deleted when the PVC is deleted.

SHARED

Shared mounts are treated as a shared storage location that all workloads will mount. For example, if you have software that is shared across all workloads, you can create a shared mount and then mount it to all workloads in the project. A common use case is to use this to mount a shared NFS share across all workloads. You can also create a HostPath pass-through volume that will allow you to install software on the host machine and then pass it through to the workloads. This allows you to reuse existing software provisioning systems like Ansible, Puppet, or Chef to install and prepare the host, then pass that on to the containers which drastically speeds up provisioning times and provides a "hybrid" workload solution.

Terra will present this volume as an option during software installs via Terra Plugins. This is very useful when creating software shares or even project data.

Data Protection

Shared volumes are not deleted when the PVC is deleted. This is because they are expected to be shared across multiple workloads and projects. If you need to delete the data, you will need to do so manually. This is a common use case for shared NFS shares or HostPath pass-through volumes.

Create

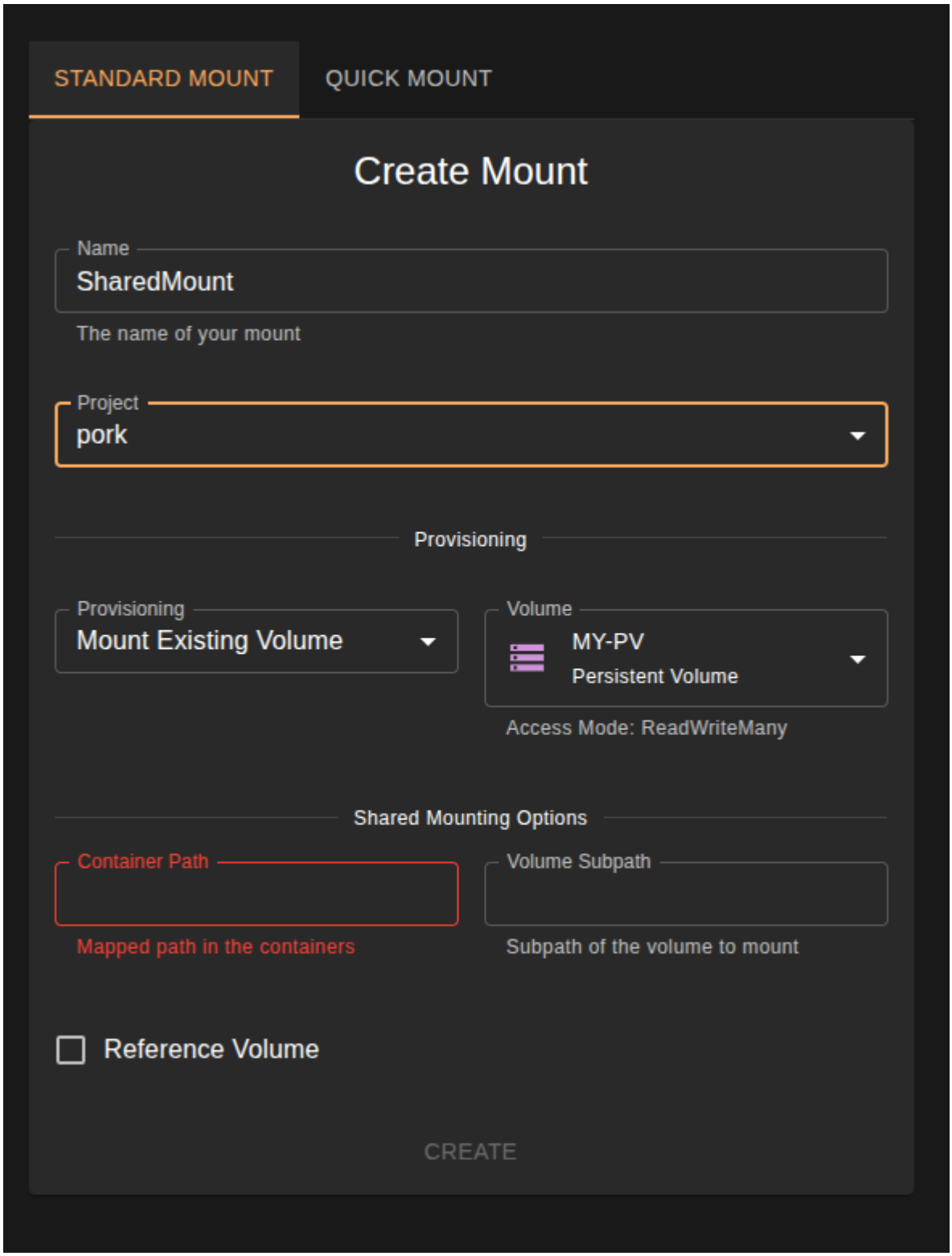
You can create a shared mount by applying a persistent volume manifest to your cluster via kubectl with the accessModes set to ReadWriteMany.

pv-shared.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-pv
spec:
  capacity:
    storage: 10Ti
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
  hostPath:
    path: /tmp
```

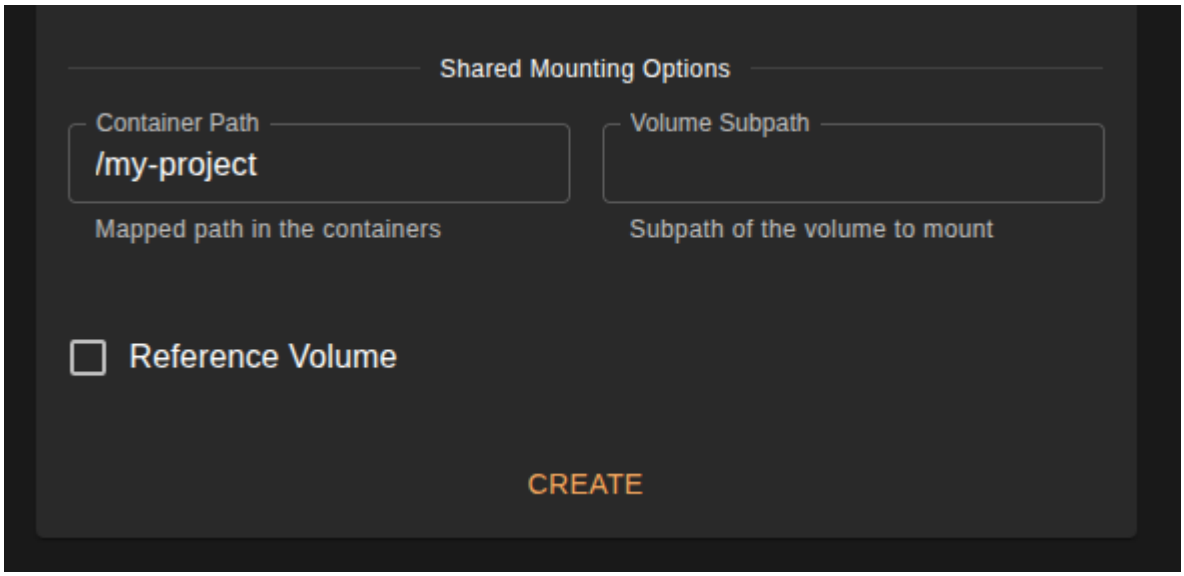
```
kubectl apply -f pv-shared.yaml
```

Once this is applied, Navigate back to the create mount form in Genesis. Fill in the `name`, Select a `project`, Make sure the provisioning dropdown is set to `Mount Existing Volume`, then use the volume dropdown to select the shared mount you created. This will display the `Shared Mounting Options`.



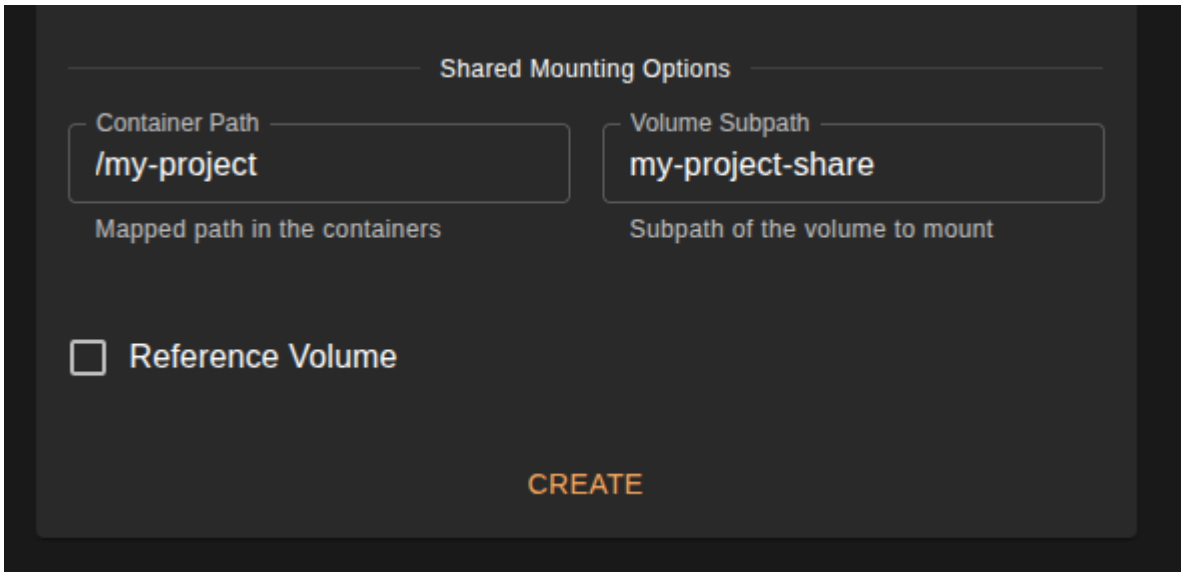
Container Path

Mount the volume to the container at the specified path. (This is passed through to Terra as well) In the example below, we are mounting the volume to /my-project



Volume Subpath

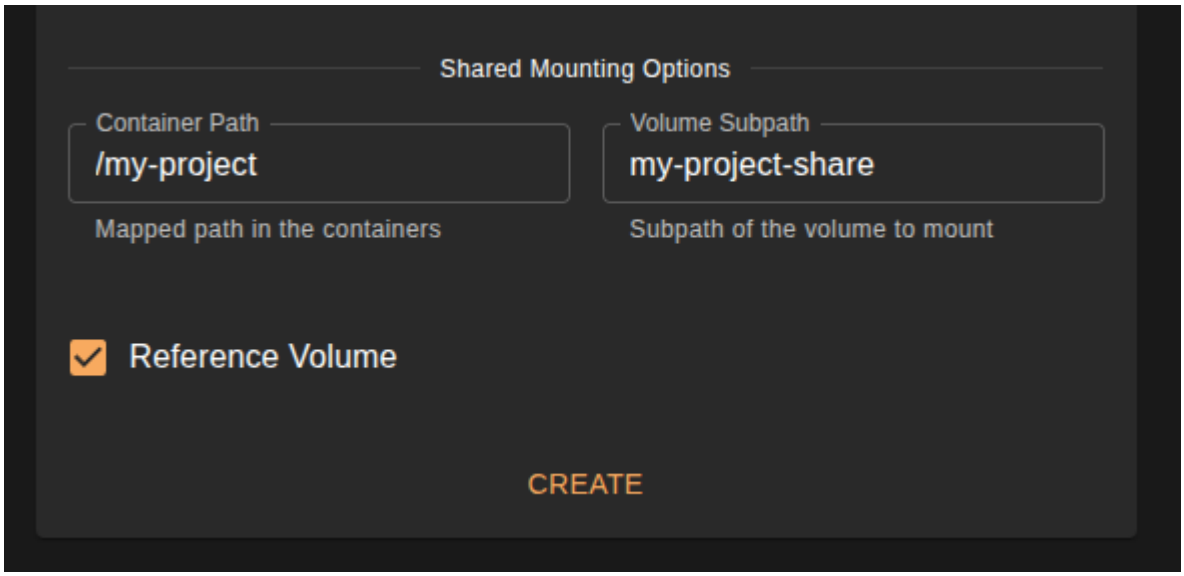
Volume Subpath specifies the location on the Volume to start the mount. In the below example, we are mounting the my-project-share/ location to the container at /my-project. This allows you to mount a specific directory on the Volume to the container, rather than the entire Volume.



Reference Volume

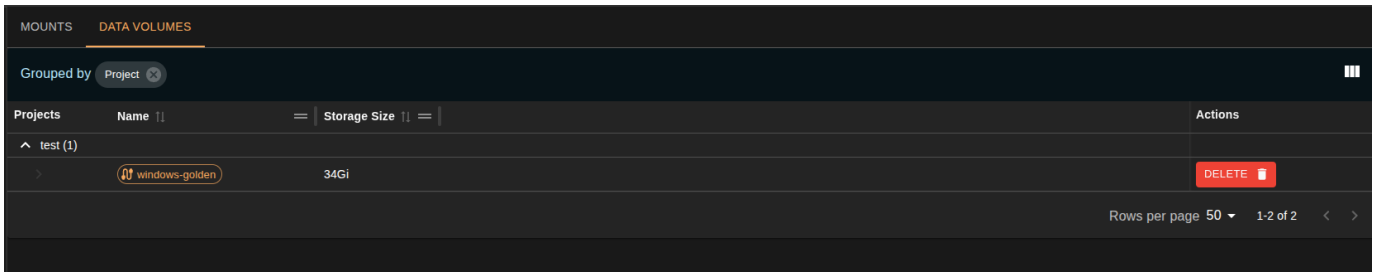
In Kubernetes, Persistent Volumes and the Persistent Volume Claims are a 1:1 match. Persistent Volume Claims can be mounted multiple times, but it can be very confusing and if you want to specify multiple mounts. For example, you have a single NFS server, but you want to mount it to multiple locations in the container. This would mean you would have to do a number of subpath mounts and it can get very messy very quickly. This also limits mount points. Because Persistent Volume Claims are namespaced, this means you would have to have a separate Persistent Volume Claim for each project and obviously a matching Persistent Volume.

Genesis allows for you to "reference" an existing Persistent Volume. By doing this, Genesis will create a duplicated Volume that will match the original Persistent Volume, and it will also tag it with a UUID as well as the project name. This allows you to mount the same Persistent Volume to multiple locations in the container without having to create multiple Persistent Volume Claims. This also allows you to mount the same reference volume in multiple projects across namespaces.



KubeVirt Data Volumes

If KubeVirt is installed in the cluster, an additional Data Volumes tab will appear at the top of the storage table. This tab will provide access to a table showing all Data Volumes in the cluster with the `kuiper.juno-innovations.com/user-created-datavolume` label. This label is automatically generated by Kuiper when cloning a data volume. This process of cloning is part of the Kubevirt VM golden image workflow. You can learn more about our VM golden image workflow here



Real World Examples

SHARED NFS VOLUME

In this example, we will mount an existing NFS server that has the address `nfs-server.example.com` and the path `/`. We have installed a number of shared apps on that NFS server at the path `/shared-apps`. We want to mount these apps to each workload in the project at the `/apps` folder inside the container.

The container mount path will resolve to the following.

```
nfs-server.example.com/shared-apps/ -> /apps/
```

Existing Volume	Subpath	Container Path
test-nfs	/shared-apps	/apps

You can equate this to more normal terms.

Server	Network Share	Container Path
nfs-server.example.com	/shared-apps	/apps

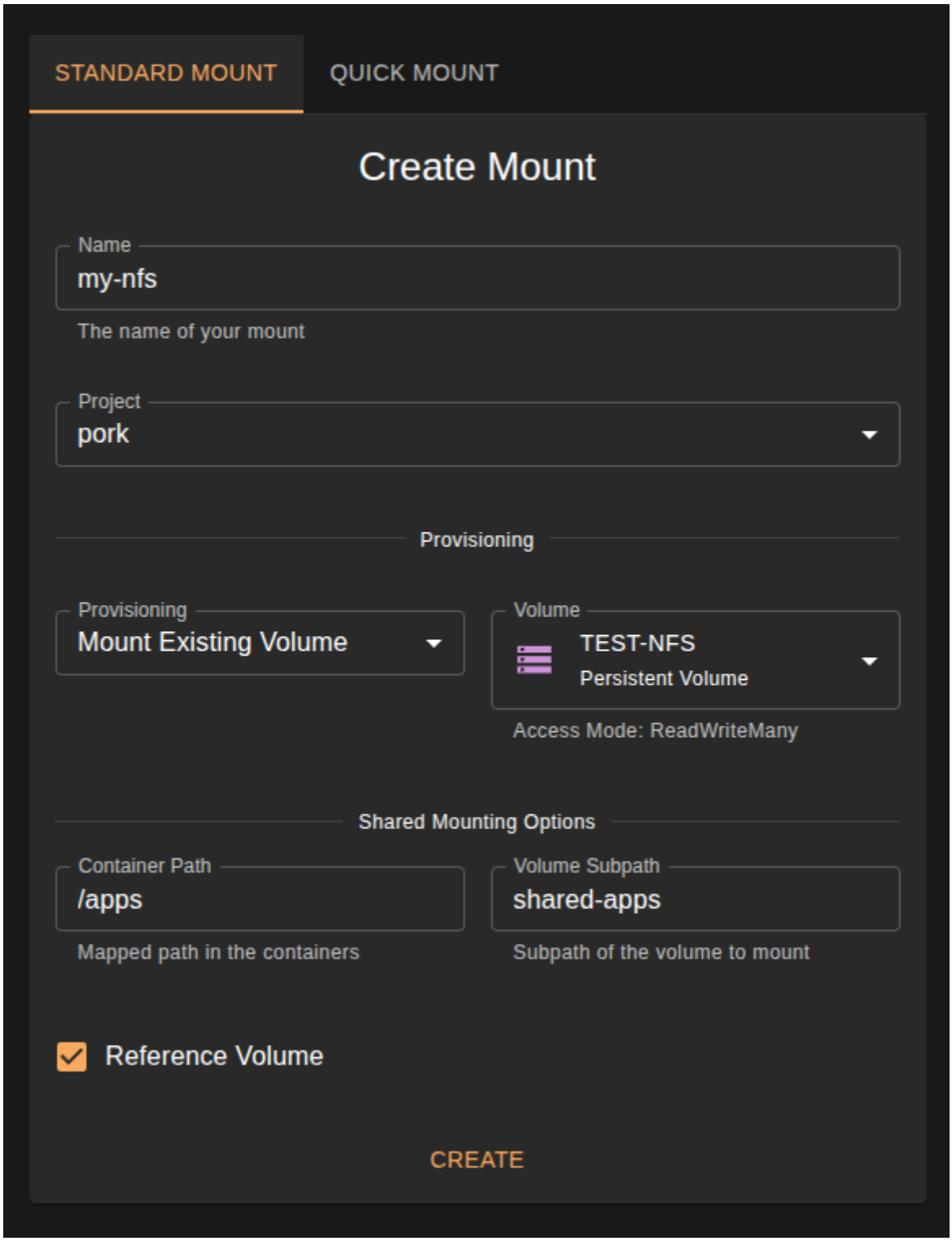
1. Create a Persistent Volume using NFS

test-nfs-pv.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: test-nfs
spec:
  capacity:
    storage: 10Ti
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: nfs
  nfs:
    path: /
    server: nfs-server.example.com
```

```
$ kubectl apply -f test-nfs-pv.yaml
```

2. Create a new mount using the below example values.



3. The mount will be displayed in the storage table with the REF in the name, it will already be bound. It will show that the path is shared as well as where the container path is mounted.

Projects	Name	Status	Capacity	Type	Claim Name	Mount Location
pork (1)						
	REF: test-nfs	Bound	10Ti	nfs	pork-my-nfs	shared-apps /apps

HOST PASS-THROUGH VOLUME

In this example, we have installed a number of tools on all servers that will be running our workloads. This package is installed on the host at `/opt/luna-tools`. These tools are geared toward AI and data science workloads, and we want to mount these tools to each workload in the project at the `/opt/luna-tools` folder inside the container. This will help us keep our workload containers very small and lightweight, while still allowing us to use the tools on the host machine.

This can also be used to pass through existing software or even VFX/GFX pipelines. Many companies already have existing software provisioning systems in place, such as Ansible, Puppet, or Chef. By using a HostPath pass-through volume, you can reuse these systems to install and prepare the host, then pass that on to the containers. This drastically speeds up provisioning times and provides a "hybrid" workload solution. This also provides a clear path for migration without having to fully ditch existing software provisioning systems.

The container mount path will resolve to the following.

```
(host)/opt/luna-tools -> /opt/luna-tools
```

Existing Volume	Subpath	Container Path
luna	/opt/luna-tools	/opt/luna-tools

You can equate this to more normal terms.

Server	Bind Mount	Container Path
host	/opt/luna-tools	/opt/luna-tools

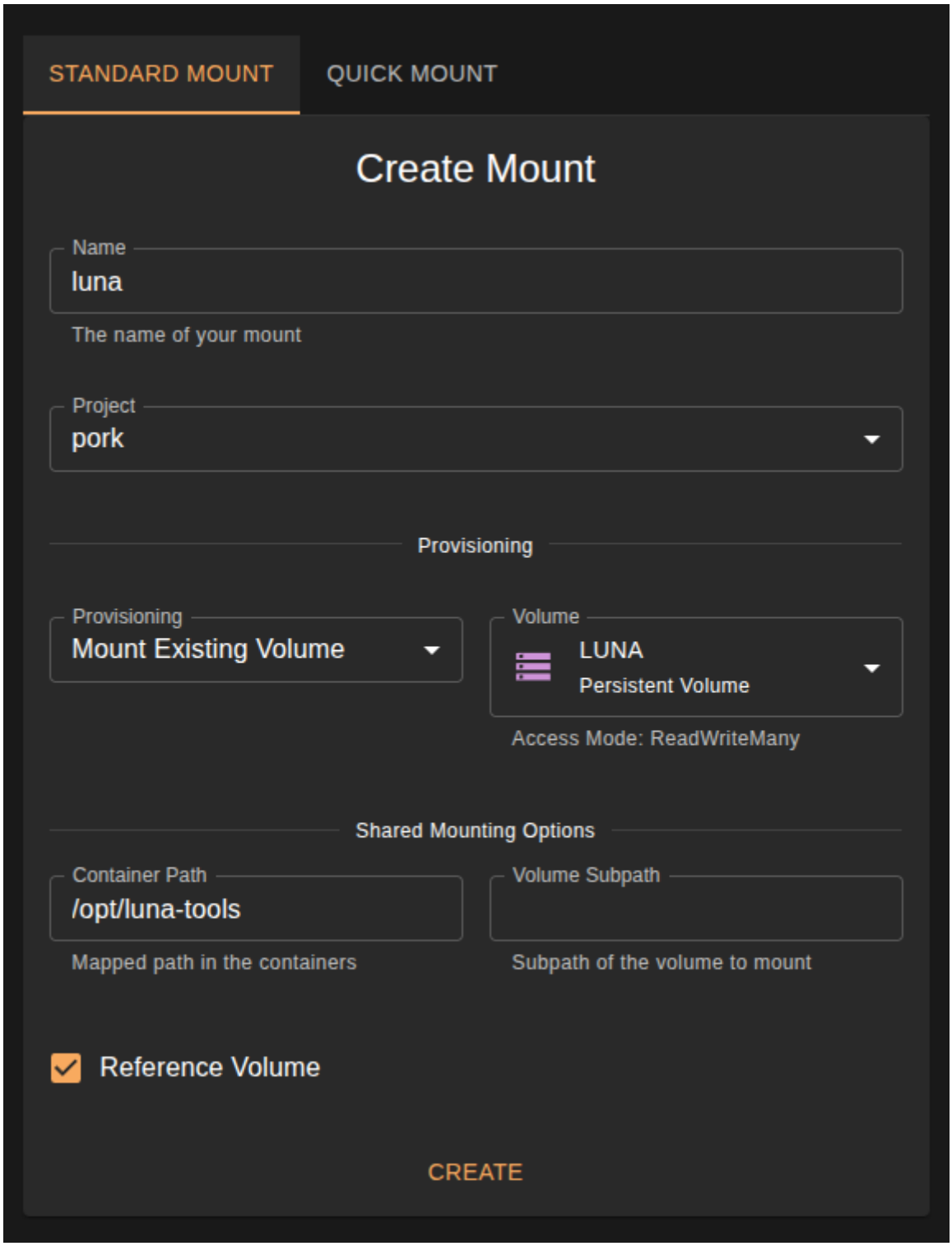
1. Create a Persistent Volume using NFS

luna-pv.yaml

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: luna
spec:
  capacity:
    storage: 10Ti
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
  hostPath:
    path: /opt/luna-tools
```

```
$ kubectl apply -f luna-pv.yaml
```

2. Create a new mount using the below example values.



3. The mount will be displayed in the storage table with the REF in the name, it will already be bound. It will show where the container path is mounted.

Projects	Name	Status	Capacity	Type	Claim Name	Mount Location
pork (2)						
	REF: luna	Bound	10Ti	standard	pork-luna	/opt/luna-tools

EXCLUSIVE DATABASE VOLUME

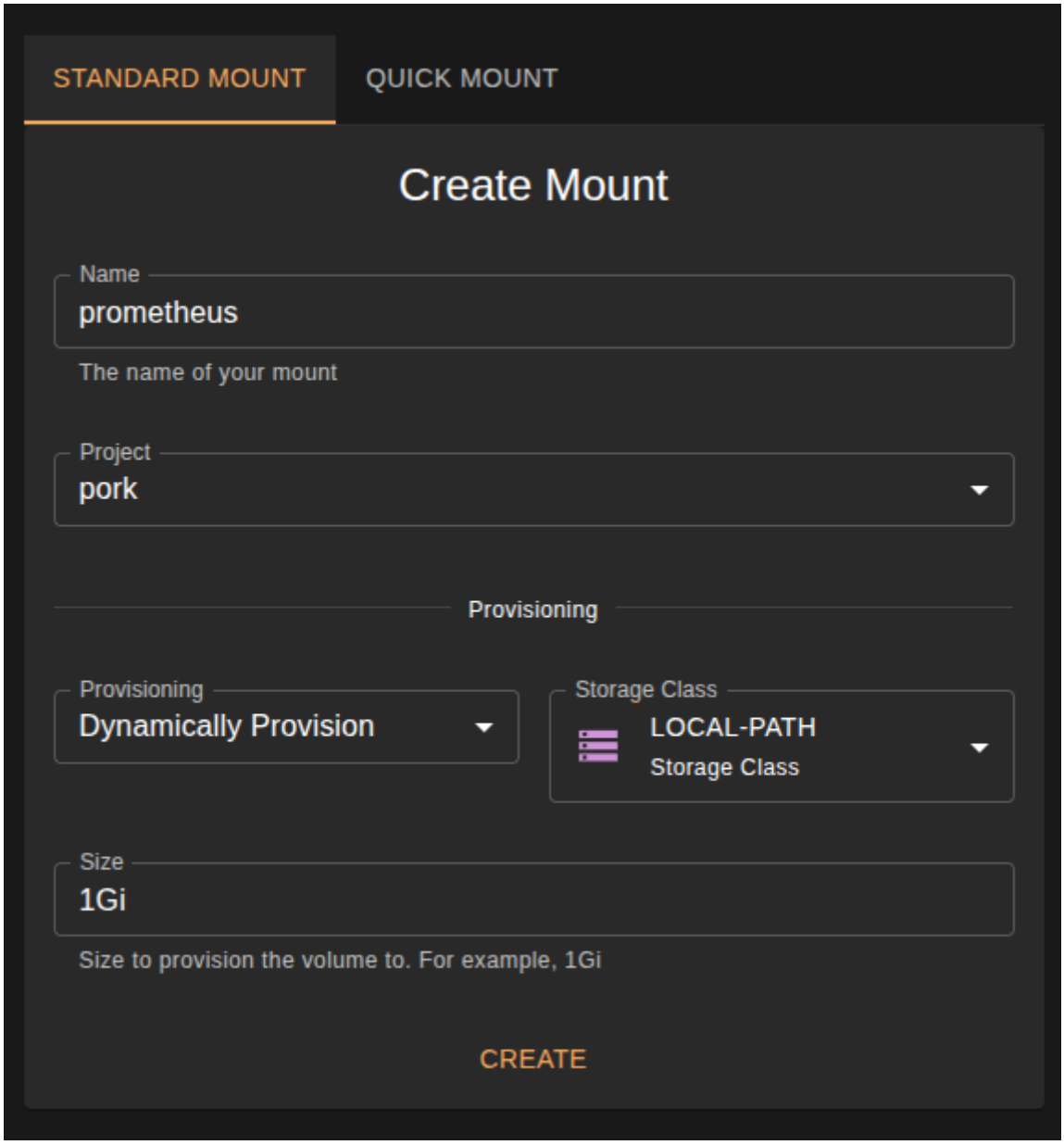
In this example, we want to create a database volume that is set to be exclusive. Meaning, it will be consumed by a single container somewhere downstream. This is mainly used with Terra Plugins. In our case, we will create a volume that will be used as a prometheus block device.

We are going to use the GP2 provisioner from AWS EKS, but you can use any provisioner that can dynamically provision block devices.

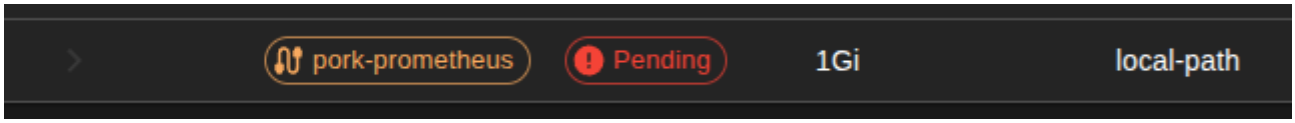
GP2

The GP2 storage class is a part of the EBS CSI driver that ships with EKS by default. Depending on your deployment environment, you will see different classes here. Please refer to your CSI drivers docs to see what is available.

1. Create a new mount using the below example values.



2. The mount will be displayed with an orange wire icon meaning it is a 1:1 relationship. It will also be in a "Pending" state until a container requests it. This is because it is an exclusive mount and will not be bound until a container requests it.



QUMULO CLOUD FABRIC (GLOBAL STORAGE)

In this example, we have deployed a Qumulo Cloud Data Fabric cluster across multiple locations in both the cloud and on-prem. In all locations, we have deployed Juno and intend to have a single project that spans all locations with a unified storage and high

performance workloads that scale. This grants us the ability to have full location failover and disaster recovery as well as have access to talent across the globe.

ATA Media

Juno Innovations sister company, ATA Media, uses this exact setup to do global VFX for high-end commercial clients while accessing talent across the globe. This allows them to have a single project that spans multiple locations and have a unified storage solution that is fast and reliable.

Because Qumulo can be mounted via NFS, we can use the same NFS mount across all locations.

The container mount path will resolve to the following.

```
qumulo-nfs.example.com/shared-apps/ -> /apps/
```

Existing Volume	Subpath	Container Path
qumulo-nfs	/shared-apps	/apps

You can equate this to more normal terms.

Server	Network Share	Container Path
qumulo-nfs.example.com	/shared-apps	/apps

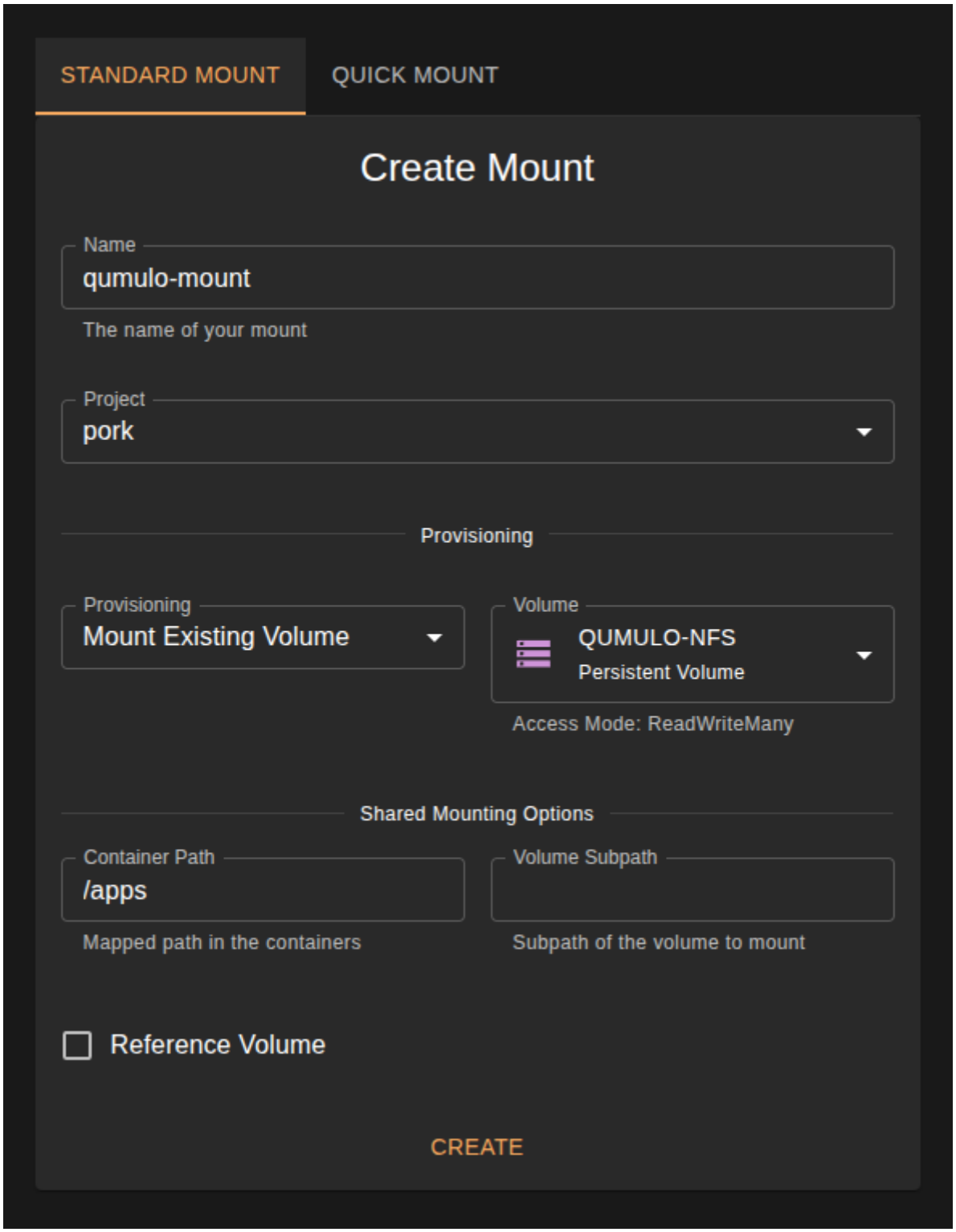
1. Create a Persistent Volume using NFS

qumulo-nfs-pv.yaml

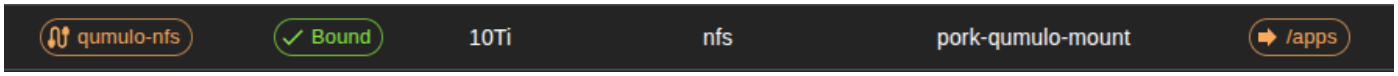
```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: qumulo-nfs
spec:
  capacity:
    storage: 10Ti
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: nfs
  nfs:
    path: /
    server: qumulo-nfs.example.com
```

```
$ kubectl apply -f qumulo-nfs-pv.yaml
```

2. Create a new mount using the below example values.



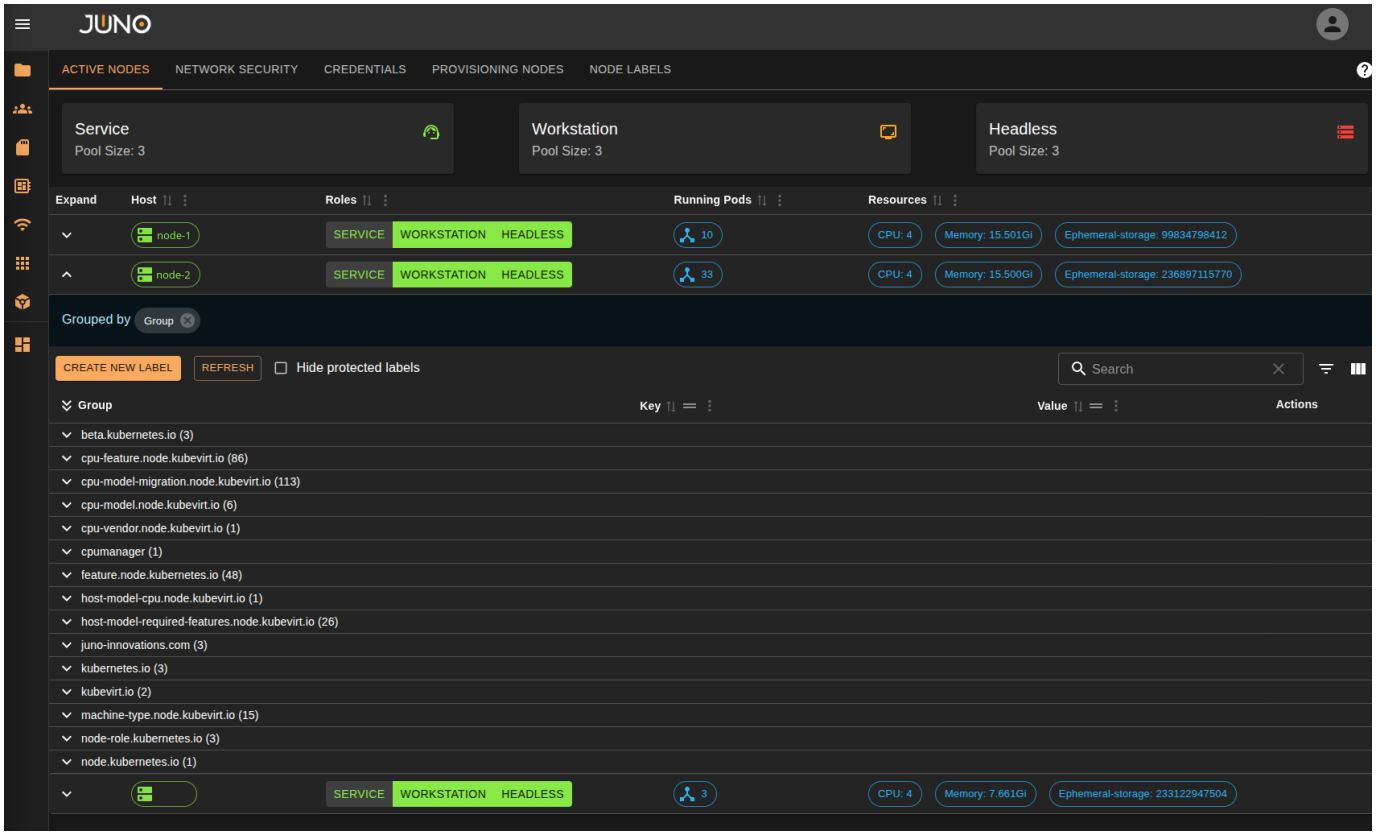
3. The mount will be displayed in the storage table with the REF in the name, it will already be bound. It will show that the path is shared as well as where the container path is mounted.



4.1.6 Networking

> **Note**
>
> Network Management is restricted to members of the 'admin' user group. If your cluster is running k3s, you will have the ability to provision new nodes directly through Genesis.

The Network page allows users to label nodes, manage & create per-project network policies, see some of the allocatable resources on a node, and provision new nodes if your cluster is running k3s.



Label Nodes

Your current nodes in your cluster will be displayed on the Active Nodes tab. We provide a quick access feature to label a node for the Juno specific roles by simply clicking the options in the row column. We also provide the ability to see and edit all node labels as well as create new labels via our node labels table. You can access the node labels table either by clicking the tab at the top of the networking page, or within each individual nodes details panel drop down. If accessing via a nodes details panel, the labels will be filtered specifically for the node selected.

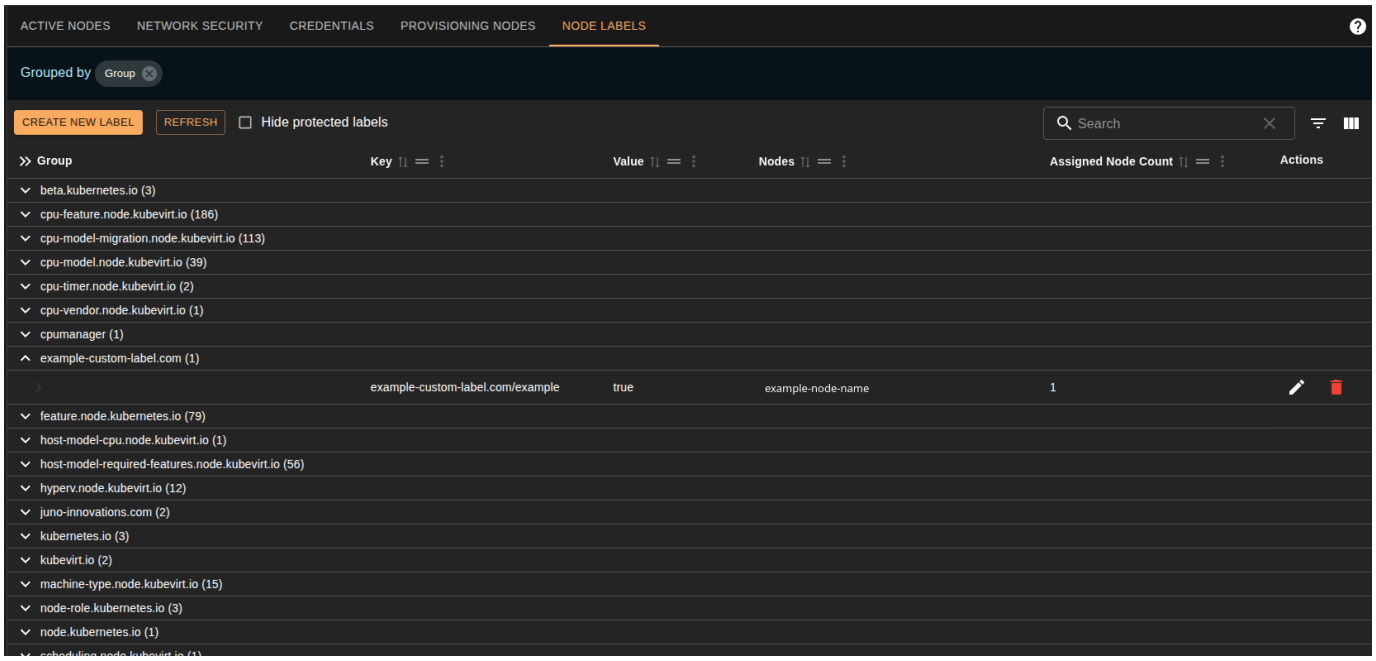
SERVER ROLES

Label	Description
Service	The cluster's main control node. At least 1 node must be labeled as a Service node. These nodes run core Orion services and APIs.
Workstation	Nodes that will be used to run Kuiper workloads, such as Helios containers. This labels usage is at the descrection of the Terra plugin author.
Headless	Nodes that will be used to run intensive batch processing workloads. This labels usage is at the descrection of the Terra plugin author.

You learn more about our server roles and requirements here

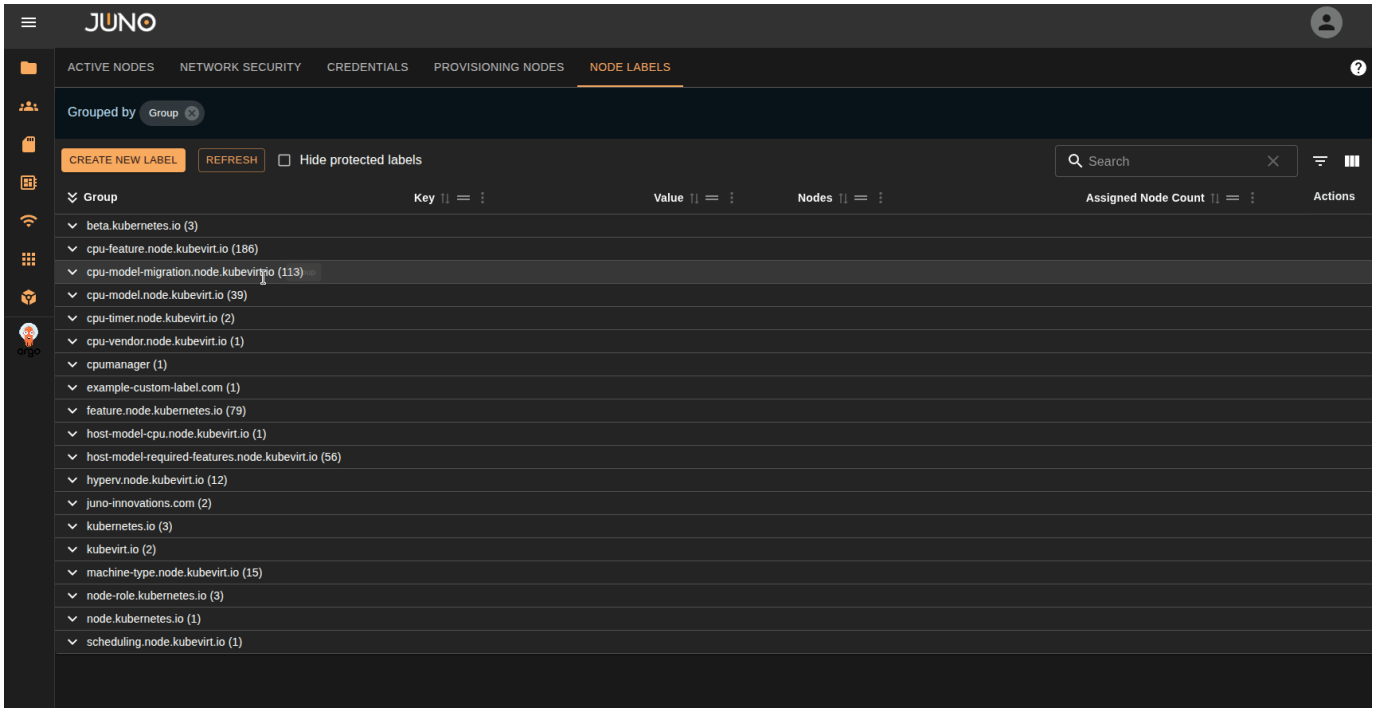
NODE LABELS TABLE

From the Node labels table, you can see all labels currently assigned to nodes in your cluster. Edit these labels, and create new labels. Each label is grouped by its key. You can expand each key to show the label values, nodes assigned, and it's available actions.



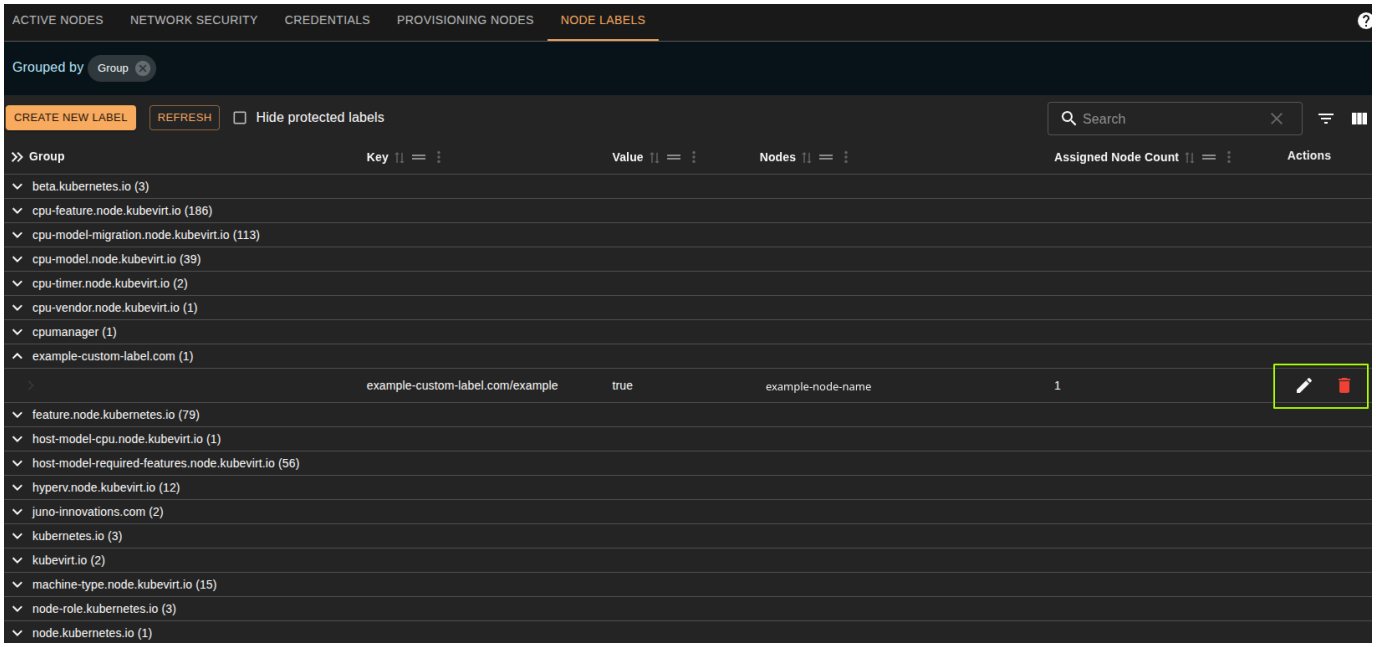
Create New Label

To create a new label, simply click the "CREATE NEW LABEL" button on the top left of the table. Fill out the form, and click create.



Actions

Each label has two available actions, edit, and delete. Editing allows you to easily update a labels node assignment as well as value. Please note some labels are protected. These are typically required for Orion to run properly and are not allowed to be edited via the UI.

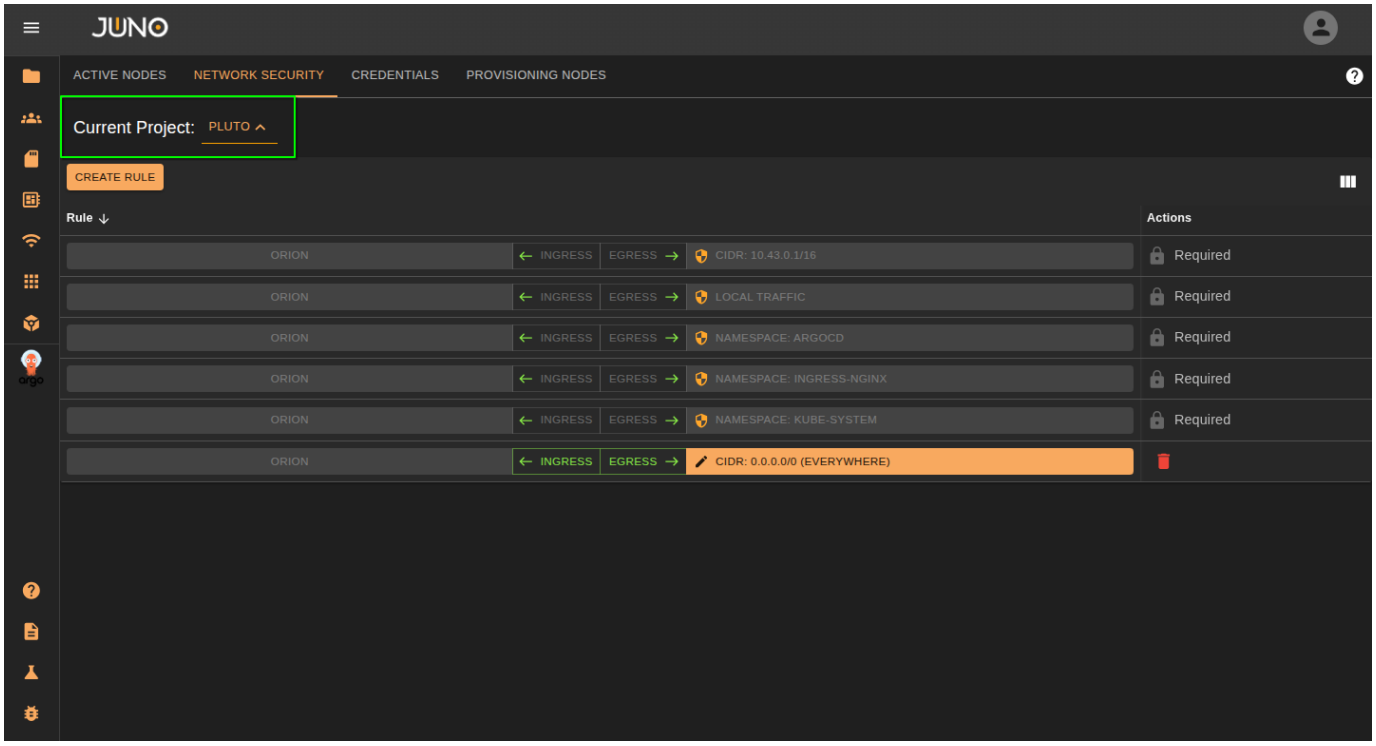


Network Security

Here you can set per-project network policies.

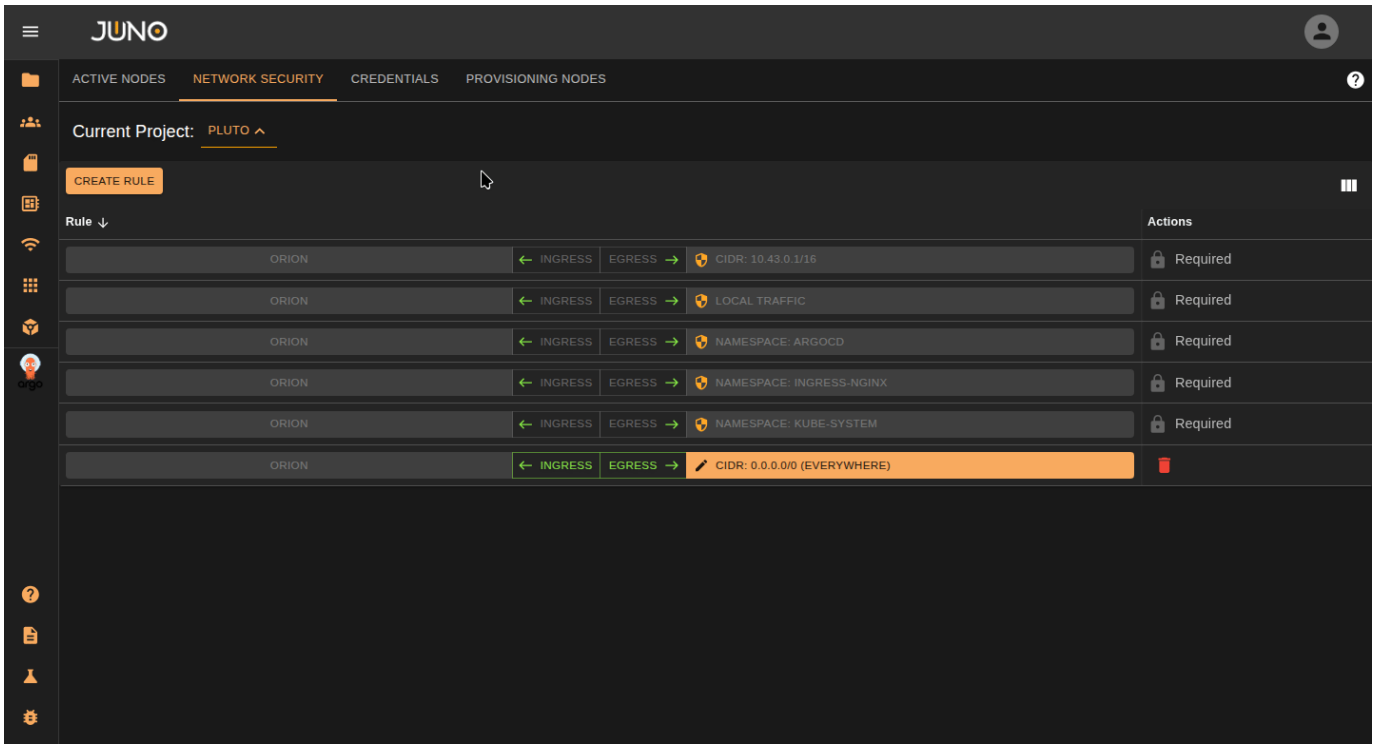
SELECT PROJECT

Use the project dropdown selector located on the top right of the network policies table.



CREATE/DELETE RULE

Click the CREATE RULE button. A new rule will be appended to the table for the project. You can click the delete icon located on the right of the row to delete the rule.



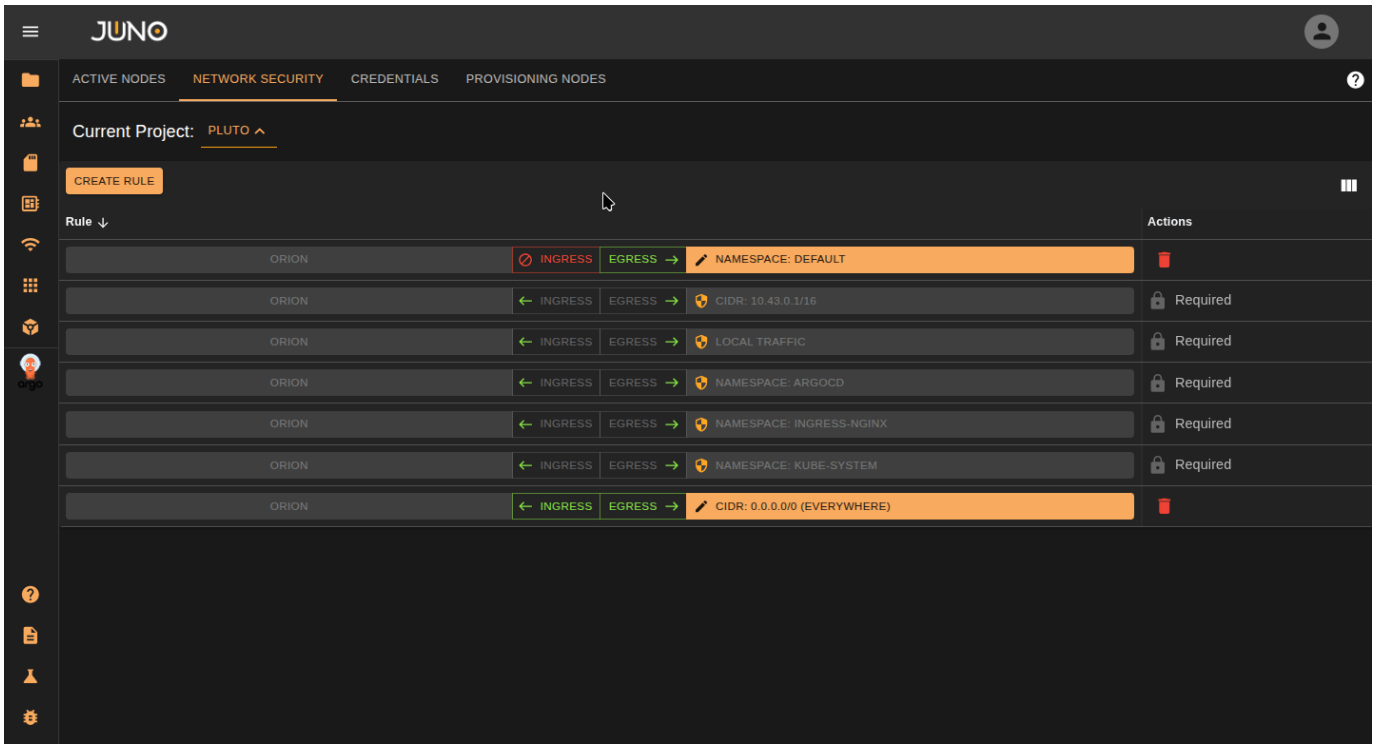
EDIT RULE

Ingress/Egress

Select the Ingress or Egress button on the row to control the traffic on your rule. One of the options should be open for traffic. If you want to close both ingress and egress, Just delete the rule.

Ingress Open - All traffic from the targeted namespace/range can reach us

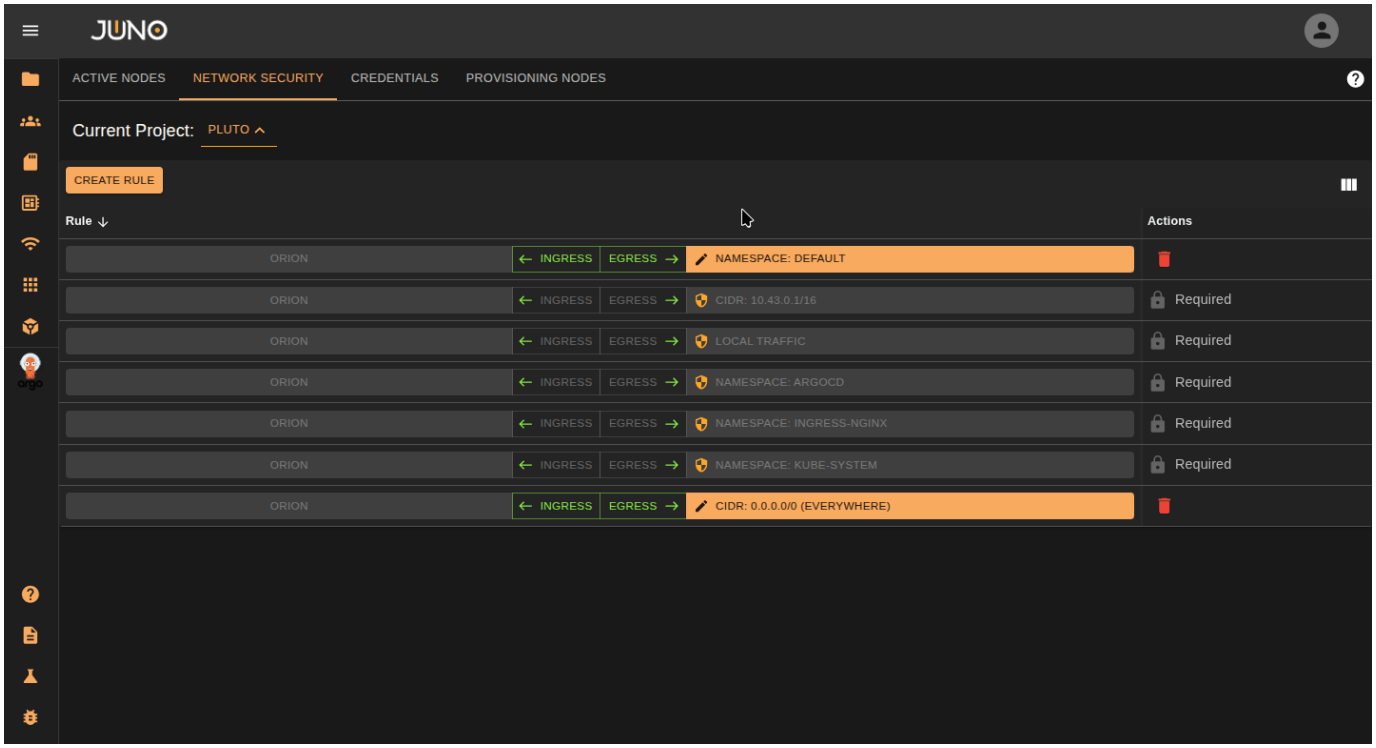
Egress Open - Pods from our namespace are allowed to connect to the target



Edit Rule Properties

Click on the orange bar on the row to edit the rule properties. You can change the rule type and values within the popup.

Rule Type	Description
Namespace	The namespace you want to apply the network policy to.
CIDR	Network range you want the network policy to apply to.
Grant External Access (Internet)	Lets the project have internet access.



Provision New Nodes (k3s clusters)

For k3s-based clusters, we offer you the ability to expand your cluster using our playbooks. We assume that the cluster settings match our QuickStart deployment, meaning that:

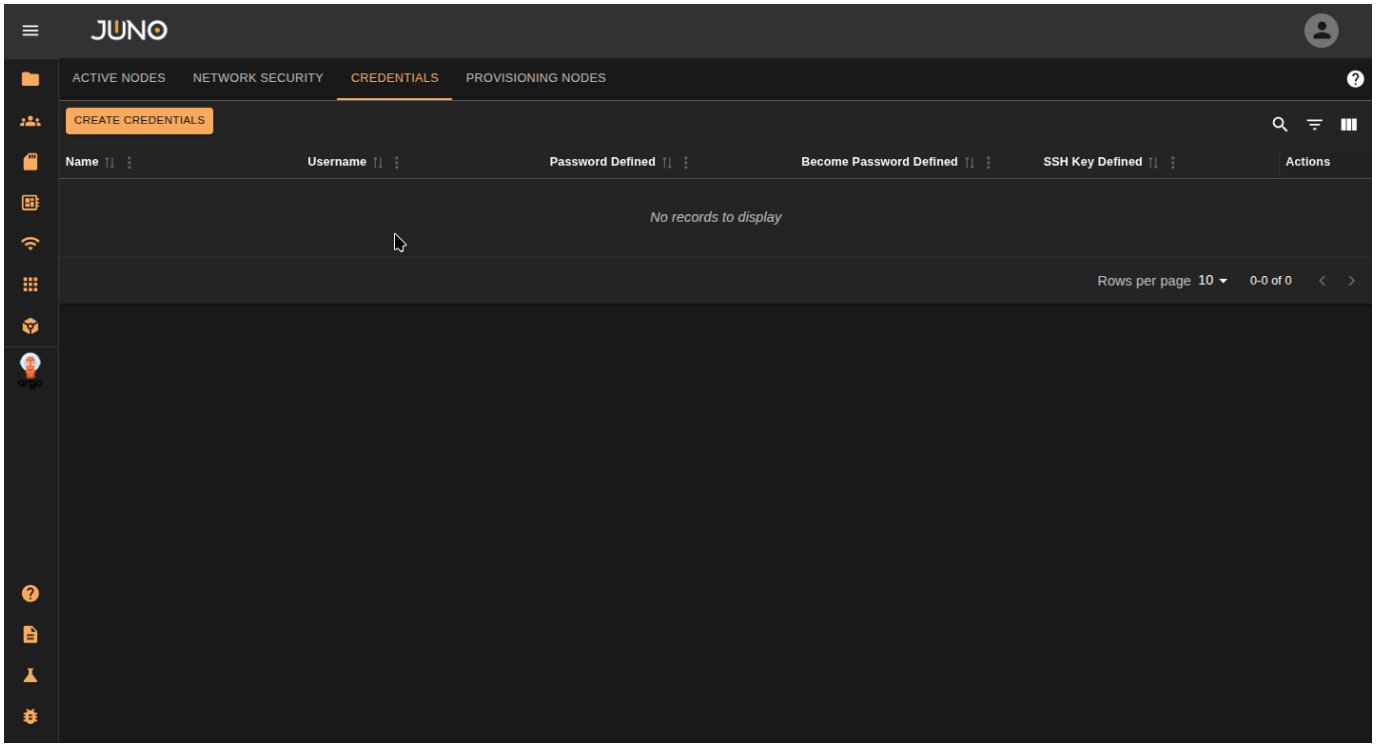
- Flannel is the CNI and the wireguard backend is enabled
- by default, k3s uses the less secure unencrypted backend. Our platform currently forces controlplane nodes are set up with encryption enabled

If your cluster is running k3s, Genesis allows you to provision a new node within the UI. You can do this by adding new credentials for the node in the credentials tab, then adding the new node using those credentials in the provision node tab.

ADDING CREDENTIALS

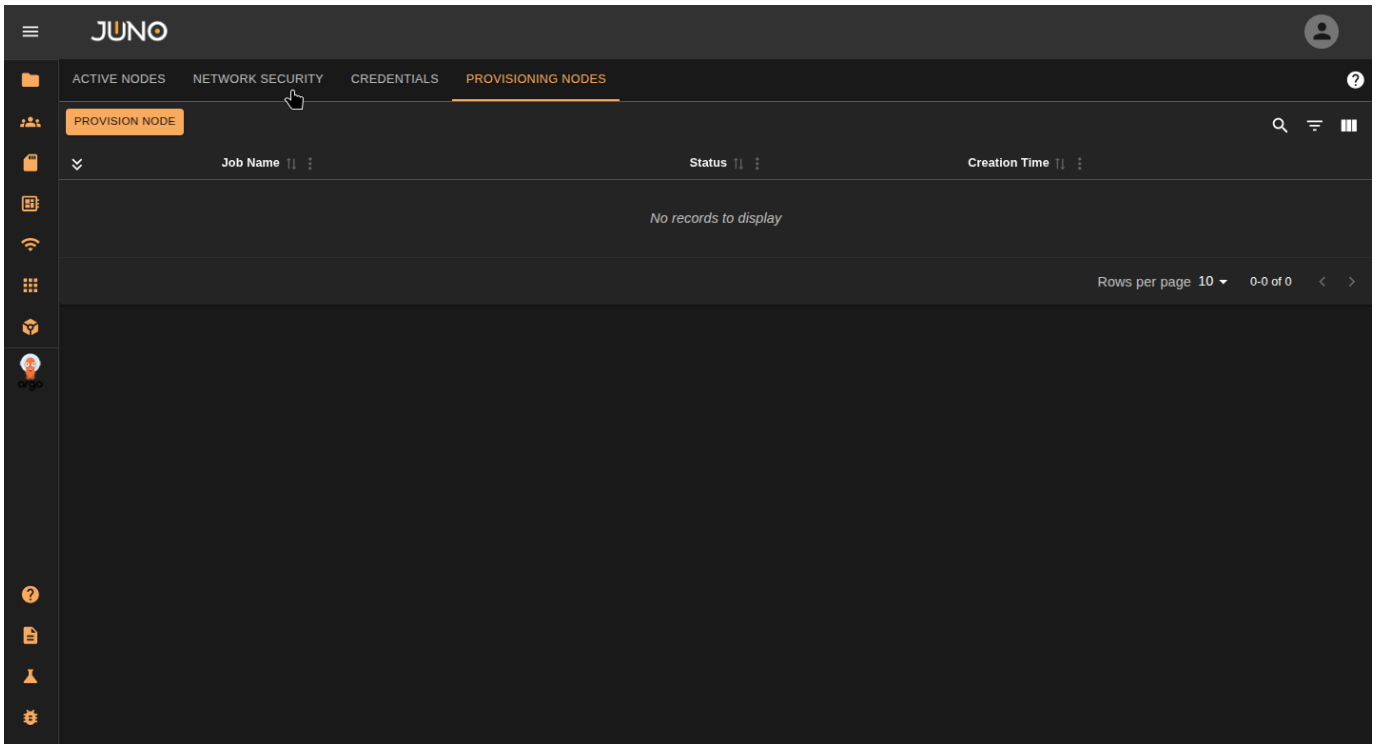
Before provisioning a new node, you will need to add your nodes credentials to the credentials table. You can do this by navigating to the credentials tab and clicking the create credentials button. You can authenticate to a node with a user and use their password or ssh key under the Become Password field.

> **Warning**
>
> If your node does not have passwordless sudo, you will need to supply the user's 'become password'. This is the password Ansible will use to escalate privileges via sudo. This does not apply if you connect as root.



PROVISION NODE

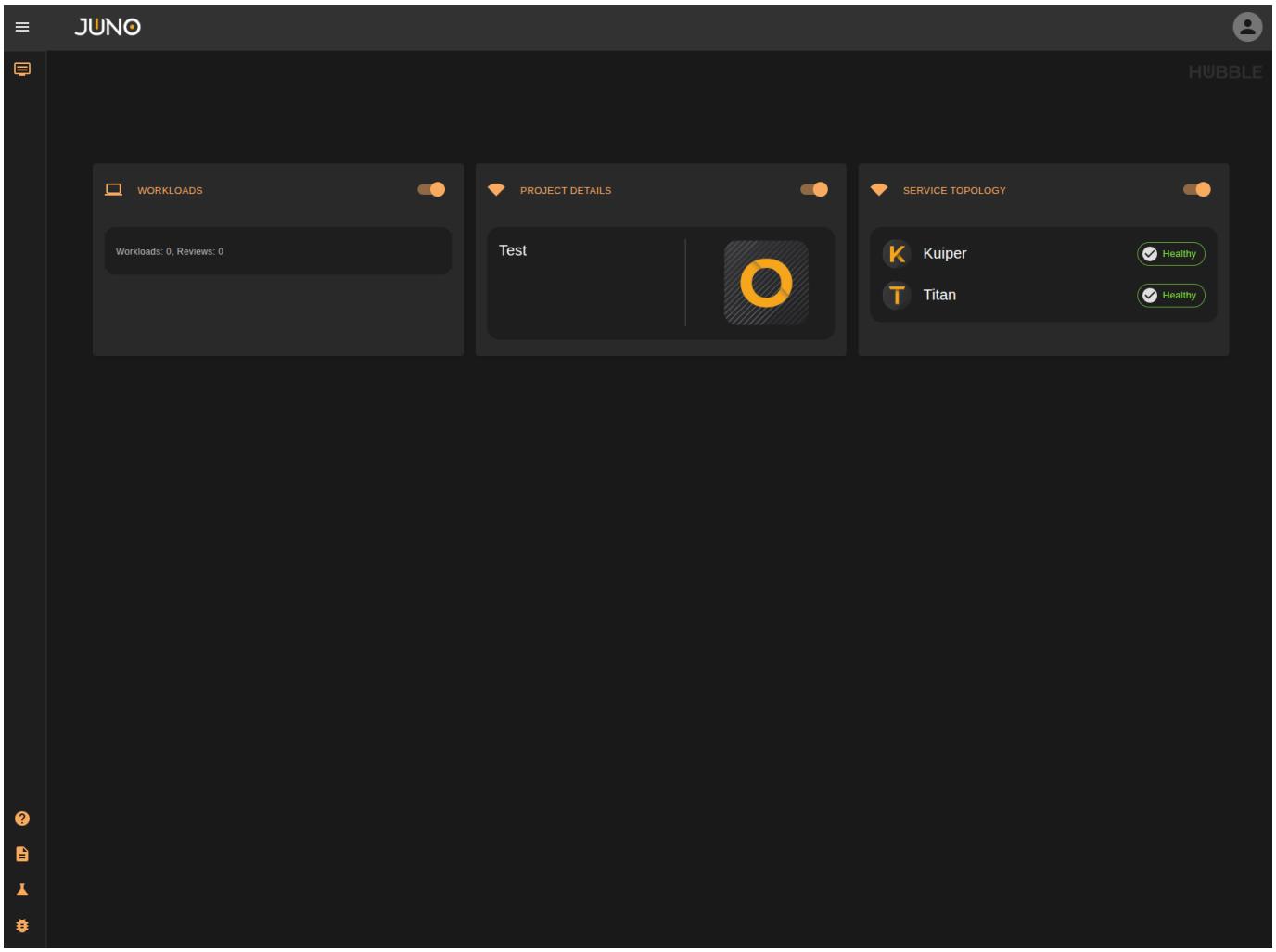
Once you added your node's credentials, navigate to the Provision Nodes tab and click the provision node button. This will open a popup where you can enter the node's information. Make sure to select the credentials for that node and what role you want to be applied to the node. You can later adjust this role in the active nodes tab. Once you submit the form, this will start the ansible job. Expand the row's detail panel to view the logs of the job. Once the node is fully provisioned, the job status will switch to success.



4.2 Hubble

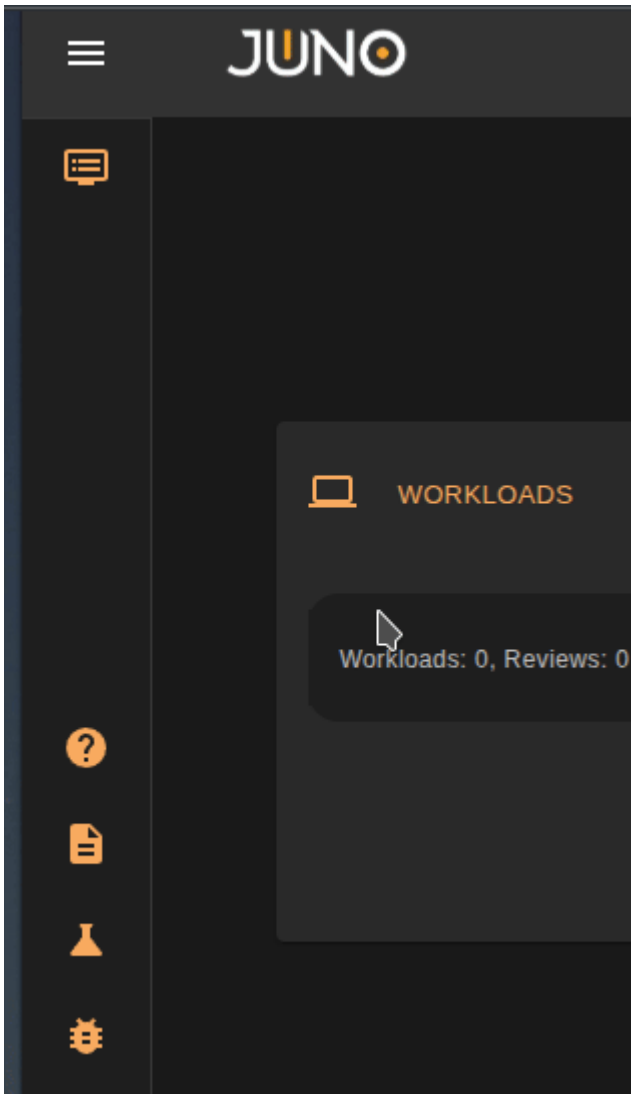
4.2.1 Introduction

Hubble is a user's primary access point for an Orion project. From here users can use the active Orion services. Such as Kuiper to launch and connect to workloads.



Navigation

On the left side of the screen, users will find a sidebar. To expand it, click the icon in the top-left corner of the page. To collapse the sidebar, click the arrow at the top. The sidebar contains links to active services and help resources. You can also click the Juno logo to return to the Hubble homepage.



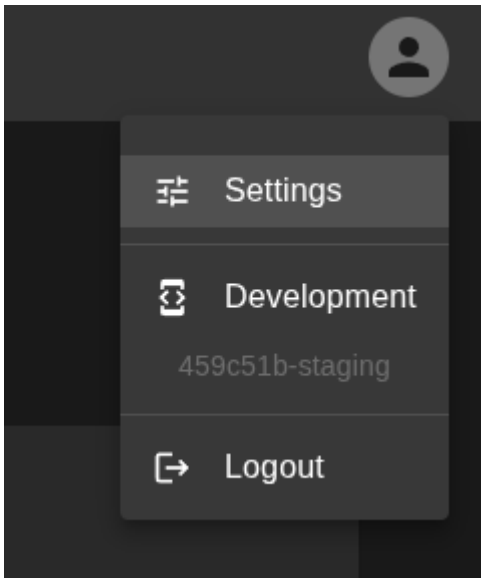
HELP RESOURCES

These links are located at the bottom of the sidebar.

Link Name	Definition
Help	Starts a help stepper that will guide you on how to use the main functionalities of the Hubble UI.
Documentation	Links you to our Official Orion Documentation.
Support	Links you to our support request form
Sales	Links you to our sales request form.

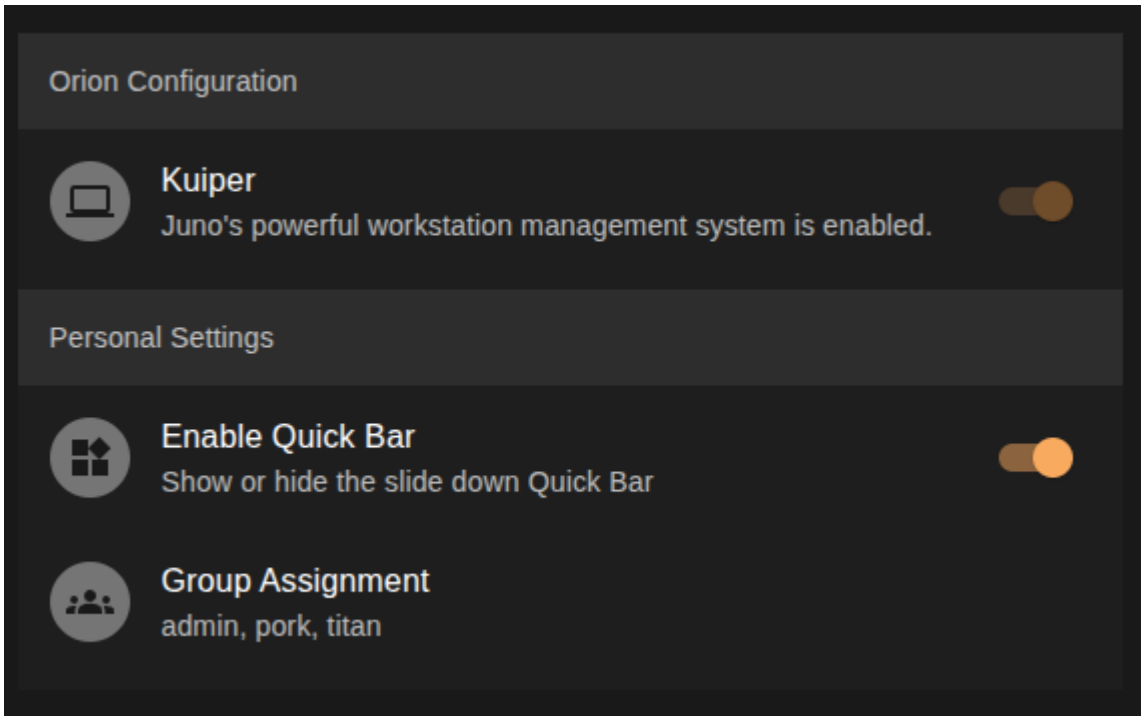
User Menu

On the top right of the UI, You can open the user menu by clicking the user icon. The menu gives you the ability to navigate to the settings page, development page, and logout.



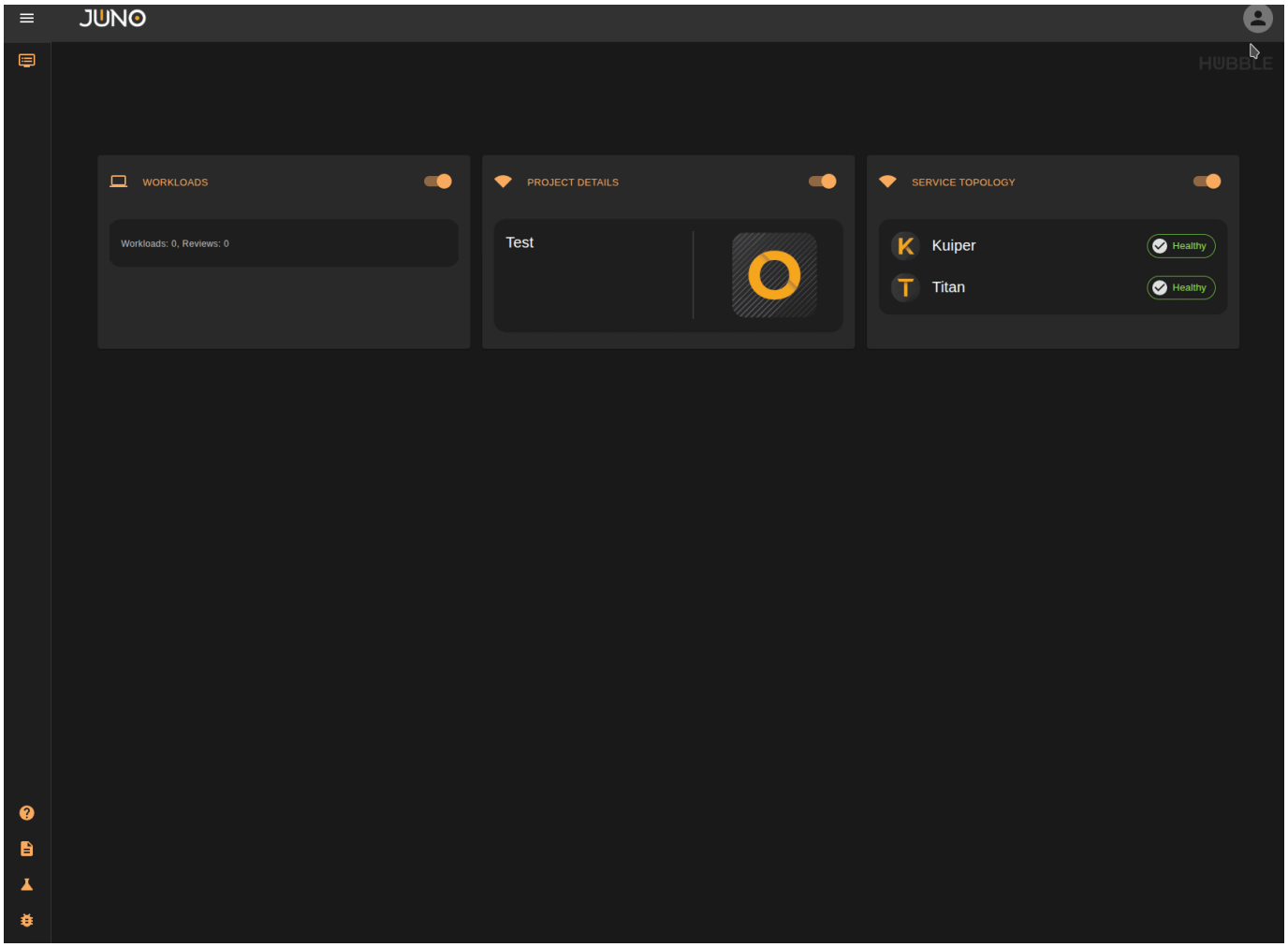
SETTINGS PAGE

The settings page shows which services are currently enabled, lets you enable or disable the quickbar, and displays the groups to which you are assigned.



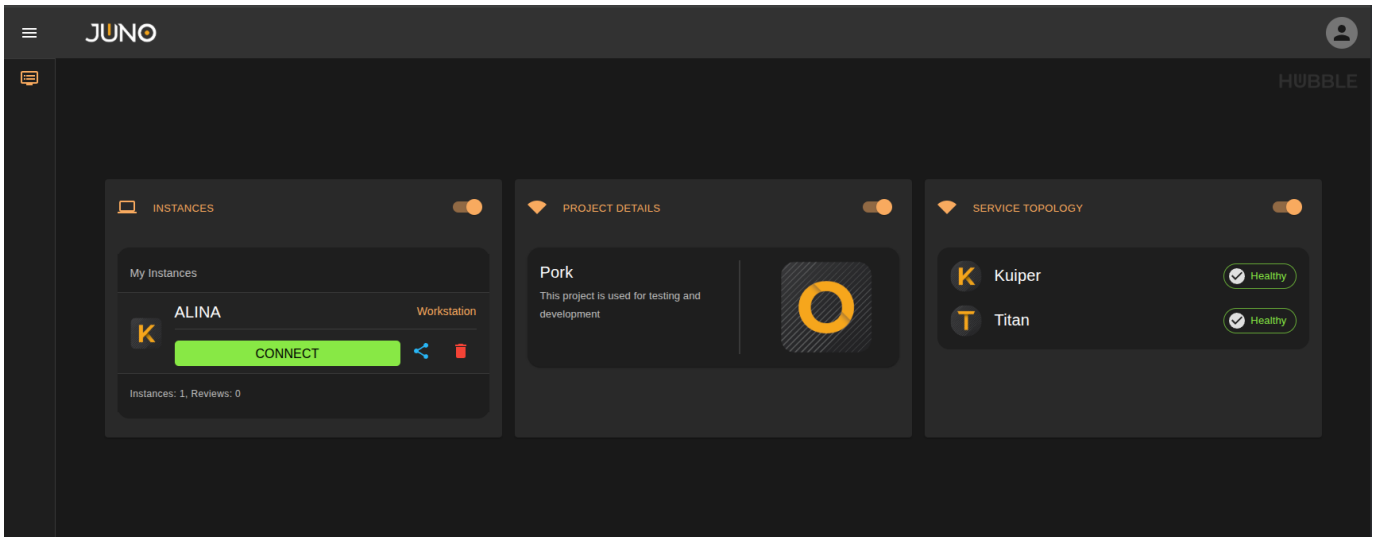
DEVELOPMENT PAGE

We use FastAPI under the hood for our services. You can navigate to our development page to directly use the FastAPI Swagger UI to hit our services. Each service is available specifically to admin users or users assigned to the respective group. This can be used for testing or development purposes.



Dashboard Widgets

The homepage of hubble will have widgets displayed for the services currently active. These are used for quick access/viewing.



WORKLOADS

This widget lets you quickly connect to any running workload and delete workloads directly from the widget. It also lets you share and connect to other workloads that are being shared with you. This is available in the quickbar as well.

PROJECT DETAILS

This widget displays the project's name, description, and associated icon.

SERVICE TOPOLOGY

This widget displays the health status of services. Services that are running are shown as healthy; services that are down are shown as unhealthy. This is available in the quickbar as well.

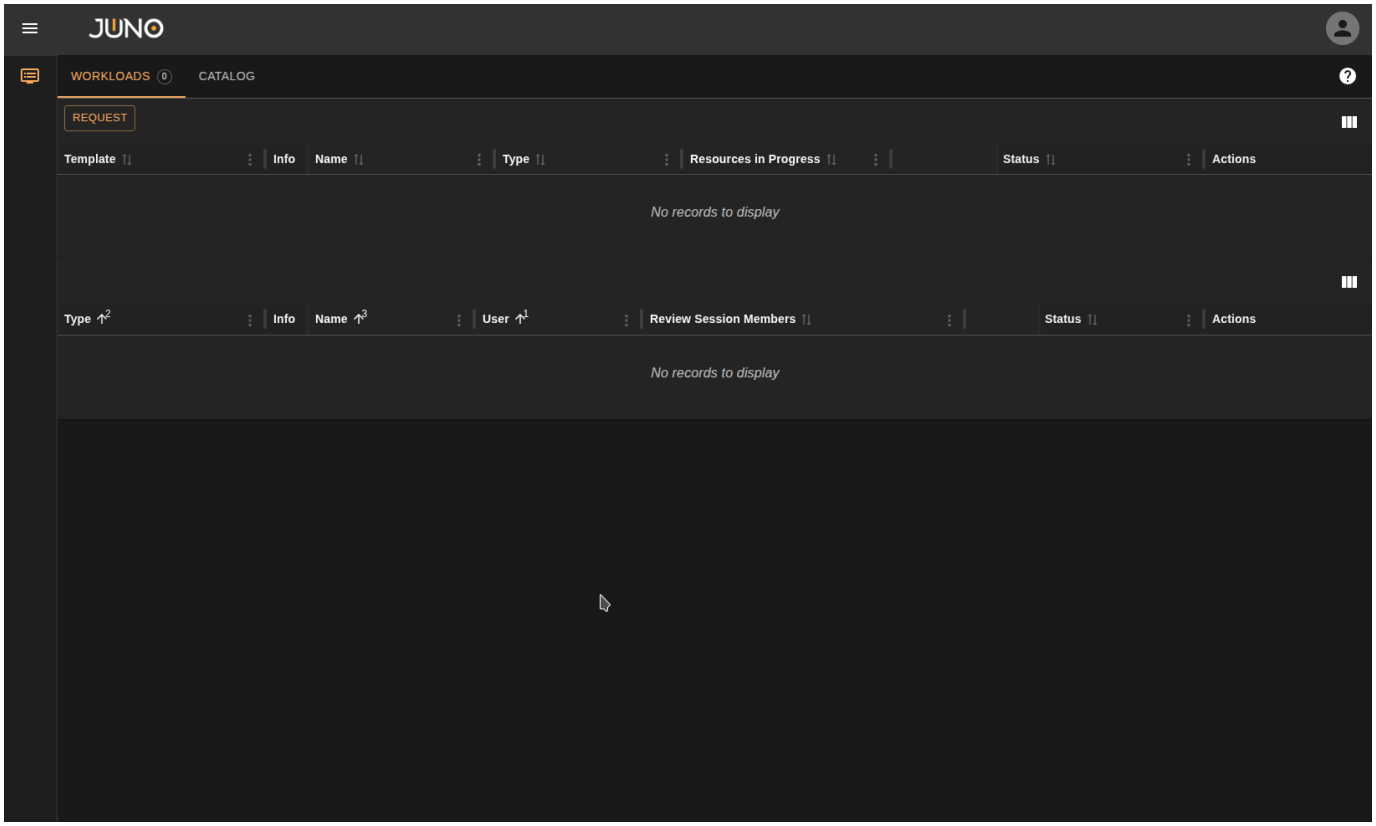
4.2.2 Workloads

One of the core Juno products is our Kuiper workloads. This is where users can launch workloads and connect, manage, and share them.

Workload Management

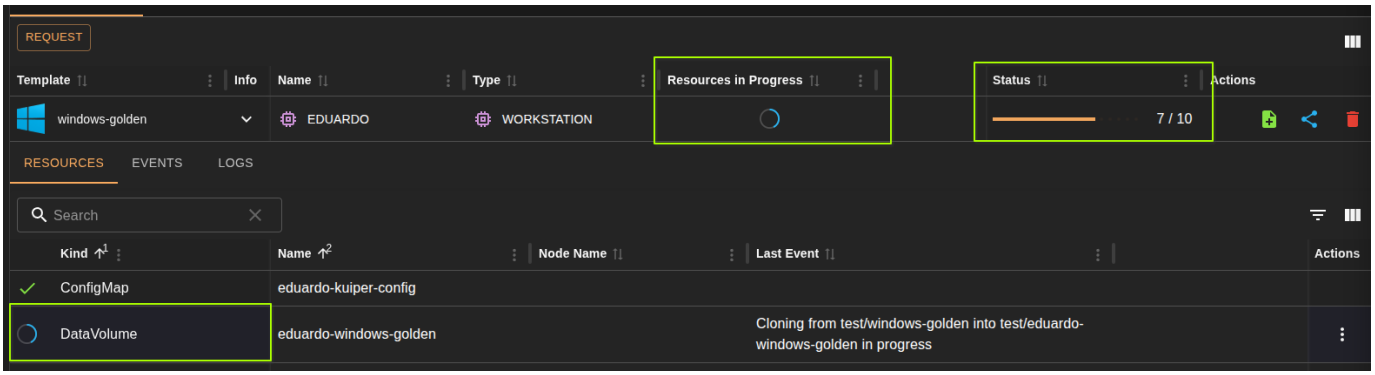
REQUEST

Users can quickly request a workload by clicking the "Request" button in the top-left corner or by opening the Catalog tab. The catalog lets you filter available templates and request a workload. You can also add new workload templates from within Genesis. Read more about workload template creation here



Workload Status:

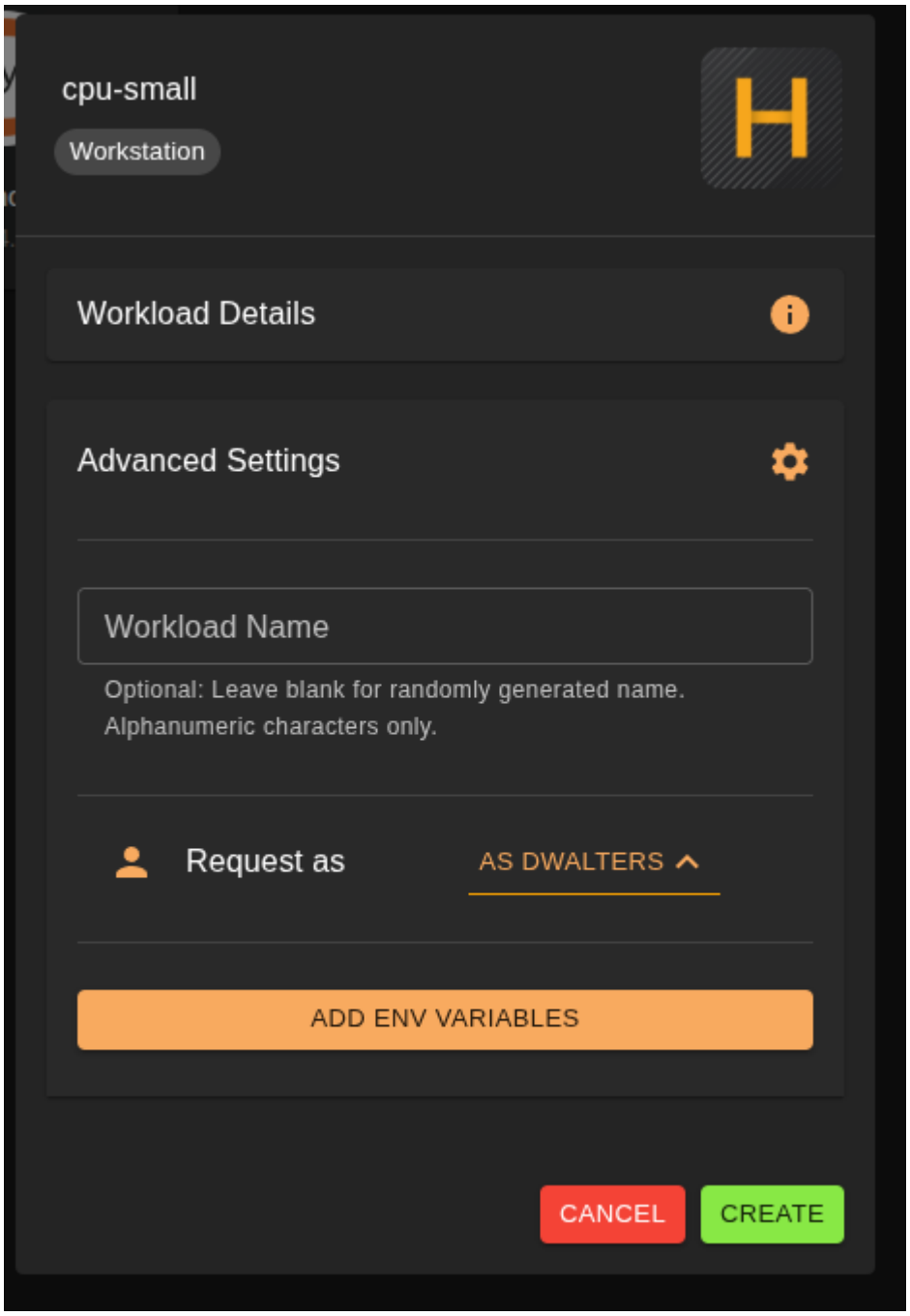
Once requested you will see a status bar showing the number of resources that are ready, out of how many are left. You can also expand the workloads details panel, to see the resources table. Here you will be able to see each resource and their current state. Once all resources are ready, a connect button will appear. Some resource actions will also provide a progress wheel showing the progress percentage for that resources current action. If you hover over this progress wheel you will see the current percentage and the resource name.



Advanced Settings

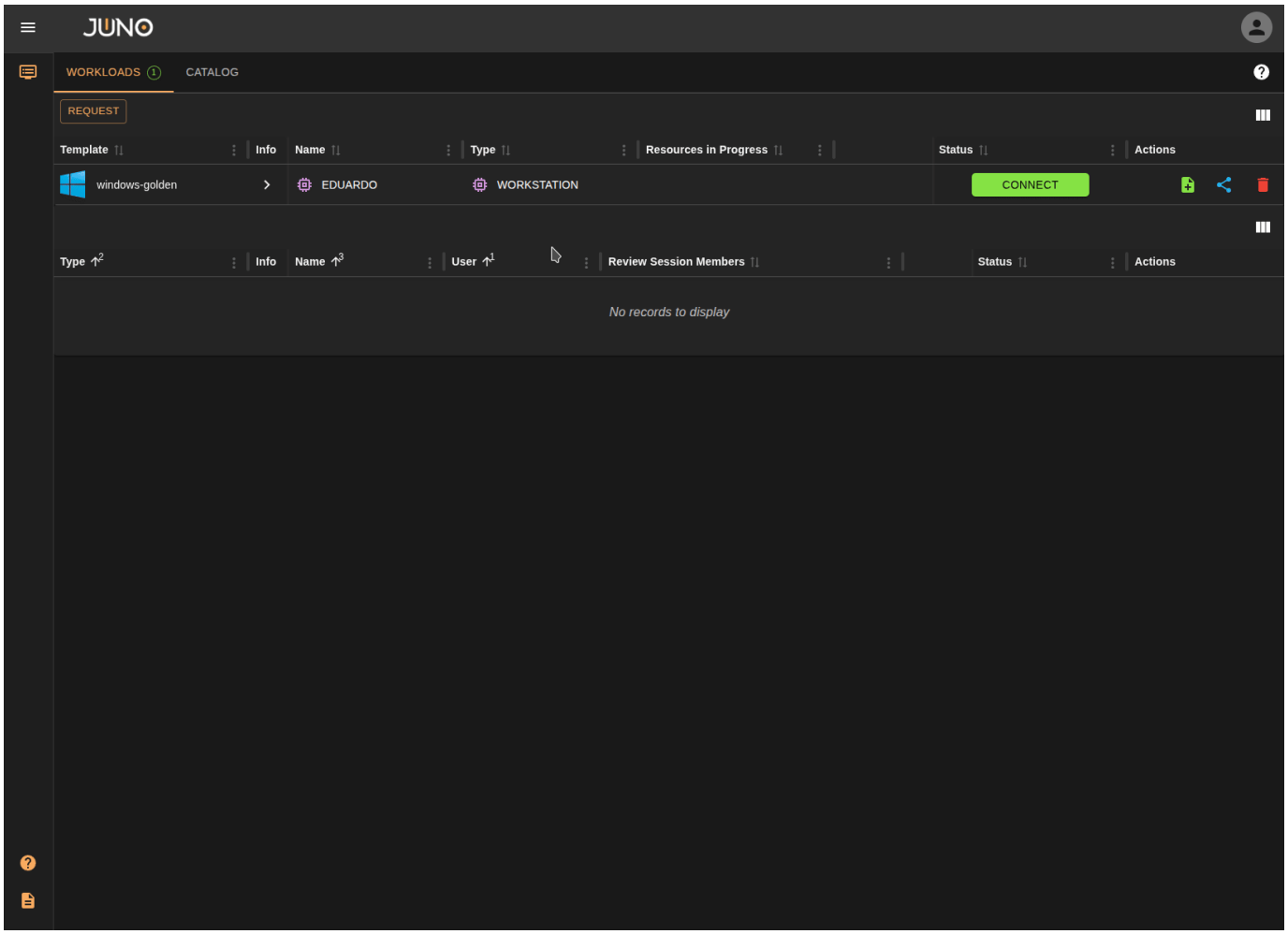
When requesting a workload, users can configure advanced settings such as a custom workload name, requesting the workload on behalf of another user, and custom environment variables.

> **Note**
>
> Requesting as a different user is restricted to members of the 'admin' and 'kuiper' user group. Read more about user management here



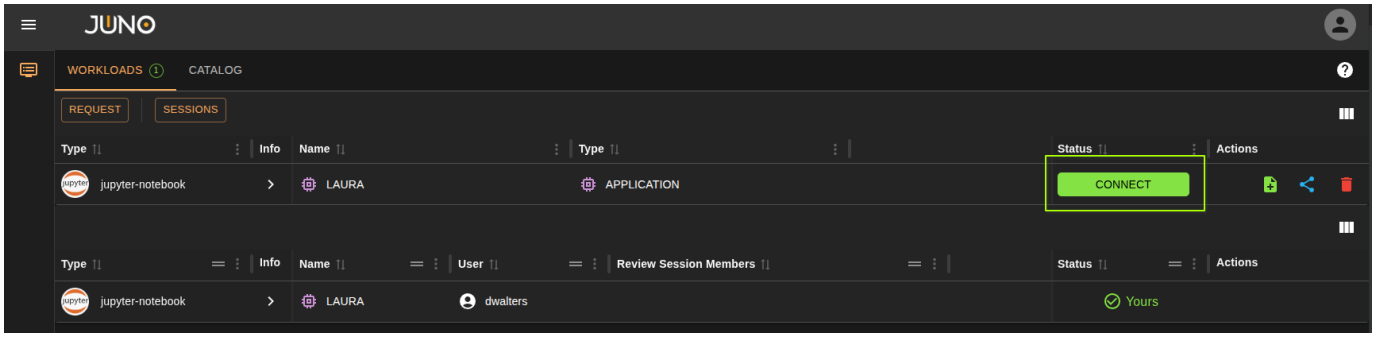
RESOURCES, EVENTS, AND LOGS

You can expand the workload row details panel to view the resources, events and logs of the running workload. Any resource that has available actions, will have these listed on the right in the "Actions" column. Simply click the 3 dots to show the resources available actions. To learn more about some of the available resource actions. See our resource actions documentation



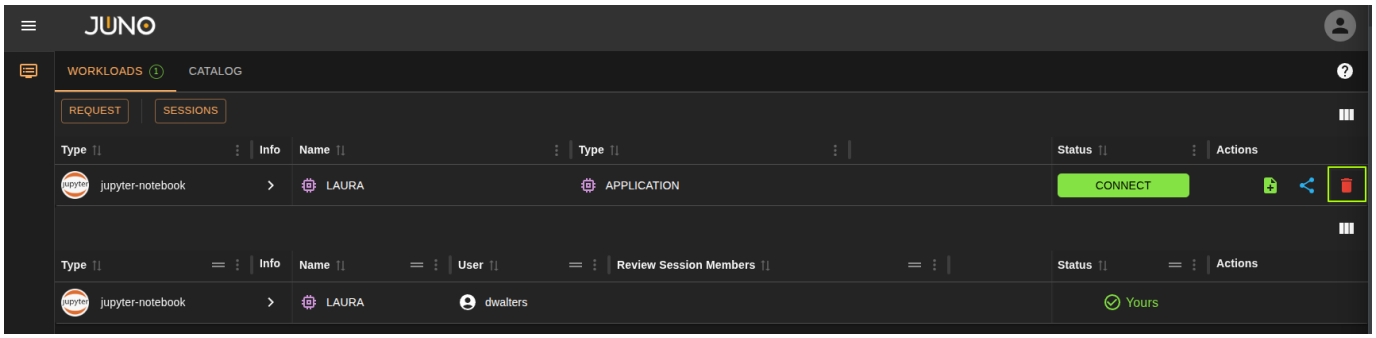
CONNECT

Once the workload is ready to be connected too. A green CONNECT button will appear in the status. Click this button to connect.



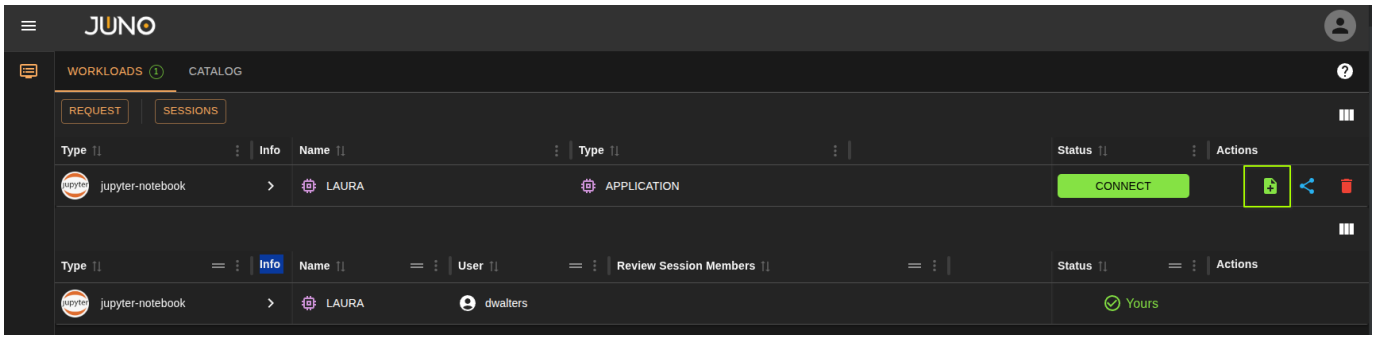
DELETE

Click the delete icon in the actions column on the workload row to delete the workload.



CREATE TEMPLATE FROM WORKLOAD

Click the green icon in the Actions column for a workload to create a template from that running workload. This is useful when you create a workload with custom environment variables at request time and want to save them for later; the created template will include those environment variables.

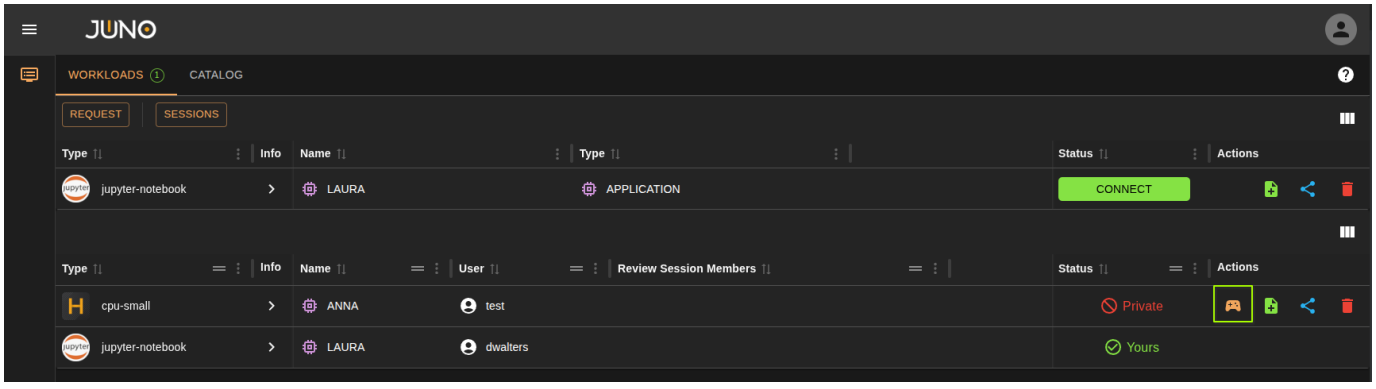


REVIEW SESSIONS (COMING SOON)

This functionality is currently being revised. Coming soon...

ADMIN CONTROL WORKLOADS

Admins have the ability to control another users workload. This works great for debugging, and training purposes.



4.2.3 Resource Actions

VM Actions Explained

Once you have a VM workload running, you'll find several actions available in the Resources table under the VirtualMachine resource. Here's what each one does:

Action	What It Does	When to Use It
Stop	Immediately terminates the VM and releases compute resources back to the cluster. This is equivalent to yanking the power cord.	When you're done with the VM and want to free up resources. Unsaved work will be lost.
Restart	Fully deletes the existing VM instance and recreates it from scratch. This is a hard reset—yank the plug and plug it back in.	When the VM is unresponsive or you need a clean reboot from the hypervisor level.
Soft Reboot	Sends a reboot signal to the guest OS (Windows). This is the same as clicking "Restart" from within Windows.	For routine reboots — after installing updates, applications, or when Windows asks for a restart.
Pause	Freezes the VM in its current state. The VM remains in memory but stops executing.	Special use cases only — such as temporarily freeing CPU cycles or troubleshooting. Resume when ready.
Migrate	An offline migration moves a virtual machine from one node to another by shutting it down on the source node and restarting it on the target node without maintaining uptime	when you need to move a virtual machine between nodes for maintenance, resource optimization, or troubleshooting and can tolerate downtime.

Tip

For day-to-day reboots, use **Soft Reboot** — it's graceful and lets Windows shut down properly. Use **Restart** only when the VM is stuck or unresponsive. **Stop** is for when you're truly done and want resources back.

DataVolume Actions Explained

DataVolume resources have a clone action available.

Action	What It Does	When to Use It
Clone	Creates an exact copy of the volume, including its data and storage configuration. This will then be tracked by Orion and shown in the Genesi Data Volumes table. It will also be available to add to a workload template from within Genesis.	Use this if you want to utilize this Data Volume for another workload. Example, when creating a golden image for a VM workload

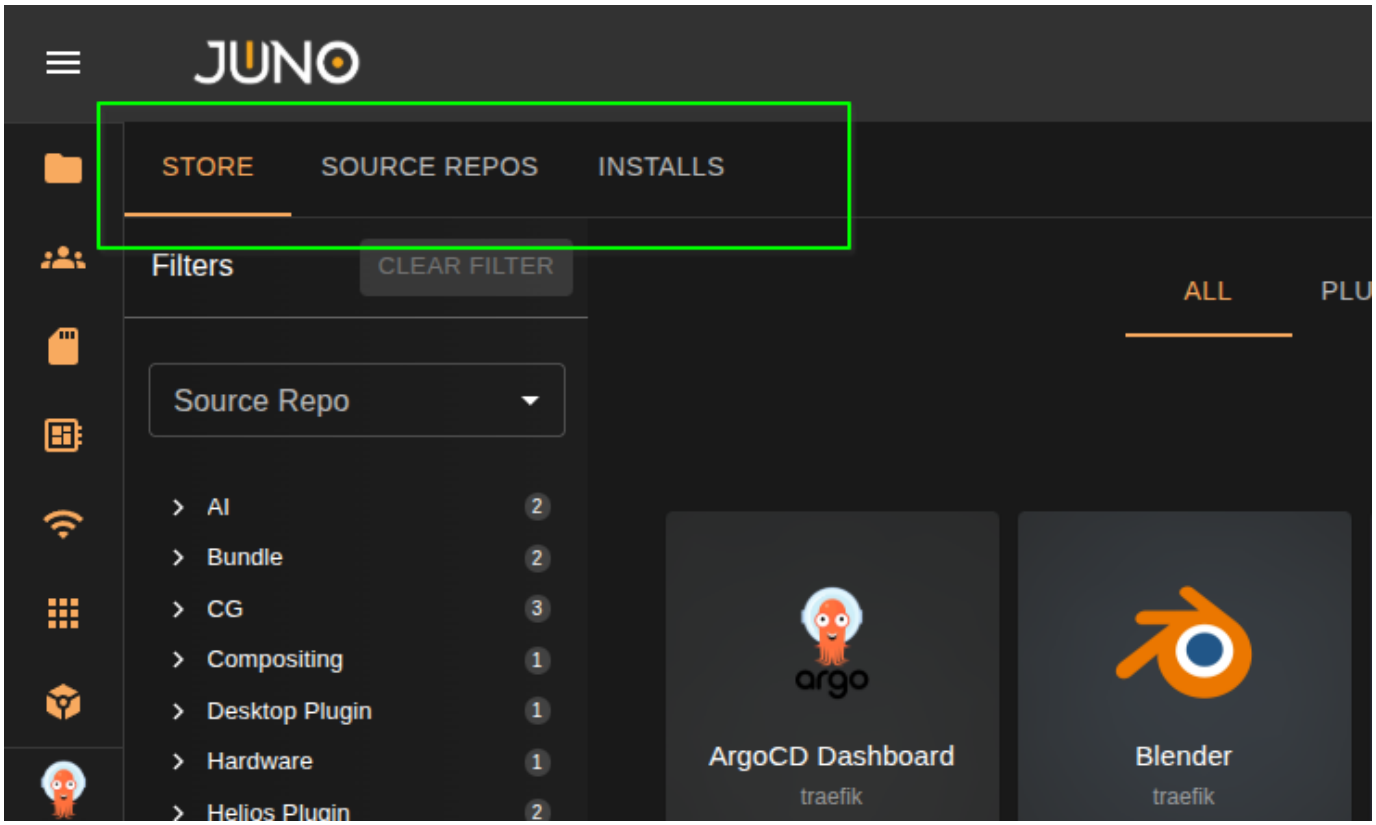
4.3 Terra

4.3.1 Introduction

Terra is Juno's open-source Kubernetes application marketplace that serves as a plugin system utilizing Helm charts in conjunction with Argo CD. Terra opens the door to the whole world of Kubernetes, expanding far beyond basic workload deployments by providing access to thousands of existing applications through a standardized, one-click deployment system.

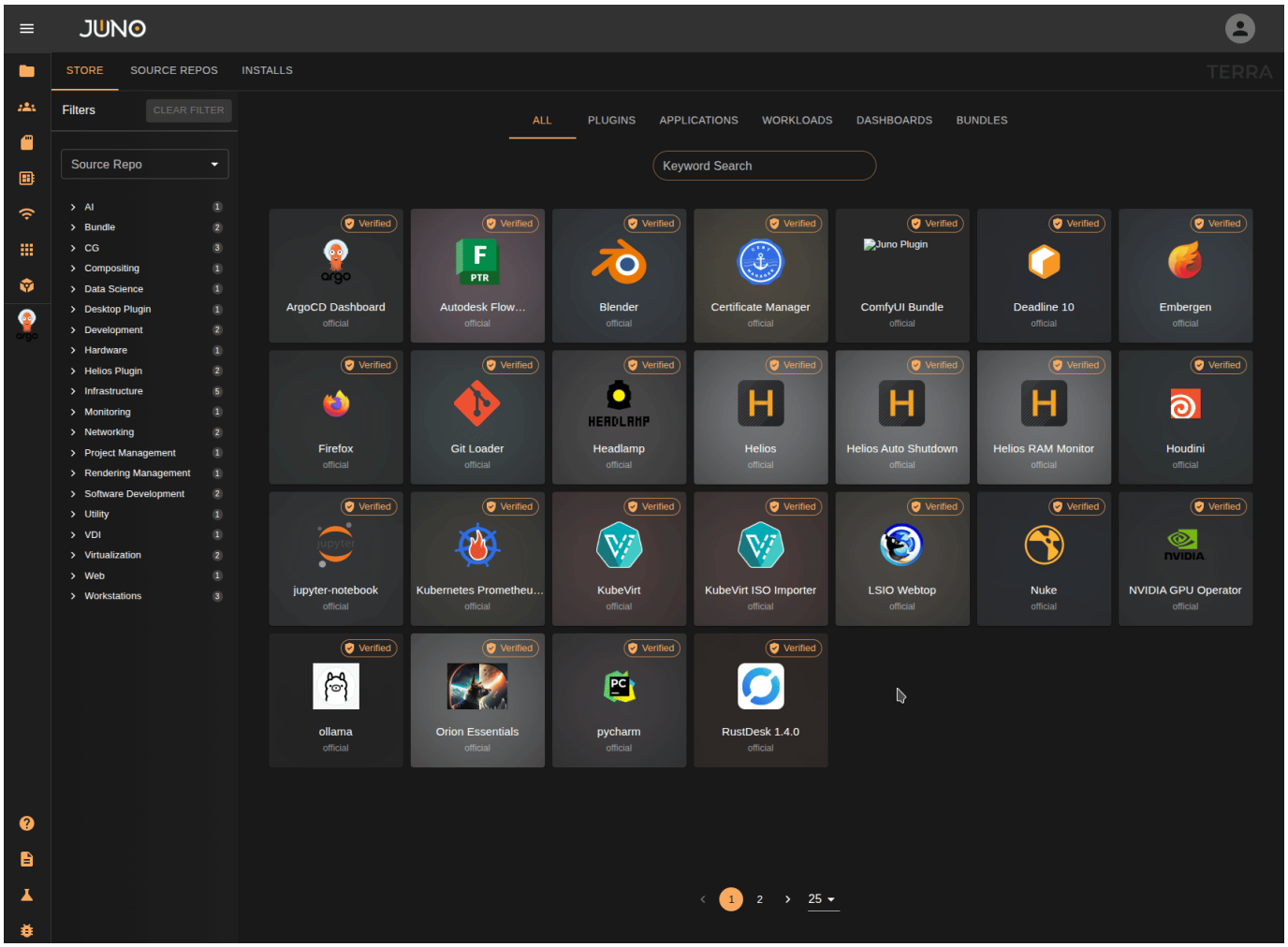
> **Initial Source**
>
> When first using Terra, it will not have any plugin sources configured. You can add the official Juno repository by following this guide: Adding the Juno Official Plugins Repository.

At the top of the Terra marketplace, users will see tabs to navigate between the main store, source repositories, and their installed applications.



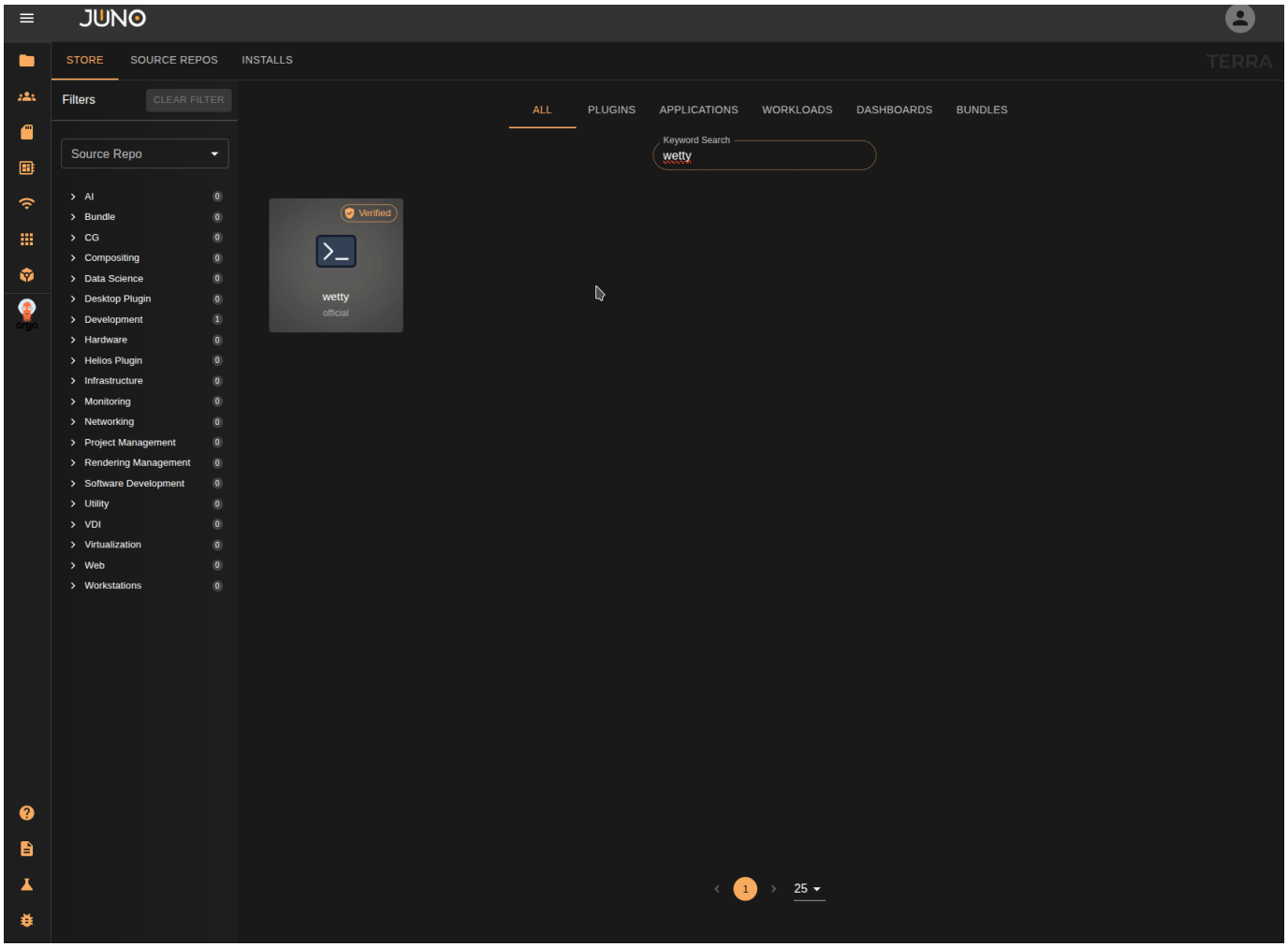
App Store

When first navigating to the marketplace, users will see a list of available plugins to install as well as search tools. You can click on the plugin cards to view the plugin details and install form.



INSTALL A PLUGIN

1. Click which plugin you want to install.
2. Fill out the install form and submit.
3. Navigate to the terra `installs` tab. Read more about the installs tab here
4. Click the refresh button to get the latest status of the installed plugin or if you have the ArgoCD Dashboard plugin installed. View the install within Argo. Read more about the plugin here

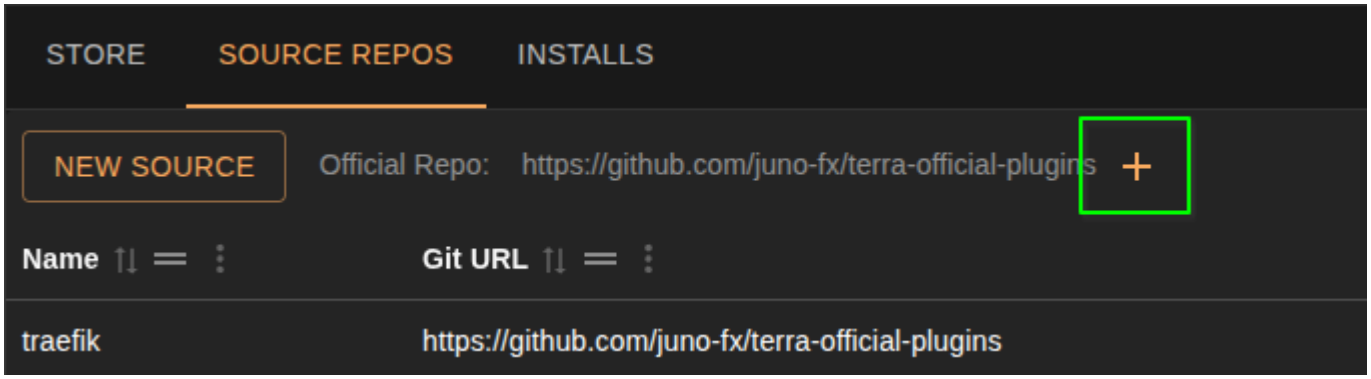


Source Repositories

Here you can manage your Terra git repository sources. Read more about Terra Repositories here

ADD OFFICIAL SOURCE

Click on the plus icon next to the Official Repo URL to add the Juno Innovations Official Terra Repository. We recommend forking the official repository. Read more about contributing here

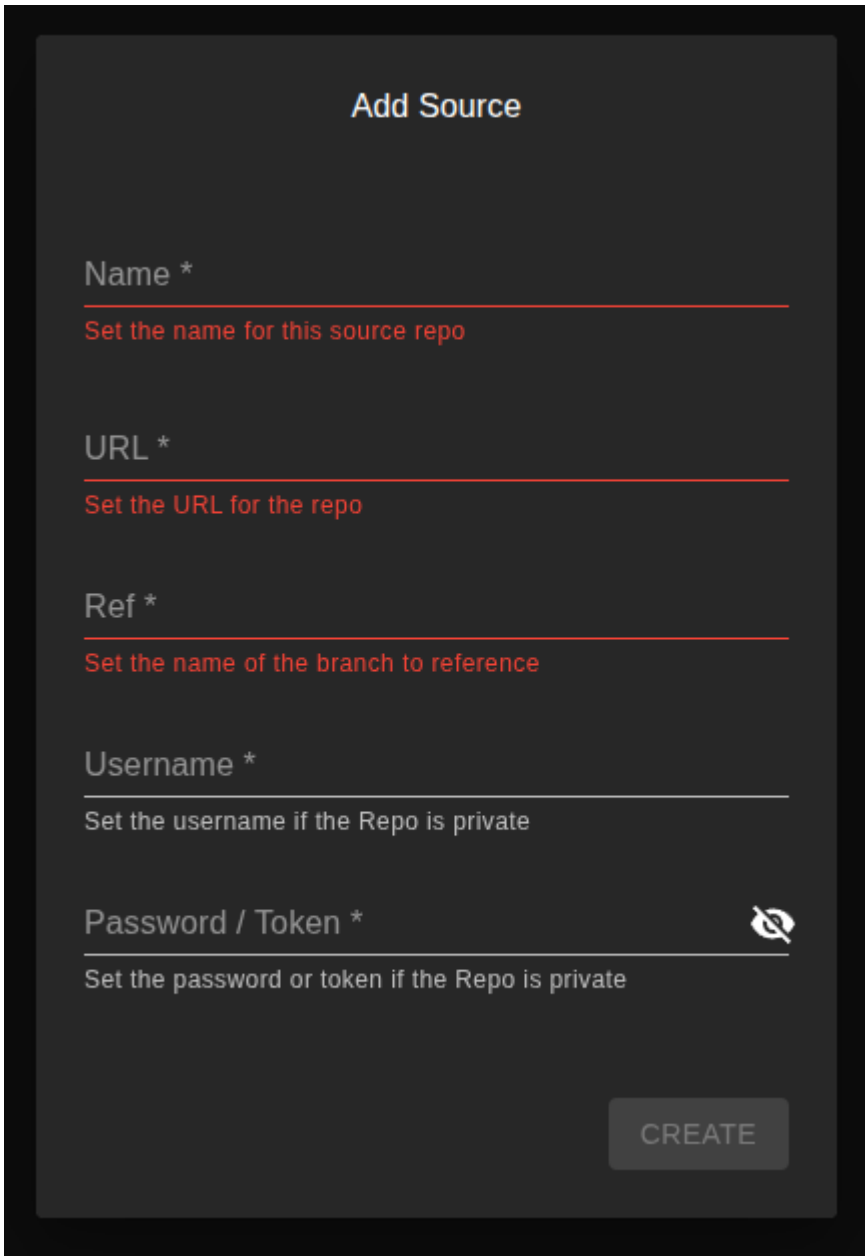


CREATE NEW SOURCE

1. Navigate to Terra `source repo` tab.
2. Click the NEW SOURCE button.

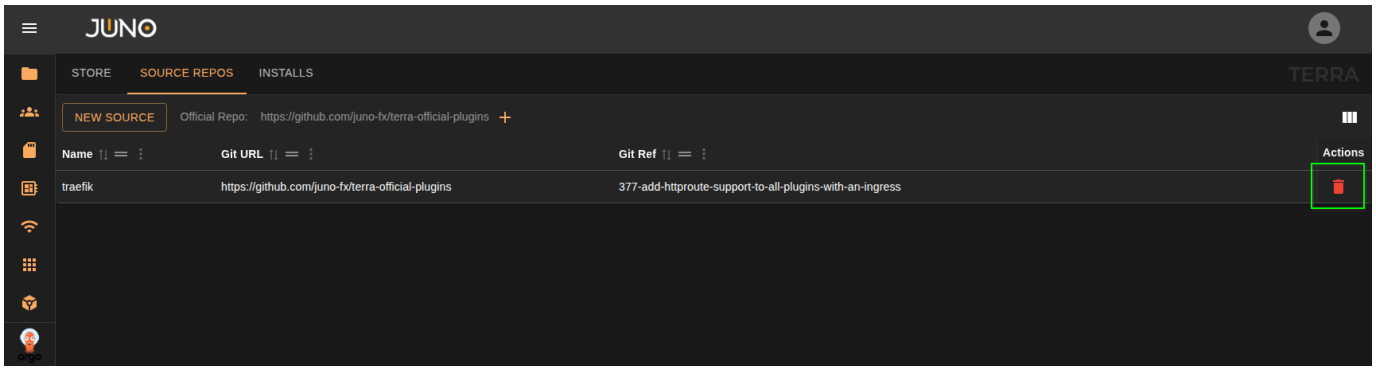
3. Fill in the form and submit.

Field	Required	Description
Name	Yes	This will be your reference for the repo within the terra app store.
URL	Yes	Git URL for the repo.
Ref	Yes	Git branch you want to pull from.
Username	No	If the repo is private, provide a username that can access it.
Password/Token	No	If the repo is private, provide a password/token that can access it.



DELETE SOURCE

Click the delete icon on the source row to delete. If you delete a source and a plugin is installed from that source. The plugin will be labeled as orphaned in the installs table. The plugin is still usable but simply cannot be tracked back to its original source. You can add the source back and terra will pick it up and re-sync the plugins installed from that source.



Plugin Installs

Here you can view and manage your currently installed plugins. The install table groups installs by namespace, we do this so you can see which plugins are a part of which projects. At anypoint you want to refresh the table to get the latest data. Click the refresh button located in the top left of the table.

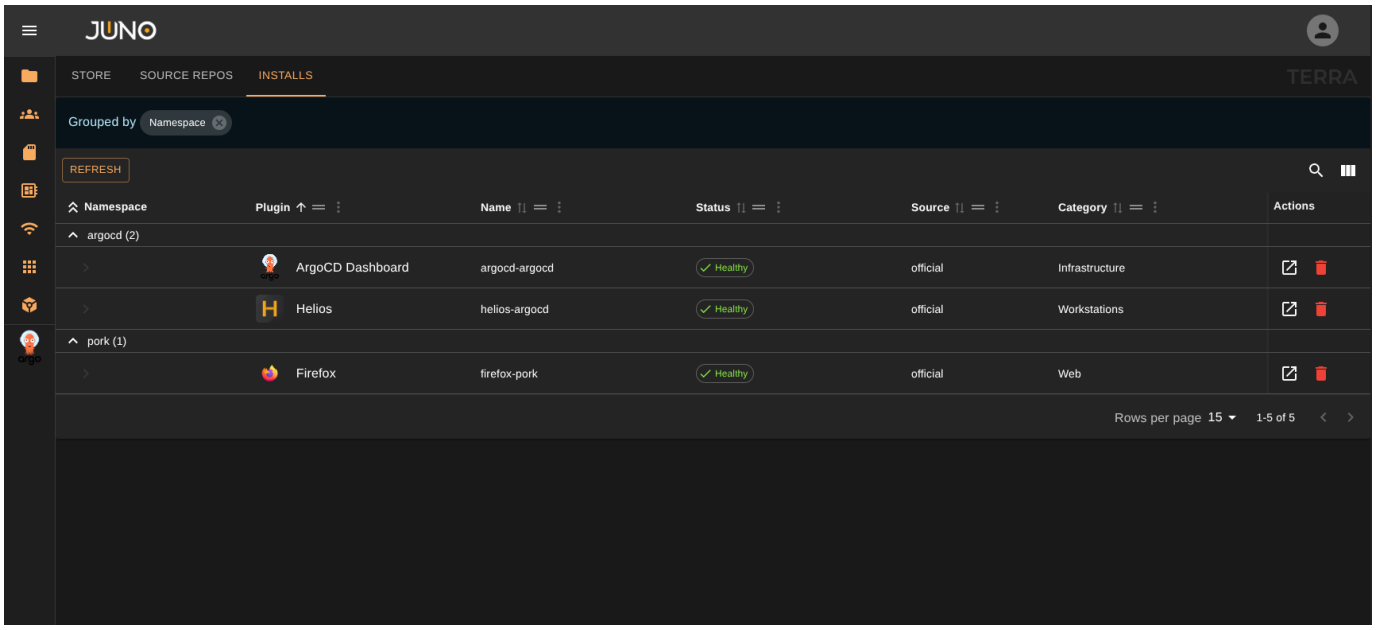
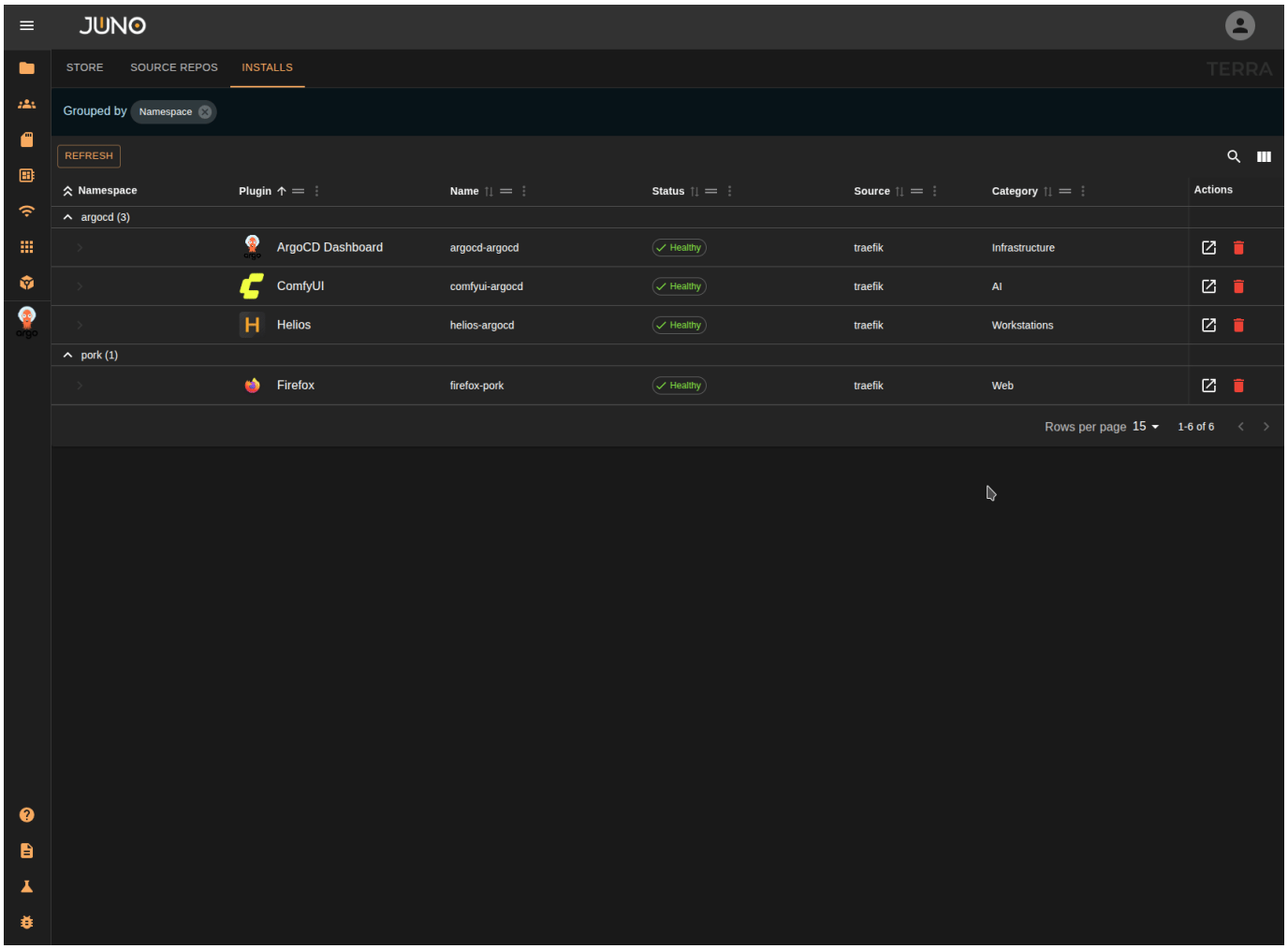


TABLE ACTIONS

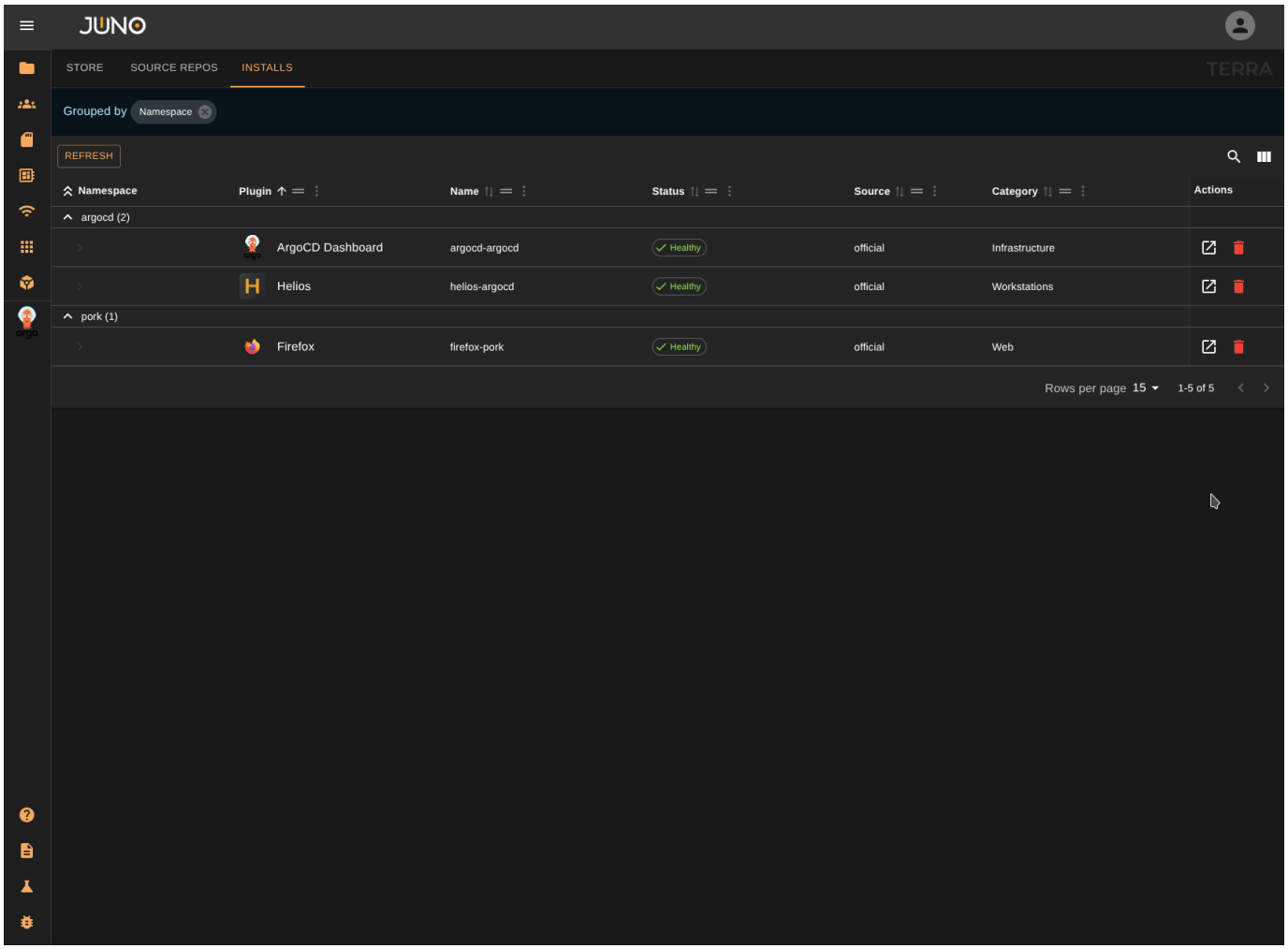
Uninstall

Click the delete icon in the action column for the plugin, then click the refresh button to update its status. Once the plugin is removed from the table it is fully uninstalled.



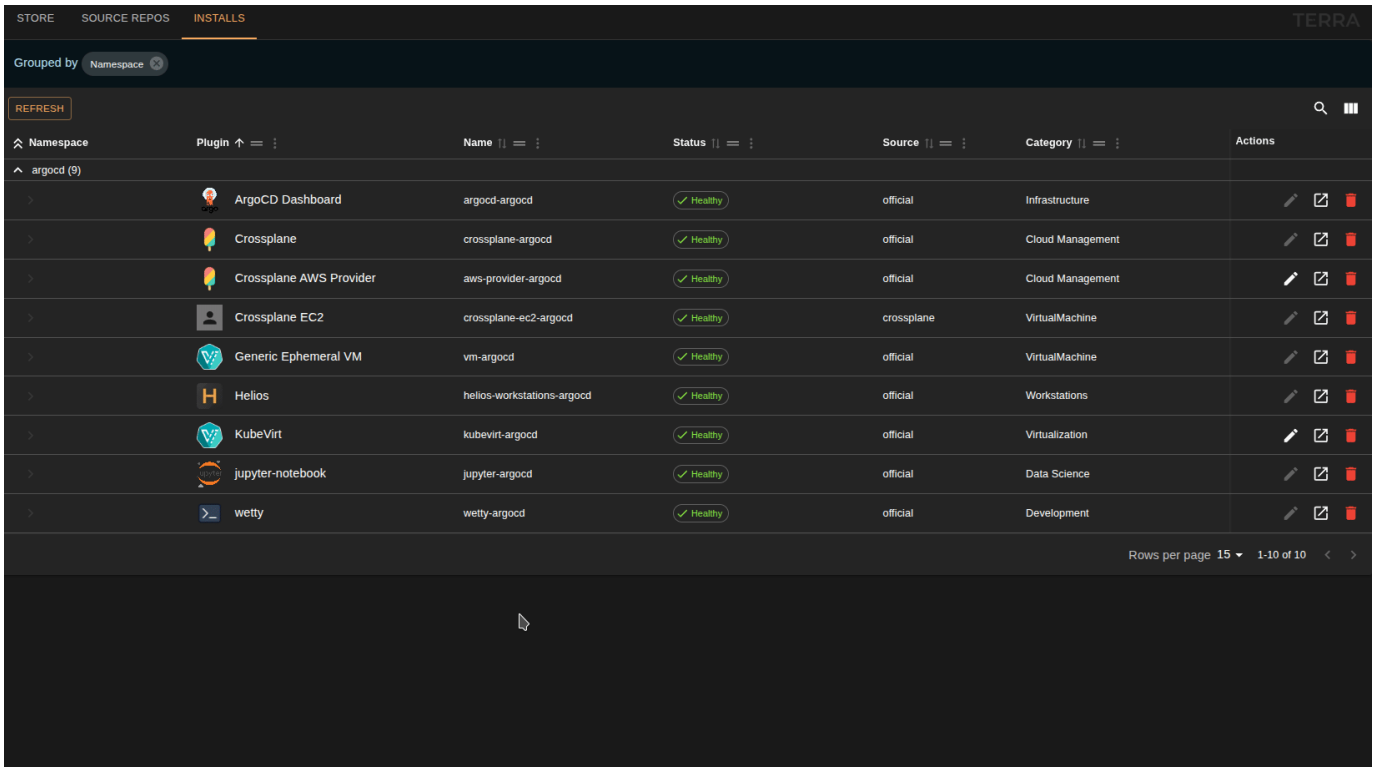
Open in ArgoCD Dashboard

If you have the ArgoCD Dashboard plugin installed. You can navigate to the installed plugin in ArgoCD directly from the install table. Click on the link icon in the action column on each row. Read more about the ArgoCD Dashboard plugin here



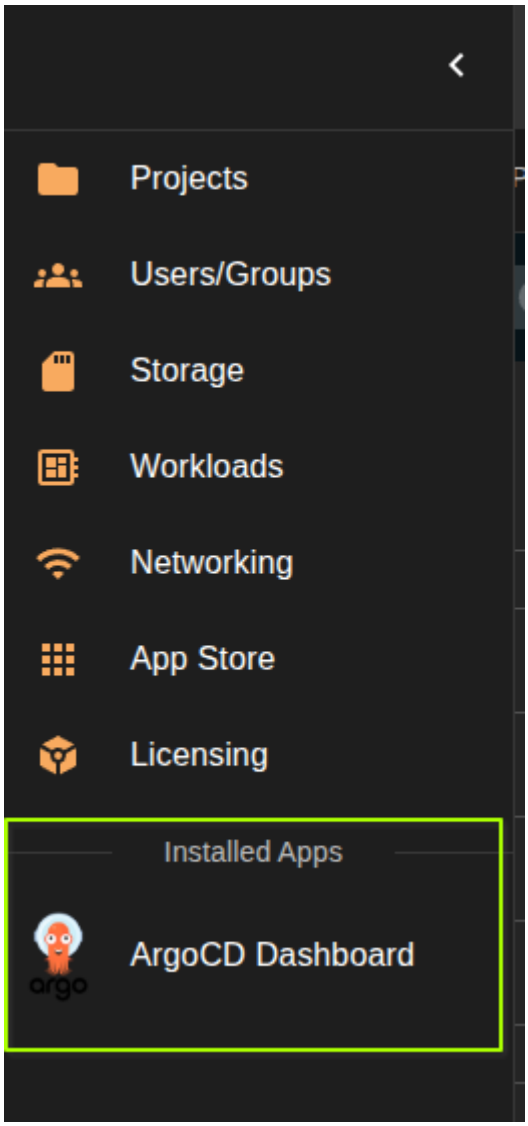
Edit Install

If a Terra plugin author marks a plugin as `editable` within the terra.yaml, users will be able to edit the install via the install table. A pencil icon will appear and be clickable, providing access to the original install form, with the current values pre-populated.



DASHBOARD PLUGINS

Installed dashboard plugins will show up in the sidebar. This allows for easy access to them while you are navigating genesis. Read more about dashboard plugins here



4.4 Rhea

4.4.1 Introduction

Rhea is a small component of the Orion suite that handles Authentication and Authorization, it is deployed alongside all other components and is a part of any inter-service communication.

At the time of writing Rhea is a internal only application and does not supply any functionality to the user interface.

5. API Reference

5.1 Getting Started

Orion is an API first platform. Each service ships with its own REST API and OpenAPI documentation. This allows you to interact with the services directly and build your own custom integrations. Getting Started is quick and easy.

Goal

We will follow the process of creating a python script that will communicate with Orion's Kuiper API to launch a workload.

Permissions

Orion is a role-based system. To get started, you will need to have proper permissions. In this example, we are going to interact with Kuiper to launch a workload. To do this, you will need to have the `kuiper` or `admin` role. If you aren't assigned to either role. You'll need to have an `admin` or `titan` user add you to the needed group. See User Management documentation for details.

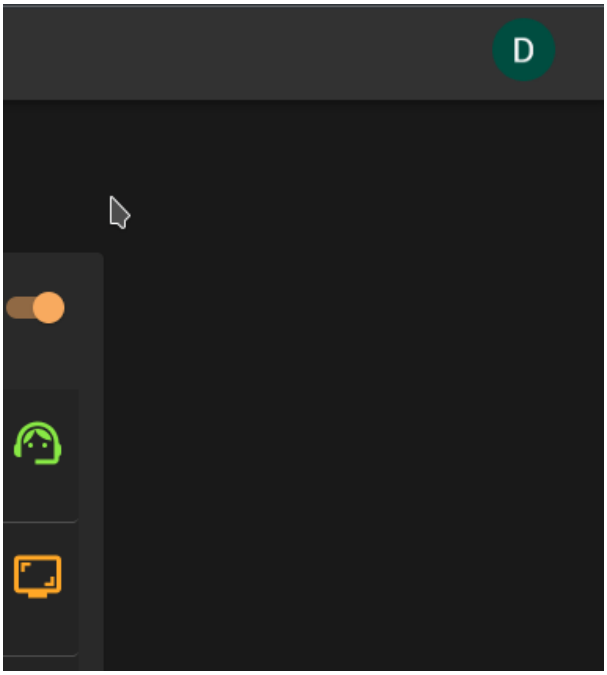
Generating API Key

Juno uses standard API keys that are passed to the API in the `Authorization` header.

> **Warning**
>
> API keys are sensitive information. Do not share them with anyone.

1.1 Generate API Token

In Genesis click on your user icon in the top right corner select Create API Token.



Warning

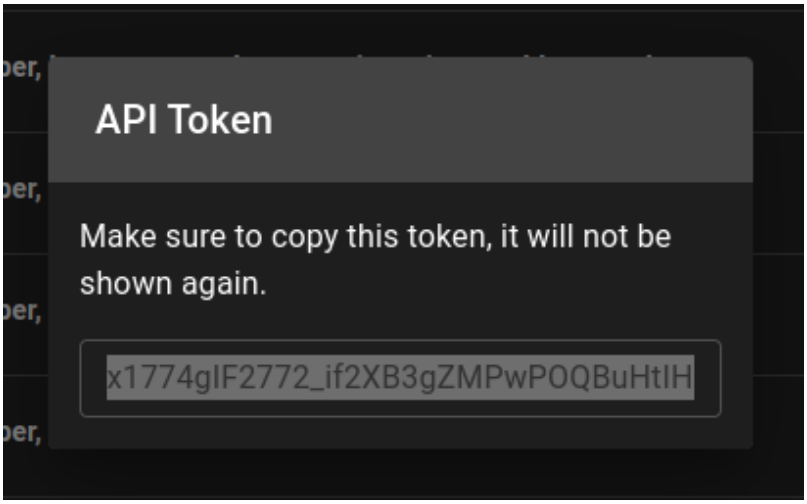
API keys are only displayed once. If you lose your API key, you will need to generate a new one.

Warning

Only one API key exists per user. If you generate a new API key, the old one will be invalidated.

1.2 Copy API Key

Your API key will be displayed. Copy it to your clipboard.



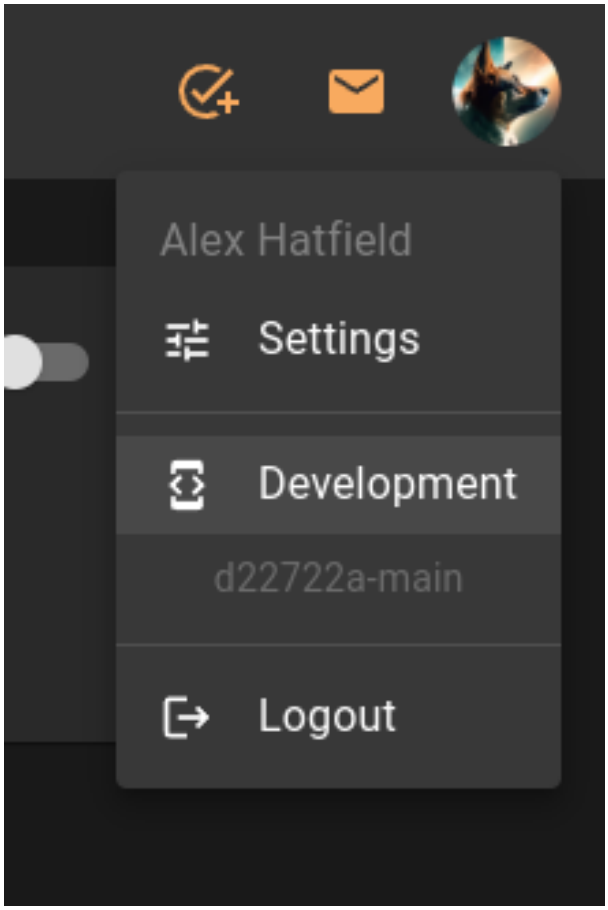
Now that we have our API key, save it some place safe and let's move on to the next step.

OpenAPI Swagger

Orion ships with OpenAPI documentation for each service that is actively installed. This documentation is custom tailored to your installation and will provide you with the necessary information to interact with the service. To access the documentation, you will need to navigate to the integrated documentation section either in Hubble or Genesis. Hubble will have API documentation for project level services, such as `Kuiper` workload management. While Genesis will have API documentation for cluster level services such as `Genesis`, `Titan`, and `Terra`.

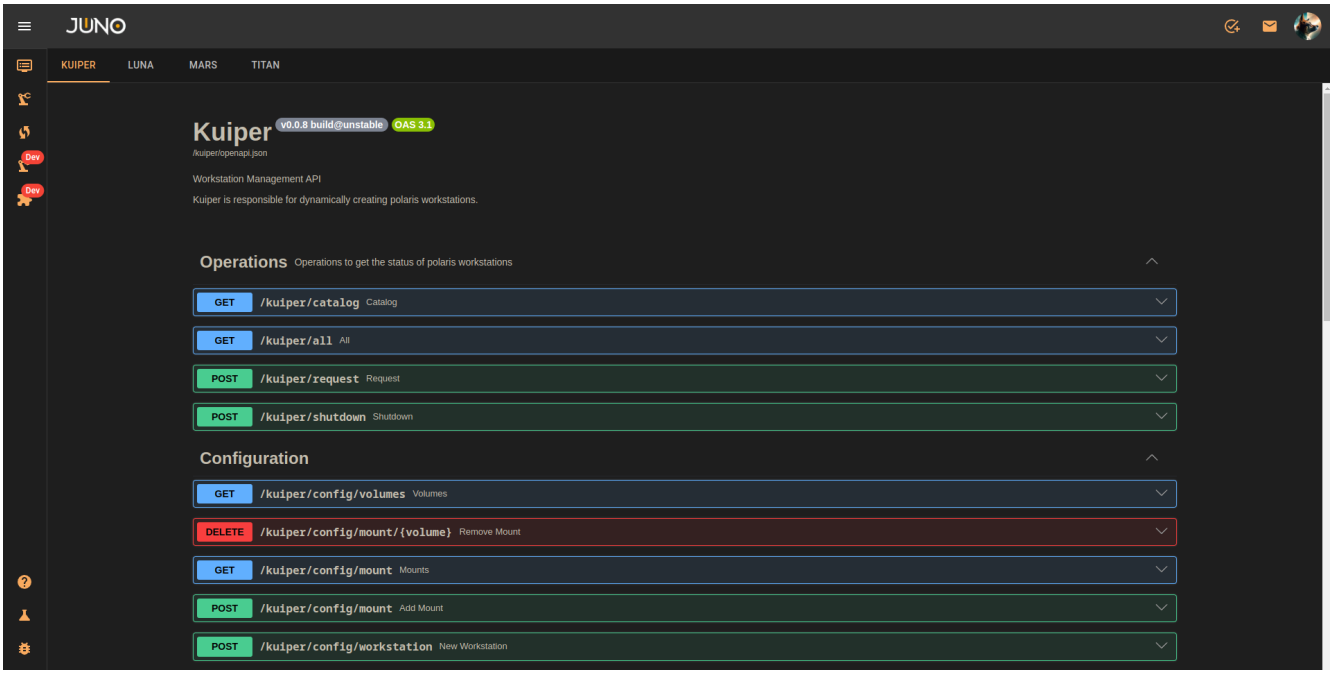
1.3 Open Development

Open the documentation page by clicking on your user icon in the top right corner and selecting Development



1.4 Open Integrated OpenAPI Documentation

Open the documentation section in the Development page for Kuiper. Here you can find the OpenAPI documentation for each service and even try them out.



Now we know where to find our API's documentation, let's move on to the next step and start writing our script.

Writing Our Integration

We are going to write a simple python script that will interact with Kuiper to launch a workload. We will use the `requests` library to make the API calls.

> **Warning**
>
> If you get a `requests.exceptions.SSLError` you can pass in the `verify=False` parameter to the request to disable SSL verification.

1.5 Setup Virtual Environment

We will be using Python 3.11 and a virtual environment to manage our dependencies. Then we will activate it.

```
$ python3.11 -m venv venv && source venv/bin/activate

---> 100%
```

1.6 Install Requests

We will be using the python requests library to make our API calls. Let's install it.

```
$ pip install requests

---> 100%
```

1.7 Import Credentials

We will be passing our API key and server URL as environment variables. Let's import them.

```
import os
import requests
from time import sleep

server = os.environ['SERVER']
token = os.environ['TOKEN']
```

1.8 Identify Ourselves

Since we will be triggering this from a different application, we need to be able to identify ourselves.

```
# Identify the user via token
rsp = requests.get(
    f'{server}/titan/identity',
    headers={'Authorization': token}
)
user = rsp.json()['name']  # provides the user's name associated with the token
```

1.9 Get Available Workload Templates

Now we need to get the catalog of available workload types that we can launch. We will then select the first one for now.

```
# Get workload catalog
rsp = requests.get(
    f'{server}/kuiper/catalog',
    headers={'Authorization': token}
)

# Select first workload template and get it's idx
idx = rsp.json()[0]['idx']  # the idx is a unique identifier for the workload template that is used to launch it
```

1.10 Request Launch Workload

Now we have everything we need to request the workload to launch.

```
# Create a new workload
rsp = requests.post(
    f'{server}/kuiper/request',
    headers={'Authorization': token},
    json={'instance_type': idx, 'user': user},
)
workload_name = rsp.json()['name']  # the name of the workload that was launched
```

1.11 Wait for Workload to Launch

Now that the request has been put in, we now can watch for the API to flag that the workload has been launched and is ready. We do this by entering a loop that will check the status of the workload every 5 seconds. Once the workload is ready, it will print the join link which consists of the workloads name and your servers url. Once clicked, it should open your browser and take you to the workload where you will need to log in.

```
# Wait for the workload to be ready
while True:
    rsp = requests.get(
        f'{server}/kuiper/{user}/all',
        headers={'Authorization': token}
    )
    data = rsp.json()
    for workload in data:
        name = workload["name"]
        if name != workload_name:
            continue
        state_object = workload['state'][0]
        message = state_object['message']
        state = state_object['state']

        if message:
            print(f'{name}: {message}')
        if state == 'running':
            print(f'{name}: {state}')
            print(f"Join Link: {server}/viewer/{name}")
            break

        print(f'{name}: {state} (checking again in 5 seconds)')
    else:
        sleep(5)
        continue
    break
```

1.12 Run Your Integration

Now we run it and see the results!

```
$ export TOKEN="your-token"
$ export SERVER="https://orion-install"
$ python launch_workload.py

---> 100%

Join Link: https://orion-install/viewer/your-workload-name
```

5.1.13 Full Script

```python
import os
import requests
from time import sleep

server = os.environ['SERVER']
token = os.environ['TOKEN']


# Identify the user via token
rsp = requests.get(
    f'{server}/titan/identity',
    headers={'Authorization': token}
)
user = rsp.json()['name']

# Get workload catalog
rsp = requests.get(
    f'{server}/kuiper/catalog',
    headers={'Authorization': token}
)

# Select first workload template and get its idx
idx = rsp.json()[0]['idx']

# Create a new workload
rsp = requests.post(
    f'{server}/kuiper/request',
    headers={'Authorization': token},
    json={'instance_type': idx, 'user': user},
)
workload_name = rsp.json()['name']

# Wait for the workload to be ready
while True:
    rsp = requests.get(
        f'{server}/kuiper/{user}/all',
        headers={'Authorization': token}
    )
    data = rsp.json()
    for workload in data:
        name = workload["name"]
        if name != workload_name:
            continue
        state_object = workload['state'][0]
        message = state_object['message']
        state = state_object['state']

        if message:
            print(f'{name}: {message}')
        if state == 'running':
            print(f'{name}: {state}')
            print(f"Join Link: {server}/viewer/{name}")
            break

        print(f'{name}: {state} (checking again in 5 seconds)')
    else:
        sleep(5)
        continue
    break
```

Summary

We have successfully written a python script that interacts with Kuiper to launch a workload. As you can imagine, this is just the beginning, and the possibilities are endless. Orion can be integrated with any system that can make HTTP requests. This allows you to build custom integrations with your existing tools and services.

5.1.14 Examples

- External User Sync
- Passing Environment Variables to Workload at Launch

5.2 Examples

5.2.1 External User Sync

This example demonstrates how to sync users from an external source to Orion's `titan` service for workload launching.

Requirements

- A running Orion instance
- Orion API Key
- `titan` role assigned to the user running the script
- External source that provides all the following fields for Users
- `username` - POSIX username
- `uid` - POSIX user ID
- `email` - Email address
- External source that provides all the following fields for Groups/Roles
- `name` - POSIX username
- `guid` - POSIX Group ID
- `members` - List of its members (usernames, uids, or emails)

Source Data

Since the data source can be anything, we will use a simple JSON file as an example. This data could be pulled from something like Active Directory, LDAP, or any other user management system. Juno internally uses Google Workspace and AWS Cognito for user management and this is how users are synced in for us.

Script

> **Warning**
>
> If you get a `requests.exceptions.SSLError` you can pass in the `verify=False` parameter to the request to disable SSL verification.

Data from our external source can look like this:

```
user_data = [
    {
        "username": "johndoe",
        "uid": 1050,
        "email": "johndoe@example.com",
    }
]
group_data = [
    {
        "name": "corp_users",
        "guid": 3050,
        "members": [1050]
    }
]
```

User sync integration script

```
import os
import requests

server = os.environ['SERVER']
token = os.environ['TOKEN']

user_map = {}
```

```
# create our users first
for user in user_data:
    response = requests.post(
        f"{server}/titan/user",
        headers={"Authorization": f"{token}"},
        # while we know this is kind of silly to do this way,
        # it is just to show what is required to make the request
        json={
            "username": user["username"],
            "uid": user["uid"],
            "email": user["email"],
        }
    )

    # we know that our groups need the username and not the ID, so to
    # help minimize looping, we store for later indexing
    user_map[user["uid"]] = user['username']

# titan stores it membership with usernames and not uids, so we need to
# get the username for each user and then add them to the group
for group in group_data:
    members = []
    for member in group["members"]:
        members.append(user_map[member])  # get the username from the uid

    response = requests.post(
        f"{server}/titan/group",
        headers={"Authorization": f"{token}"},
        json={
            "name": group["name"],
            "guid": group["guid"],
            "members": members
        }
    )
```

```
$ export TOKEN="your-token"
$ export SERVER="https://orion-install"
$ python sync_users.py

---> 100%
```

This script will create the users and groups in the `titan` service for Orion. Once the users and groups are created, they will be automatically loaded into the workloads at launch time and membership will be assigned to match your `titan` configuration.

Verification

We can verify this by checking the `titan` service for its "state".

```
$ export TOKEN="your-token"
$ export SERVER="https://orion-install"
$ curl "$SERVER/titan/state" -H "Accept: application/json" -H "Authorization: $token"
{
  "users": [
    {
      "username": "johndoe",
      "uid": 1050,
      "email": "johndoe@example.com",
      "groups": [
        {
          "name": "corp_users",
          "uid": 3050
        }
      ]
    }
  ],
  "groups": [
    {
      "name": "corp_users",
      "uid": 3050,
      "members": [
        {
          "username": "johndoe",
          "uid": 1050,
          "email": "johndoe@example.com",
          "groups": [
            {
              "name": "corp_users",
              "uid": 3050
            }
          ]
        }
      ]
    }
  ]
}
```

We can also run `id` inside of a workload terminal and see our new membership and user is set up properly.

```
$ id
uid=1050(johndoe) gid=1050(johndoe) groups=1050(johndoe),3050(corp_users)
```

5.2.2 Passing Environment Variables to Workload at Launch

This example demonstrates how to pass environment variables to a workload at launch.

Requirements

- A running Orion instance
- Orion API Key
- `kuiper` role assigned to the user running the script

Script

> **Warning**
>
> If you get a `requests.exceptions.SSLError` you can pass in the `verify=False` parameter to the request to disable SSL verification.

Custom environment variables passed at launch

```
import os
import requests

server = os.environ['SERVER']
token = os.environ['TOKEN']
custom_var = os.environ['CUSTOM_VAR']
idx = int(os.environ['IDX'])


# Identify the user via token
rsp = requests.get(
    f'{server}/titan/identity',
    headers={'Authorization': token}
)
user = rsp.json()['name']

# Create a new workload
rsp = requests.post(
    f'{server}/kuiper/request',
    headers={'Authorization': token},
    json={
        'instance_type': idx,
        'user': user,
        'env': {
            'CUSTOM_VAR': custom_var
        }
    }
)
```

```
$ export TOKEN="your-token"
$ export SERVER="https://orion-install"
$ export IDX="999111555"
$ export CUSTOM_VAR="Hello, World!"
$ python custom_var.py

---> 100%
```

This script will create a workload with a custom environment variable `CUSTOM_VAR` set to `Hello, World!`.

Verification

We can verify this by running `env` inside of a workload terminal.

```
$ env | grep CUSTOM_VAR
CUSTOM_VAR=Hello, World!
```

6. Resources

6.1 Coming Soon

We're working hard to update this document section soon.

6.2 Useful Resources

Continuous Learning Resources

- Orion Documentation
- Kubernetes Official Documentation
- Argo CD Documentation
- Prometheus Documentation
- Helm Documentation

7. Related Documentation

7.1 Related Documentation

Welcome to the Juno Innovations documentation ecosystem. Our documentation is organized across three main areas to help you find exactly what you need.

7.1.1 Documentation Sites

Platform Documentation

Orion Documentation

The main platform documentation covering:

- Installation and deployment guides
- Product overviews (Genesis, Hubble, Terra)
- API reference and integration examples
- Operations and management

Best for: System administrators, DevOps engineers, and anyone setting up or managing the Juno platform.

Workstation Documentation

Helios Workstations

Complete guide to building and customizing workstations:

- Getting started with workstation creation
- Package management and customization
- Filesystem modifications
- Build and event hooks
- Deployment strategies

Best for: Developers and administrators creating custom workstation images.

Plugin Development

Terra Official Plugins

Everything you need to know about Terra plugins:

- Plugin development workflow
- Configuration and best practices
- Complete plugin catalog by category
- Community contribution guidelines

Best for: Developers creating or customizing Terra plugins for specific workflows.

7.1.2 Quick Links

External Resources

- **GitHub Organization:** github.com/juno-fx
- **Docker Hub:** hub.docker.com/u/junoinnovations
- **LinkedIn:** Juno Innovations Company Page
- **Discord:** Discord Community

7.1.3 Need Help?

If you can't find what you're looking for:

1. Use the search function (press `/` to focus)
2. Check the FAQ sections in each documentation site
3. Reach out to our support team
4. Join our community discussions on GitHub

All documentation is kept up-to-date with the latest releases. For version-specific documentation, use the version selector in the header of each site.

8. Support

8.1 Support Documentation

8.1.1 Support Details

Primary Support Channels

EMAIL SUPPORT (PRIMARY)

- **Email**: support@juno-innovations.com
- **Best For**: Technical issues, incident reports, detailed troubleshooting

DISCORD COMMUNITY

- **Server**: Juno Innovations Community
- **Best For**: Quick questions, community discussion, real-time chat

SALES & LICENSING

- **Email**: sales@juno-innovations.com
- **Purpose**: Licensing questions, renewals, pricing inquiries

Support Hours and Coverage

STANDARD SUPPORT HOURS

- **Business Days**: Monday - Friday
- **Time Zone**: Eastern Standard Time (EST)
- **Hours**: 9:00 AM - 6:00 PM
- **Coverage**: Full technical support team availability

8.2 Troubleshooting Guide

8.2.1 Overview

This guide provides helpful kubectl commands for general cluster troubleshooting

CONNECT TO ARGOCD

```
# Port forward argocd-server. Providing access to the argoCD GUI.
# View logs, sync status, deployed services etc.
# example connection: localhost:8080/argocd
kubectl port-forward svc/argocd-server -n argocd 8080:443
```

GENERAL CLUSTER CHECKS

```
# Quick cluster health check
kubectl cluster-info

# API version
kubectl version --short
```

NODES

```
# Get list of all nodes
kubectl get nodes

# describes the given node showing individual resource requests, limits, and age
kubectl describe node <node-name>

# describes all nodes showing individual resource requests, limits, and age
kubectl describe nodes

# Verify node health
kubectl describe nodes | grep -i "ready\|schedulable"

# Check system resource usage
kubectl top nodes
```

PODS

```
# Get all pods in all namespaces
kubectl get pods -A

# Get all pods in a specific name space
kubectl get pods -n <namespace>

# describe a specific pod
kubectl describe pod <pod-name> -n <namespace>

# Get a pods logs
kubectl logs <pod-name> -n <namespace>
```

JOBS

```
# Get all pods in all namespaces
kubectl get jobs -A

# Get a specific job
kubectl get job <job-name>

# Describe a specific job
kubectl describe job <job-name>

# Delete a job
kubectl delete job <job-name>
```

DEPLOYMENTS

```
# Get deployments in a specific namespace
kubectl get deploy -n <namespace>

# Get deployments in all namespaces
kubectl get deploy -A

# describe a specific deployment
kubectl describe deploy <deployment-name> -n <namespace>
```

```
# Force restart all Orion deployments
kubectl rollout restart deployment -n argocd

# Force restart a specific deployment
kubectl rollout restart deployment/<deployment-name> -n <namespace>
```

STORAGE

```
# Check persistent volumes
kubectl get pv
kubectl get pvc -A

# Verify storage class
kubectl get storageclass
```

NETWORKING AND INGRESS

```
# Check ingress controller status
kubectl get pods -A -l app.kubernetes.io/name=ingress-nginx

# Get ingress from specific namespace
kubectl get ingress -n <namespace> -o yaml

# Get all network policies
kubectl get networkpolicy -A

# describe a specific network policy
kubectl describe networkpolicy <policy> -n <namespace>
```

EVENTS

```
# Get all events from a specific namespace
kubectl get events -n <namespace>
```

9. Changelogs

9.1 Orion Features Changelog

This changelog contains new features where you can review the latest and greatest Orion has to offer. The changelog shows changes both in Orion's management layer, Genesis, as well as the environment/project deployments.

For technical deployment information, deprecations, and migration steps, see the Technical Changelog.

9.1.1 2026-06-01

Components updated in this release:

- **Genesis Management Layer**: v4.0.1 - https://github.com/juno-fx/Genesis-Deployment/tree/v4.0.1
- **Orion Projects**: v4.0.1 - https://github.com/juno-fx/Orion-Deployment/tree/v4.0.1

Bug Fixes

Genesis (v5.0.1)

- Fixed a bug that was blocking the ability to submit LDAPS_CA_CERT_BASE64 field value in the admin configuration settings

Hubble (v6.0.1)

- Fixed an issue where sometimes on startup the workloads template catalog would not load
- Fixed an issue where sometimes the templates catalog would not filter correctly, causing an error.

Maintenance Fixes

Genesis (v5.0.1)

- General stability updates

Security Fixes

Genesis (v5.0.1)

- Upgraded upstream dependency in response to known vulnerabilities

Hubble (v6.0.1)

- Upgraded upstream dependency in response to known vulnerabilities

---

9.1.2 2026-05-11

Components updated in this release:

- **Genesis Management Layer**: v4.0.0 - https://github.com/juno-fx/Genesis-Deployment/tree/v4.0.0
- **Orion Projects**: v4.0.0 - https://github.com/juno-fx/Orion-Deployment/tree/v4.0.0

Enhancements

Genesis (v5.0.0)

- Added in backend user authorization via Rhea, for better backend security.

Hubble (v6.0.0)

- Added in backend user authorization via Rhea, for better backend security.

Rhea (v1.2.3)

- Updates for user and group authorization handling

Bug Fixes

Genesis (v5.0.0)

- Fixed a bug where required select fields in the workload templates was not being respected
- Fixed a bug where adding project customizations would grey out the ability to click deploy when editing a project

Hubble (v6.0.0)

- Ensure the workload connect button does not show when a VM is stopped
- Fixed a bug that sometimes caused a user to be re-directed to the unauthorized page when they login and are authorized.

Maintenance Fixes

Genesis (v5.0.0)

- Updated the Google provider logos to ensure the links no longer break
- Deprecated the node role endpoints. The newer label endpoints are now favored
- Updated sales and support links to point to the Juno-Innovations website forms
- Updated re-route when deleting a project, to route the user back to the projects page, rather than the home page.
- General stability updates

Hubble (v6.0.0)

- General stability updates
- Updated sales and support links to point to the Juno-Innovations website forms
- Updated the Google provider logos to ensure the links no longer break

Titan (v2.1.2)

- General stability updates

Terra (v2.1.1)

- General stability updates

Security Fixes

Genesis (v5.0.0)

- Upgraded upstream dependency in response to known vulnerabilities

Hubble (v6.0.0)

- Upgraded upstream dependency in response to known vulnerabilities

---

9.1.3 2026-05-04

Components updated in this release:

- **Orion Projects**: v3.1.3 - https://github.com/juno-fx/Orion-Deployment/tree/v3.1.3

Bug Fixes

Kuiper (v4.1.3)

- Fixed bug to ensure proper resource version tracking

Maintenance Fixes

Kuiper (v4.1.3)

- Now utilizing open-source orion-py repo

9.1.4 2026-04-23

Components updated in this release:

- **Orion Projects**: v3.1.2 - https://github.com/juno-fx/Orion-Deployment/tree/v3.1.2

Bug Fixes

Kuiper (v4.1.2)

- Fixed an issue that could cause requested instances to not appear on frontend

---

9.1.5 2026-04-21

Components updated in this release:

- **Orion Projects**: v3.1.1 - https://github.com/juno-fx/Orion-Deployment/tree/v3.1.1

Bug Fixes

Kuiper (v4.1.1)

- Fixed an issue that could cause stale data being returned to the frontend
- Fixed an issue that could cause stale connections between Kuiper and Hubble to remain open

---

9.1.6 2026-04-14

Components updated in this release:

- **Genesis Management Layer**: v3.0.2 - https://github.com/juno-fx/Genesis-Deployment/tree/v3.0.2
- **Orion Projects**: v3.1.0 - https://github.com/juno-fx/Orion-Deployment/tree/v3.1.0

Enhancements

Kuiper (v4.1.0)

- Crossplane AWS instance support added
- Now tracking API groups `ec2.aws.crossplane.io`, `crossplane.juno-innovations.com`, `apiextensions.crossplane.io`

Hubble (v5.1.0)

- Added details type support for endpoint connections.

Bug Fixes

Kuiper (v4.1.0)

- Fixed an issue that may cause connection drops between Kuiper and Hubble

Hubble (v5.1.0)

- Fixed a display issue with the workload connect button, that could appear for some workloads who have a single connection endpoint
- Fixed an issue that may cause connection drops between Kuiper and Hubble

Security Fixes

Genesis (v4.0.2)

- Upgraded upstream dependency in response to known vulnerabilities

Hubble (v5.1.0)

- Upgraded upstream dependency in response to known vulnerabilities

Maintenance Fixes

Genesis (v4.0.2)

- General stability updates

Hubble (v5.1.0)

- General stability updates

9.1.7 2026-04-10

Components updated in this release:

- **Genesis Management Layer**: v3.0.1 - https://github.com/juno-fx/Genesis-Deployment/tree/v3.0.1
- **Orion Projects**: v3.0.1 - https://github.com/juno-fx/Orion-Deployment/tree/v3.0.1

Security Fixes

Genesis (v4.0.1)

- Upgraded upstream dependency in response to known vulnerabilities

Hubble (v5.0.1)

- Upgraded upstream dependency in response to known vulnerabilities

---

9.1.8 2026-04-03

Components updated in this release:

- **Genesis Management Layer**: v3.0.0 - https://github.com/juno-fx/Genesis-Deployment/tree/v3.0.0
- **Orion Projects**: v3.0.0 - https://github.com/juno-fx/Orion-Deployment/tree/v3.0.0

Updates in this release include all of the updates from the v3.0.0-beta.1 release on 2026-03-18 as seen below. As well as the following updates.

Enhancements

Genesis (v4.0.0)

- Projects now can be configured to enable/disable workload sharing between users. This by default is now disabled.

Hubble (v5.0.0)

- Projects will now respect the configuration for workload sharing, as set within the Genesis project configuration form.

Bug Fixes

Genesis (v4.0.0)

- The node labels table, will now allow users to create the same node label more than once, allowing for different nodes to have different values for this label. Previously this was incorrectly being blocked.

Maintenance Fixes

Genesis (v4.0.0)

- General backend and frontend maintenance.

Hubble (v5.0.0)

- General backend and frontend maintenance.

9.1.9 2026-03-18

Components updated in this release:
- **Genesis Management Layer**: v3.0.0-beta.1 - https://github.com/juno-fx/Genesis-Deployment/tree/v3.0.0-beta.1 -
- **Orion Projects**: v3.0.0-beta.1 - https://github.com/juno-fx/Orion-Deployment/tree/v3.0.0-beta.1

Enhancements

Genesis (v4.0.0-beta.1)

- StorageClass, ingressClass, and dataVolume field type support added to workload schemas
- added LOGLEVEL support. Users can now set the log level via the Genesis env. example: `LOGLEVEL: DEBUG` will enable the debug logger.
- Added endpoints for KubeVirt data volume support. Users now can get and delete KubeVirt data volumes. Tracked data volumes will include the label: `kuiper.juno-innovations.com/user-created-datavolume`
- Added some of the allocatable resources to network nodes tabe. Showing CPU, Memory, Ephemeral-storage, and the currently running pods.
- Added node labels table in networking tab, as well as into node details panel. Users can now see, edit and create new node labels. All from the Genesis UI.
- Backend endpoints added for assigning and unassigning node labels
- Added `access` to our storage schemas. Allowing for admins to restrict access to a storage mount based on user and/or group. If a storage mount is restricted for a user, it will not be mounted during workload init.
- If KubeVirt is installed a new Data Volumes table will be available via the storage page.
- Dynamic basic auth providers added. Admins can now create as many basic auth users as they like, rather than being limited to 2. As a reminder this is not intended for production environments.
- You can now see the configured basic auth email via the frontend admin settings
- Added better error handling, and front end pop up UI showing any errors that have occurred.
- Added icons to workload schema select drop down
- Added icon preview to workload schema templates
- Added new support and sales links to side bar. Replacing the older bug/feature request links.
- Added ability to specify an Ansible temp folder via frontend for node provisioning
- Added support for editable Terra installs. Users can now edit a Terra plugin install via the installs table, if the plugin author marked the plugin as editable.

Hubble (v5.0.0-beta.1)

- Added resources table to workload details panel. Showing each resource in the workload, their status, Kind type, name, current Node, last event, and available actions.
- Added support for resource actions, available via the resources table. Including KubeVirt virtual machine actions support.
- Updated workload status column to show the number of resources that are ready out of the total number of resources.
- Added new support and sales links to side bar. Replacing the older bug/feature request links.
- Added better error handling, and front end pop up UI showing any errors that have occurred.
- Dynamic basic auth providers now supported, rather than being limited to 2.
- Enhanced connect button to now include a drop down menu for multiple connection options. (HTTP/Port)
- Added circular progress wheel for resources data volume resources whose status are `CloneInProgress` or `ImportInProgress`. Hovering over the progress wheel will show the current percentage and the resource name.

Kuiper (v4.0.0-beta.2)

- Major backend rewrite, providing greater resource support including KubeVirt virtual machines as well as optimization and stability upgrades.

Terra (v2.1.0)

- Added LOGLEVEL support. Users can now set the log level via the Genesis env. example: `LOGLEVEL: DEBUG` will enable the debug logger.
- Added editable field support. Allowing plugin authors to mark a plugin as editable, doing so will provide the ability to edit the plugin via the Genesis front end UI.
- Added compatibility field support. Allowing plugin authors to mark the supported Orion and Genesis deployment version.

Titan (v2.1.1)

- Added LOGLEVEL support. Users can now set the log level via the Genesis env. example: `LOGLEVEL: DEBUG` will enable the debug logger.

Bug Fixes

Genesis (v4.0.0-beta.1)

- Fixed a bug that may cause an error loading the workloads table if a workload schema failed to load due to an invalid syntax.
- Ensure users can no longer delete a project from the UI if there are currently active running workloads.
- Ensure a workload template name is not in use prior to creation
- Ensure tooltips are no longer interactive. The interactive tool tips could sometimes block the users ability to click other parts of the front end
- Ensure no active workloads are currently running in a storage mounts namespace before deleting a storage mount. Previously deleting a storage mount with active workloads could cause errors, and an incomplete deletion.
- Ensure a user can no longer delete a project group, if that project currently exists.
- Added a default icon for dashboards in the sidebar if there is a connection issue when gathering the dashboards icons.
- Fixed UI issue with our workload schemas default values for our text input fields.

Hubble (v5.0.0-beta.1)

- Ensure tooltips are no longer interactive. The interactive tool tips could sometimes block the users ability to click other parts of the front end

Terra (v2.1.0)

- Fixed a bug with bundle installs, where an install would not throw an error if one of the plugins in the bundle failed.

Maintenance Fixes

Genesis (v4.0.0-beta.1)

- Workload schema default fields adjusted. registry, repo, gpu, and tag will no longer automatically be included in a workload template schema. Instead these must be explicitly set by the Terra plugin author.
- Updated workload idx to match workload label
- Adjusted licensing handling to ensure new kuiper VM workloads are accounted for
- OrionPY upgraded to v1.1.1
- Ensure Terra plugin install form field order matches what the authored plugins field order

Hubble (v5.0.0-beta.1)

- OrionPY upgraded to v1.1.1
- General optimization for api calls, providing better performance and stability
- Utilizing `GENESIS_NAMESPACE` env variable when making api calls to Genesis,Terra, and Titan. To ensure proper namespace handling.

Terra (v2.1.0)

- OrionPY upgraded to v1.1.1

Titan (v2.1.1)

- OrionPY upgraded to v1.1.1

9.1.10 2026-02-07

Components updated in this release: - **Genesis Management Layer**: v2.0.3 - https://github.com/juno-fx/Genesis-Deployment/tree/v2.0.3 - **Orion Projects**: v2.0.1 - https://github.com/juno-fx/Orion-Deployment/tree/v2.0.1

Bug Fixes

Genesis (v3.0.3)

- Fixed a bug that may cause an error loading workload schemas if an input fields string value was extremely long

Hubble (v4.0.1)

- Fixed the development tab to ensure users can run the API endpoints without an error from the backend service to service auth

Kuiper (v3.0.1)

- Fixed a bug that could appear when deleting resources (shutting down workloads)

9.1.11 2026-01-23

Components updated in this release: - **Genesis Management Layer**: v2.0.2 - https://github.com/juno-fx/Genesis-Deployment/tree/v2.0.2

Bug Fixes

Genesis (v3.0.2)

- Fixed a bug within our workloads table, causing the table to error out when the table was empty

Maintenance Fixes

- Fixed typos in the Terra app store alert banner

9.1.12 2026-01-09

Components updated in this release: - **Genesis Management Layer**: v2.0.1 - https://github.com/juno-fx/Genesis-Deployment/tree/v2.0.1

Maintenance Fixes

- Genesis deployment chart registry adjustment for AWS ECR required specs

Bug Fixes

Genesis (v3.0.1)

- Fixed an incorrect warning in our admin controls when attempting to set the targetRevision to v2.0.0

9.1.13 2026-01-08

Components updated in this release: - **Genesis Management Layer**: v2.0.0 - https://github.com/juno-fx/Genesis-Deployment/tree/v2.0.0 - **Orion Projects**: v2.0.0 - https://github.com/juno-fx/Orion-Deployment/tree/v2.0.0

Enhancements

Rhea (v1.0.0)

- Initial release of our backend service to service authorization handler. Injected as a direct part of a pod

Genesis (v3.0.0)

- Admin settings page refresh. Now includes ability to update targetRevision, and adjust authorization providers
- Instances table refresh. Renamed to Workloads
- Workloads table is now a single table, rather than split between workstations and workloads
- Workload templates in the workload table are grouped by workload type
- Workstation endpoints removed, in favor of workload endpoints
- Rhea backend authorization service implemented, ensuring authorization of service to service communication
- Added `list` field type support to workload schema
- Added `multi-line` field type support to workload schemas
- Terra app store filters adjusted. Workstation and Workloads filter now consolidated into Workloads

Hubble (v4.0.0)

- Instances table refresh. Renamed to Workloads
- Workload table has been consolidated into a single tab for Workloads. Removing the workstation differentiation
- Rhea backend authorization service implemented, ensuring authorization of service to service communication

Terra (v2.0.0)

- Rhea backend authorization service implemented, ensuring authorization of service to service communication

Kuiper (v3.0.0)

- Rhea backend authorization service implemented, ensuring authorization of service to service communication

Titan (v2.0.0)

- Rhea backend authorization service implemented, ensuring authorization of service to service communication

Bug Fixes

Genesis (v3.0.0)

- Ensure kubectl is installed correctly based on arch platform

Maintenance Fixes

Kuiper (v3.0.0)

- Updated backend call to Genesis catalog to account for new workload endpoint

---

9.1.14 2025-12-19

Components updated in this release: - **Genesis Management Layer**: v1.7.3 - https://github.com/juno-fx/Genesis-Deployment/tree/v1.7.3 - **Orion Projects**: v1.6.3 - https://github.com/juno-fx/Orion-Deployment/tree/v1.6.3

Maintenance Fixes

Hubble (v3.2.5)

- General internal maintenance

Genesis (v2.3.6)

- General internal maintenance

Bug Fixes

Genesis (v2.3.6)

- Fixed bug, that would cause workload templates to sometimes append multiple "Gi" values to the end of the memory and memoryLimit fields
- Fixed bug, when editing a workload template that has a name of only digits
- Fixed bug, when activating a project that would sometimes cause an error in the Genesis project table while the project is scaling up
- Fixed bug, of an invalid prop key that could pop up when using the sideBar help steppers

Hubble (v3.2.4)

- Fixed bug, that would occur when searching for a workload template via the catalog, when an item in the catalog had a name of only digits.

Security Fixes

Genesis (v2.3.6)

- Upgraded next and react versions in response to security vulnerabilities

Hubble (v3.2.4)

- Upgraded next and react versions in response to security vulnerabilities

---

9.1.15 2025-12-10

Components updated in this release: - **Genesis Management Layer**: v1.7.2 - https://github.com/juno-fx/Genesis-Deployment/tree/v1.7.2

Bug Fixes

Genesis (v2.3.3)

- Fixed bug, that was blocking users from hibernating projects from Genesis frontend UI

Genesis (v2.3.4)

- Fixed bug, where project would disappear from projects table briefly when activating a project.

9.1.16 2025-12-05

Components updated in this release: - **Genesis Management Layer**: v1.7.1 - https://github.com/juno-fx/Genesis-Deployment/tree/v1.7.1 - **Orion Projects**: v1.6.2 - https://github.com/juno-fx/Orion-Deployment/tree/v1.6.2

Security Fixes

Hubble (v3.2.3)

- Upgraded react versions in response to security vulnerabilities

Hubble (v3.2.2)

- Upgraded nextjs versions in response to security vulnerabilities
- security vulnerability fix that previously could reveal a limited set of information

Genesis (v2.3.2)

- security vulnerability fix that previously could reveal a limited set of information

9.1.17 2025-11-17

Components updated in this release:

- **Orion Projects**: v1.6.1 - https://github.com/juno-fx/Orion-Deployment/tree/v1.6.1

Bug Fixes

Hubble (v3.2.1)

- Fixed bug ensuring catalog filters can handle both integer and float tag values.

9.1.18 2025-11-11

Components updated in this release:

- **Genesis Management Layer**: v1.7.0 - https://github.com/juno-fx/Genesis-Deployment/tree/v1.7.0
- **Orion Projects**: v1.6.0 - https://github.com/juno-fx/Orion-Deployment/tree/v1.6.0

Enhancements

Genesis (v2.3.0)

- Add frontend user and group ID validation to ensure system level IDs are not used
- Endpoint added to return Genesis URL
- Basic Auth login password field now, has visibility toggle. Added alert if credentials are invalid

Hubble (v3.2.0)

- Added instance chip number in instance table tabs. Showing how many active workstations/workloads the current user has running
- Added GPU information to template details
- Basic Auth login password field now, has visibility toggle. Added alert if credentials are invalid

Titan (v1.1.0)

- Add backend user and group ID validation to ensure system level IDs are not used

Maintenance

Genesis (v2.3.0)

- When updating project configuration, the frontend form will now say updating rather than creating
- Now preventing empty environment variables being passed in during workload template creation

Bug Fixes

Genesis (v2.3.1)

- Ensure when fetching Orion projects, we now use full argo api group when referencing applications to avoid conflicts.

9.1.19 2025-10-25

Components updated in this release:

- **Genesis Management Layer**: v1.6.0 - https://github.com/juno-fx/Genesis-Deployment/tree/v1.6.0

Bug Fixes

Genesis (v2.2.2)

- Frontend bugfix, when editing project host field. Frontend UI would override with the default if edited.
- Frontend bugfix, workload schemas boolean fields with a default value of True, frontend UI would override with default value if editing template, and field was set to False.

Maintenance

Genesis (v2.2.2)

- Adjusted schema protected fields warning logger to debug

9.1.20 2025-10-17

Components updated in this release:

- **Orion Projects**: v1.5.1 - https://github.com/juno-fx/Orion-Deployment/tree/v1.5.1

Bug Fixes

Hubble (v3.1.1)

- Instance Catalog filter tree bugfixes
- Ensure instance Catalog can support old CRD's with missing fields

---

9.1.21 2025-10-10

Components updated in this release:

- **Genesis Management Layer**: v1.5.0 - https://github.com/juno-fx/Genesis-Deployment/tree/v1.5.0
- **Orion Projects**: v1.5.0 - https://github.com/juno-fx/Orion-Deployment/tree/v1.5.0

Enhancements

Genesis (v2.2.0)

- Admin settings page accessible via user settings drop down now available. With ability to change Genesis,Terra, Titan service version as well as restart deployments.
- Kuiper workload schemas and templates now supported and integrated into instances table
- Kuiper workload schema fields now support default values
- Kuiper workload schema now support additional field types. See our template creation docs for more details: Instances Docs
- Project description and icon customization added
- Project create form will now default the owner selection to the current user, while still maintaining ability to change owner.
- Terra app store installs table now includes plugin icon, pagination, and is grouped by namespace
- Terra app store installs table namespace filter dropped in favor of built in table search field
- Terra app store plugin install form now allows install names to be Alphanumeric characters with optional dashes inbetween.
- Terra app store plugin install form UI refresh
- Terra app store plugin install form now supports default field values
- Terra app store plugin install form now supports additional field types. See our docs for more details: Terra Docs
- Terra app store sources table now includes a quick access button to add the official repo to Terra sources
- Help steppers added to sidebar navigation

Terra (v1.1.0)

- Now passing image details to plugin install
- Now returning sources on refresh endpoint
- Now returning list of installed plugins during bundle install

Kuiper (v2.1.0)

- Multi container pod logs support added, along with additional endpoints for pods

Hubble (v3.1.0)

- Workstation/Workload instances table and request catalog UI refresh
- Multi container log support added
- Help steppers added to instances table
- Ability to create workload templates from a running instance
- Ability to set workload name upon request
- Ability to add environment variables to workload upon request
- Project details widget added to home page

Bug Fixes

Genesis (v2.2.1) - Fixed license expiration warning banner placement

Genesis (v2.2.0)

- workload empty value bugfix
- Storage creation bugfixes ensuring proper mapping between PV&PVC for ReadWriteMany and ReadWriteOnce volume types
- Storage creation form validation for PV and PVC names
- Storage creation form UI updates for PV type handling
- Project edit bugfix, now allowing editing of project owner
- API token creation bugfix, ensuring token is now generated.
- Tooltips bugfix. Tooltips previously at times could block users from selecting table elements

Terra (v1.1.0)

- Ensure proper handling of non-string values being passed into install

Kuiper (v2.1.0)

- Ensuring proper namespace is passed in during our instance launch

Hubble (v3.1.0)

- Instances quick menu UI bugfix for elements overlaying eachother
- Tooltips bugfix. Tooltips previously at times could block users from selecting table elements

Maintenance

Genesis (v2.2.0)

- Margins added to top of project edit page
- Adjusted spacing of Terra app store installs table
- Updated dashboard loading, speeding up Genesis load times
- Sorting Terra app store installs table by plugin
- Login form updated to include Genesis logo
- Adjusted user icon padding

Terra (v1.1.0)

- General logger updates

Kuiper (v2.1.0)

- Defaulting our instance pod state to "requesting" if not yet available
- General backend clean up

Hubble (v3.1.0)

- Home widgets sizing adjustment
- Kuiper and Titan icon added to service widget
- Instance actions column sizing adjusted
- Adjust sidebar docs icon
- Adjusted kuiper quick menu heading

9.1.22 2025-08-15

Components updated in this release:

- **Genesis Management Layer**: v1.4.0 - https://github.com/juno-fx/Genesis-Deployment/tree/v1.4.0
- **Orion Projects**: v1.4.0 - https://github.com/juno-fx/Orion-Deployment/tree/v1.4.0

Enhancements

Genesis (v2.1.0)

- K3s node provisioning and Ansible credential creation directly in the networking page
- Cluster-level Terra iFrame plugin dashboard support
- Refreshed Terra App Store UI
- Dropped restriction forcing hibernation before deleting storage
- User UID editing now allowed
- Storage validation on front-end forms and backend API
- Updated license page context
- Added ingress auth handler
- Improved page and table loading states
- Updated Terra source create forms autofill settings
- Help steppers added to Workstations, Users/Groups, and Networking pages
- HTTP scheme selection for project creation

Terra (v1.0.4)

- Mixed namespace support - bundles can now include both cluster-level and project-level plugins
- Updated plugin field backend schemas for better front-end handling

Titan (v1.0.3)

- User UID editing enabled

Hubble (v3.0.0)

- Added favicon to browser tab title
- Removed Legacy Luna project management support
- Updated error log handling

Bug Fixes

Genesis (v2.1.0)

- Fixed empty sub-path key-value being passed to Terra backend during install
- Fixed Juno logo button box sizing
- Fixed admin group user updates being blocked from group table
- Fixed license banner warning sizing/placement

Titan (v1.0.3)

- Added Titan owner validation on creation to ensure valid user
- Fixed admin group user updates being blocked from group backend

Security Updates

Genesis (v2.1.0) & Hubble (v3.0.0)

- Improved session secret generation.

9.2 Orion Technical Changelog

This technical changelog provides guidance for technical teams operating Orion deployments. It is intentionally kept brief and only informs you of any deprecations, migration steps between major versions, or addressed security vulnerabilities.

This page tracks the releases of Genesis, Orion's management layer, as well as the per-environment Project Deployments. **We recommend you link this changelog within your internal documentation and review it prior to each upgrade.**

If a release has nothing extra to consider, it is explicitly listed as such for you to provide confirmation. Any major release requiring special upgrade steps also contains a detailed migration guide linked under the changelog.

For new features and enhancements, see the Feature Changelog.

9.2.1 2026-06-01 Genesis v4.0.1, Orion Projects v4.0.1

Components updated in this release:

- **Genesis Management Layer**: v4.0.1 - https://github.com/juno-fx/Genesis-Deployment/tree/v4.0.1
- **Orion Projects**: v4.0.1 - https://github.com/juno-fx/Orion-Deployment/tree/v4.0.1

Upgrade Status: Standard upgrade - Security Fixes

Final QA tests performed against: Kubernetes 1.34.4

As part of our internal security improvement review, we identified vulnerabilities from upstream dependencies. We have reviewed our existing public deployments and found no evidence of it being actively exploited.

9.2.2 2026-05-11 Genesis v4.0.0, Orion Projects v4.0.0

- **Genesis Management Layer**: v4.0.0 - https://github.com/juno-fx/Genesis-Deployment/tree/v4.0.0
- **Orion Projects**: v4.0.0 - https://github.com/juno-fx/Orion-Deployment/tree/v4.0.0

Upgrade Status: ⚠ Major upgrade - API Changes, Security Fixes

Final QA tests performed against: Kubernetes 1.34.4

This release contains API changes, with better backend security via our Rhea service. This release will now handle backend user authorization via Rhea. Previously Rhea was only authorizing backend service to service communication.

As part of our internal security improvement review, we identified vulnerabilities from upstream dependencies. We have reviewed our existing public deployments and found no evidence of it being actively exploited.

9.2.3 2026-05-04 Orion Projects v3.1.3

- **Orion Projects**: v3.1.3 - https://github.com/juno-fx/Orion-Deployment/tree/v3.1.3

Upgrade Status: Standard upgrade - no special steps required

Final QA tests performed against: Kubernetes 1.34.4

This release contains no breaking changes, API deprecations or major security updates.

9.2.4 2026-04-23 Orion Projects v3.1.2

- **Orion Projects**: v3.1.2 - https://github.com/juno-fx/Orion-Deployment/tree/v3.1.2

Upgrade Status: Standard upgrade - no special steps required

Final QA tests performed against: Kubernetes 1.34.4

This release contains no breaking changes, API deprecations or major security updates.

9.2.5 2026-04-21 Orion Projects v3.1.1

- **Orion Projects**: v3.1.1 - https://github.com/juno-fx/Orion-Deployment/tree/v3.1.1

Upgrade Status: Standard upgrade - no special steps required

Final QA tests performed against: Kubernetes 1.34.4

This release contains no breaking changes, API deprecations or major security updates.

9.2.6 2026-04-14 Genesis v3.0.2, Orion Projects v3.1.0

- **Genesis Management Layer**: v3.0.2 - https://github.com/juno-fx/Genesis-Deployment/tree/v3.0.2
- **Orion Projects**: v3.1.0 - https://github.com/juno-fx/Orion-Deployment/tree/v3.1.0

Upgrade Status: Standard upgrade - no special steps required

Final QA tests performed against: Kubernetes 1.34.4

This release contains no breaking changes, API deprecations.

We recommend promptly updating existing deployments. As part of our internal security improvement review, we identified vulnerabilities from upstream dependencies. We have reviewed our existing public deployments and found no evidence of it being actively exploited.

9.2.7 2026-04-10 Genesis v3.0.1, Orion Projects v3.0.1

- **Genesis Management Layer**: v3.0.1 - https://github.com/juno-fx/Genesis-Deployment/tree/v3.0.1
- **Orion Projects**: v3.0.1 - https://github.com/juno-fx/Orion-Deployment/tree/v3.0.1

Upgrade Status: Standard upgrade - no special steps required

Final QA tests performed against: Kubernetes 1.34.4

This release contains no breaking changes, API deprecations.

We recommend promptly updating existing deployments. As part of our internal security improvement review, we identified vulnerabilities from upstream dependencies. We have reviewed our existing public deployments and found no evidence of it being actively exploited.

9.2.8 2026-04-03 Genesis v3.0.0, Orion Projects v3.0.0

- **Genesis Management Layer**: v3.0.0 - https://github.com/juno-fx/Genesis-Deployment/tree/v3.0.0
- **Orion Projects**: v3.0.0 - https://github.com/juno-fx/Orion-Deployment/tree/v3.0.0

Upgrade Status: ⚠ Major upgrade - API Changes

Final QA tests performed against: Kubernetes 1.34.1

Unless upgrading from the beta release, this release includes major API changes to Kuiper, along with a host of other features. It is recommended if upgrading to ensure all services are upgraded for each environment.

9.2.9 2026-03-18 Genesis v3.0.0-beta.1, Orion Projects v3.0.0-beta.1

- **Genesis Management Layer**: v3.0.0-beta.1 - https://github.com/juno-fx/Genesis-Deployment/tree/v3.0.0-beta.1
- **Orion Projects**: v3.0.0-beta.1 - https://github.com/juno-fx/Orion-Deployment/tree/v3.0.0-beta.1

Upgrade Status: ⚠ Major upgrade - API Changes

Final QA tests performed against: Kubernetes 1.34.1

This release includes major API changes to Kuiper, along with a host of other features. It is recommended if upgrading to ensure all services are upgraded for each environment.

9.2.10 2026-02-07 Genesis v2.0.3, Orion Projects v2.0.1

- **Genesis Management Layer**: v2.0.3 - https://github.com/juno-fx/Genesis-Deployment/tree/v2.0.3
- **Orion Projects**: v2.0.1 - https://github.com/juno-fx/Orion-Deployment/tree/v2.0.1

Upgrade Status: Standard upgrade - no special steps required

Final QA tests performed against: Kubernetes 1.34.1

This release contains no breaking changes, API deprecations or major security updates.

9.2.11 2026-01-23 Genesis v2.0.2

- **Genesis Management Layer**: v2.0.2 - https://github.com/juno-fx/Genesis-Deployment/tree/v2.0.2

Upgrade Status: Standard upgrade - no special steps required

Final QA tests performed against: Kubernetes 1.34.1

This release contains no breaking changes, API deprecations or major security updates.

9.2.12 2026-01-09 Genesis v2.0.1

- **Genesis Management Layer**: v2.0.1 - https://github.com/juno-fx/Genesis-Deployment/tree/v2.0.1

Upgrade Status: Standard upgrade - no special steps required **when upgrading from v2.0.0**

Final QA tests performed against: Kubernetes 1.34.1

This release contains no breaking changes, API deprecations or major security updates. The only adjustments were to the Genesis deployment charts registry handling in an effort to meet AWS ECR spec requirements, along with a fix for an incorrect warning in our admin controls panel.

Please note v2.0.0 was a major update, please see the v2.0.0 changelogs for fuller detailed release notes.

9.2.13 2026-01-08 Genesis v2.0.0, Orion Projects v2.0.0

- **Genesis Management Layer**: v2.0.0 - https://github.com/juno-fx/Genesis-Deployment/tree/v2.0.0

- **Orion Projects**: v2.0.0 - https://github.com/juno-fx/Orion-Deployment/tree/v2.0.0

Upgrade Status: ⚠ Major upgrade - API Changes, Terra Plugin Support Affected.

Upgrading to Genesis v2.0.0 introduces our Rhea backend service to service authorization handler. This will require Orion Projects to also be upgraded to v2.0.0 in order for the backend requests to be authorized and function properly.

Final QA tests performed against: Kubernetes 1.34.1

- Rhea our backend service to service authorization handler has been implemented. Currently Terra plugins that try to interact directly with our API will not be supported. This includes plugins such as `Helios-Auto-Shutdown` this support will be added in future releases
- We have changed how our Genesis, Terra, and Titan images are formatted in our charts. Prior to upgrading to this targetRevision, please ensure any custom overrides to these image versions is cleared out for a seamless transition.
- The Genesis `/workstation` endpoints have been removed, and replaced with the `/workload` endpoint
- The Genesis `/schemas` and `/catalog` endpoints have been replaced with `/workload/schemas` and `/workload/catalog`
- Rhea our backend service to service authorization handler has been implemented. Currently Terra plugins that try to interact directly with our API will not be supported. This includes plugins such as `Helios-Auto-Shutdown` this support will be added in future releases
- Free Community license now allows for 2 active workloads rather than 5

9.2.14 2025-12-19 Genesis v1.7.3, Orion Projects v1.6.3

- **Genesis Management Layer**: v1.7.3 - https://github.com/juno-fx/Genesis-Deployment/tree/v1.7.3
- **Orion Projects**: v1.6.3 - https://github.com/juno-fx/Orion-Deployment/tree/v1.6.3

Upgrade Status: Standard upgrade - no special steps required

Final QA tests performed against: Kubernetes 1.34.1

This release contains no breaking changes, API deprecations.

We recommend promptly updating existing deployments. As part of our internal security improvement review, we identified vulnerabilities from upstream dependencies in Next and React. We have reviewed our existing public deployments and found no evidence of it being actively exploited. To prevent making this more likely, we will contact customers directly with specific information on the vulnerability.

9.2.15 2025-12-10 Genesis v1.7.2

- **Genesis Management Layer**: v1.7.2 - https://github.com/juno-fx/Genesis-Deployment/tree/v1.7.2

Upgrade Status: Standard upgrade - no special steps required

Final QA tests performed against: Kubernetes 1.34.1

This release contains no breaking changes, API deprecations or major security updates.

9.2.16 2025-12-05 Genesis v1.7.1, Orion Projects v1.6.2

- **Genesis Management Layer**: v1.7.1 - https://github.com/juno-fx/Genesis-Deployment/tree/v1.7.1
- **Orion Projects**: v1.6.1 - https://github.com/juno-fx/Orion-Deployment/tree/v1.6.2

Upgrade Status: Standard upgrade - no special steps required

Final QA tests performed against: Kubernetes 1.34.1

This release contains no breaking changes, API deprecations.

We recommend promptly updating existing deployments. As part of our internal security improvement review, we identified a component that could be exploited to reveal a limited set of information. We have reviewed our existing public deployments and found no evidence of it being actively exploited. To prevent making this more likely, we will contact customers directly with specific information on the vulnerability.

9.2.17 2025-11-17 Orion Projects v1.6.1

- **Orion Projects**: v1.6.1 - https://github.com/juno-fx/Orion-Deployment/tree/v1.6.1

Upgrade Status: Standard upgrade - no special steps required

Final QA tests performed against: Kubernetes 1.34.1

This release contains no breaking changes, API deprecations or major security updates.

9.2.18 2025-11-11 Genesis v1.7.0, Orion Projects v1.6.0

- **Genesis Management Layer**: v1.7.0 - https://github.com/juno-fx/Genesis-Deployment/tree/v1.7.0
- **Orion Projects**: v1.6.0 - https://github.com/juno-fx/Orion-Deployment/tree/v1.6.0

Upgrade Status: Standard upgrade - no special steps required

Final QA tests performed against: Kubernetes 1.34.1

This release contains no breaking changes, API deprecations or major security updates.

9.2.19 2025-10-25 Genesis v1.6.0

- **Genesis Management Layer**: v1.6.0 - https://github.com/juno-fx/Genesis-Deployment/tree/v1.6.0

Upgrade Status: Standard upgrade - no special steps required

Final QA tests performed against: Kubernetes 1.34.1

This release contains no breaking changes, API deprecations or major security updates.

9.2.20 2025-10-17 Orion Projects v1.5.1

- **Orion Projects**: v1.5.1 - https://github.com/juno-fx/Orion-Deployment/tree/v1.5.1

Upgrade Status: Standard upgrade - no special steps required

Final QA tests performed against: Kubernetes 1.33.3

This release contains no breaking changes, API deprecations or major security updates.

9.2.21 2025-10-10 Genesis v1.5.0, Orion Projects v1.5.0

- **Genesis Management Layer**: v1.5.0 - https://github.com/juno-fx/Genesis-Deployment/tree/v1.5.0
- **Orion Projects**: v1.5.0 - https://github.com/juno-fx/Orion-Deployment/tree/v1.5.0

Upgrade Status: Standard upgrade - no special steps required

Final QA tests performed against: Kubernetes 1.33.3

This release contains no breaking changes, API deprecations or major security updates.

9.2.22 2025-08-15 Genesis v1.4.0, Orion Projects v1.4.0

- **Genesis Management Layer**: v1.4.0 - https://github.com/juno-fx/Genesis-Deployment/tree/v1.4.0
- **Orion Projects**: v1.4.0 - https://github.com/juno-fx/Orion-Deployment/tree/v1.4.0

Upgrade Status: Standard upgrade - no special steps required

Final QA tests performed against: Kubernetes 1.33.3

This release contains no breaking changes or API deprecations.

It contains improvements to backend token and signing generation across both the per-project and cluster management layers. We recommend upgrading any 1.3.X or earlier deployments to it.