



# Virtual Network Orchestrator (ViNO) User Guide

Software Version 1.0.1

February 4, 2020

©2020 CenturyLink. All Rights Reserved. The CenturyLink mark, pathways logo and certain CenturyLink product names are the property of CenturyLink. All other marks are the property of their respective owners.

# Contents

Introduction.....	1
ViNO Software Components .....	2
ViNO Keycloak Roles and Groups .....	3
ViNO Homepage .....	4
Accessing Keycloak Account Settings .....	5
Service Activations and Service Detail Sections .....	5
Displaying Service Details .....	8
Deactivating a Service .....	9
Reactivating a Service.....	9
Displaying and Hiding Service Entries.....	9
Activating a Service .....	10
Application Settings Menu—Settings Management.....	13
Using Settings Categories .....	15
Adding a Settings Category .....	15
Modifying the Display Name in a Settings Category.....	15
Using Groups .....	16
Adding a Group .....	16
Modifying the Display Name for a Group .....	17
Using Scalars .....	17
Adding a Scalar.....	18
Modifying a Scalar.....	19
Using Scalar Lists .....	19
Adding a Scalar List .....	19
Modifying the Display Name for a Scalar List .....	20
Downloading and Uploading a JSON File .....	20
Downloading a JSON File .....	21
Uploading a JSON File .....	21
Deleting Elements .....	22
Application Settings—User Management .....	22
Service Manager.....	22
Service Manager Components .....	24
Service Manager Concepts .....	26
Creating or Cloning a Project.....	27
Creating a Project.....	28
Cloning a Project Repository.....	28
Saving a Project .....	29
Working in the Service Designer .....	29
Working in the Palette.....	30
Managing Nodes in the Palette Manager.....	31
Accessing the Palette Manager.....	31
Managing Nodes .....	32

Adding Nodes to the Palette .....	33
Using Built-In or Third-Party Node-RED Nodes.....	33
Working with Flow Tabs .....	34
Adding a Flow Tab .....	35
Editing Flow Tab Properties .....	35
Enabling or Disabling a Flow Tab .....	35
Deleting a Flow Tab.....	35
Switching Between Flows.....	35
Working with Nodes .....	36
Using the Inject Node.....	37
Using the Debug Node .....	37
Using the Function Node.....	37
Using the Change Node.....	38
Using the Switch Node .....	38
Using the Template Node .....	38
Using ViNO Nodes .....	39
Using Conditional Start and Conditional End Nodes.....	42
Adding Status Messages to Built-In Nodes .....	43
Using the Function Node to Manipulate Parameters .....	44
Working with Wires .....	44
Moving Wires .....	45
Deleting Wires.....	45
Working with Subflows.....	45
Designing a Subflow .....	47
Working with Messages.....	48
Changing Message Properties.....	48
Using the Service Status Node .....	48
Working with the Sidebar.....	49
Using Parameters.....	50
Configuring Parameters on Nodes .....	51
Editing Parameters.....	56
Using Parameter Mapping .....	57
Using Parameter Mappings and Conditionals.....	57
Using Parameter Wrapper Injection and Extraction.....	58
Using Parameter Combiner Modes.....	58
Passing Parameters Between a Parent Flow and Subflow.....	59
Using Loops.....	61
Using Basic Loops .....	61
Considerations for Each Loop .....	62
Using Input Parameters in a Loop Step.....	63
Using Parameter Mapping and Loops.....	63
Using Parallel Execution .....	64
Joining Parallel Branches.....	64
Creating a Service Deactivation Flow .....	66

Error Handling .....	68
Using the Catch Node .....	68
Using the Throw Node.....	68
Adding Rollback Capability.....	69

## List of Tables

Table 1. ViNO Components.....	2
Table 2. ViNO Keycloak Roles and Permissions .....	3
Table 3. ViNO Homepage Navigation Bar .....	4
Table 4. Service Activations Section.....	7
Table 5. Service Details Field Descriptions.....	8
Table 6. Service Activation Screen .....	10
Table 7. Settings Management Screen .....	14
Table 8. New Scalar Field Descriptions .....	18
Table 9. Service Manager Components .....	24
Table 10. Service Manager Concepts.....	26
Table 11. ViNO Nodes .....	39
Table 12. Property Edit Window .....	52

## List of Figures

Figure 1. ViNO Homepage.....	6
Figure 2. Service Activation Screen.....	11
Figure 3. Service Activation Details Screen.....	12
Figure 4. Settings Management Screen .....	14
Figure 5. JSON Upload and Download Buttons.....	20
Figure 6. Service Manager Screen.....	23
Figure 7. Service Manager Screen in a New ViNO .....	27
Figure 8. Filter Node Field and Toggle Palette .....	30
Figure 9. User Settings Window .....	32
Figure 10. User Settings—Nodes List in Palette Tab .....	32
Figure 11. Palette Install Tab.....	33
Figure 12. Service Manager Flow Tab Navigation Bar .....	34
Figure 13. Edit Flow Sidebar.....	34
Figure 14. Example of Using the Conditional Start Node.....	42
Figure 15. Example of Using the Conditional End Node .....	43
Figure 16. Example of Using Conditional Start and Conditional End Nodes.....	43
Figure 17. Working with Wires .....	44
Figure 18. Dropping a Node on a Wire .....	45
Figure 19. Create Subflow Option.....	46
Figure 20. Example of New Subflow and Toolbar .....	46

Figure 21. Subflow Example .....	47
Figure 22. Service Manager Sidebar.....	49
Figure 23. Property Edit Window .....	51
Figure 24. Edit Input Parameter Window .....	56
Figure 25. Mode Field in Property Edit Window .....	58
Figure 26. Using a Parameter Wrapper Node .....	60
Figure 27. Example of a Basic Loop .....	61
Figure 28. Using Input List in a Loop .....	62
Figure 29. Loop Start Node Example.....	62
Figure 30. Example of Creating a Parallel Branch .....	64
Figure 31. Example of Joining Parallel Branches .....	64
Figure 32. Example of Join Node Properties .....	65
Figure 33. Service with Corresponding Deactivation Flow .....	66
Figure 34. Deactivation Flow Example .....	67
Figure 35. Implementing a Rollback.....	69

# Introduction

This document describes how to use the Virtual Network Orchestrator (ViNO) to develop services and flows in order to:

- Create, activate, deactivate, and manage virtual services on OpenStack, NFVi platforms, and servers
- Manage virtual service network interfaces
- Use ViNO nodes to interact with OpenStack controllers in order to manipulate Virtual Machines (VMs)
- Interact with servers using robust Ansible support

In addition to this guide, ViNO documentation includes the *Virtual Network Orchestrator (ViNO) Installation Guide*, which describes how to install and initialize a ViNO instance and how to create a realm in Keycloak. ViNO uses [Keycloak](#) for authentication.

ViNO enables you to create, activate, deactivate, and manage virtual services for:

- Servers
- Firewalls
- Routers
- SD WANs

ViNO is built on [Node-RED](#), which is a browser-based editor for wiring together flows using a wide range of nodes in the palette that can be deployed to its runtime in a single click.

**Note:** The Node-RED site contains detailed information and user documentation on using the product. The expectation is for ViNO users to use the [Node-RED documentation](#) to familiarize themselves with the product functionality before using ViNO to create service flows.

ViNO builds on the flow-based model provided in Node-RED by including the following components specific to creating, configuring, and delivering services:

- VNF-specific nodes that enable you to define how data is passed between the steps within in a flow.
- A backing web service API that enables new nodes to provide interfaces to complex network management actions.
- A management user interface.
- An authentication abstraction.
- Drivers that provide access to the Netconf protocol, Ansible, and the OpenStack V3 API.
- REST APIs that provide a unified and consistent interface for managing the full life cycle of service activation, deactivation, and management.

The additional nodes included in the software enhance the functionality of Node-RED by enabling ViNO to programmatically and automatically generate activation templates that have full compatibility with existing Node-RED default nodes, as well as many of the community-contributed nodes available on the internet.

**Note:** The difference between a ViNO service and a Node-RED flow is that a ViNO service must always start with a **service endpoint** node and end with a **service endpoint** node.

# ViNO Software Components

The ViNO software is comprised of several components as described in Table 1.

**Table 1. ViNO Components**

Component	Description
<b>NGINX</b>	A lightweight, high performance HTTP proxy, load balancer, and web host. In the ViNO software suite, NGINX serves the role of an HTTP reverse proxy to expose select services inside a local private network and access them from the Docker host system.
<b>Service Manager</b>	A combination of an unmodified Node-RED installation and a custom user interface on top. The Node-RED instance has been branded for CenturyLink and includes several <a href="#">custom</a> nodes that provide the ability to create service flows that can be activated programmatically and registered with an API instance.
<b>ViNO API</b>	An Express.js-based REST interface that provides service registration, activation, history, and status monitoring as well as hosts several protocol drivers that enable the Service Manager to interact with remote devices using Netconf, Ansible, and Openstack APIs.
<b>Settings Server</b>	Provides a hierarchical data store that enables users to define groups of settings and variables.
<b>Keycloak Authentication Server</b>	<p>ViNO uses the <a href="#">Keycloak</a> authentication server to provide local authentication and the ability to incorporate federated authentication methods such as Kerberos or LDAP. In addition, ViNO can use Single Sign On (SSO) Identity Providers (IdPs) over SAML or OpenID Connect (OIDC).</p> <p>This component alleviates the need to log into each individual piece of the ViNO software suite. It also simplifies the process of integrating the open source projects that comprise ViNO into the corporate authentication mechanisms.</p>
<b>PostgreSQL Database</b>	ViNO uses a PostgreSQL database to store various pieces of information such as the configuration for authentication and the history of activations including service input and output.



# ViNO Keycloak Roles and Groups

ViNO Keycloak roles each have unique permissions that apply to users assigned a role or multiple roles.

- **Roles** are permission-centric. Roles apply permissions to users.
- **Groups** are user-centric. They are a collection of users. Use groups to manage users.

Menu options differ depending on the role you are assigned. For example, a user assigned only the operator role cannot view the **Service Manager**, **Activate a Service**, or **Application Settings** menus.

ViNO provides four roles that correspond to different permissions. Table 2 defines the roles and their permissions.

**Table 2. ViNO Keycloak Roles and Permissions**

Keycloak Role	Permissions
administrator	<ul style="list-style-type: none"><li>▪ Change settings in Settings Management.</li><li>▪ View service details on the homepage.</li></ul>
designer	<ul style="list-style-type: none"><li>▪ Create and modify flows in Service Manager.</li><li>▪ Activate services.</li><li>▪ View Settings Management, but cannot edit the screen.</li><li>▪ View service details on the homepage.</li></ul>
provisioner	<ul style="list-style-type: none"><li>▪ Activate and deactivate services.</li><li>▪ View Settings Management, but cannot edit the screen.</li><li>▪ View service details on the homepage.</li></ul>
operator	View service details on the homepage. This role is read-only.
user	<p>Required role that provides basic access permissions to ViNO.</p> <p><b>Note:</b> All users must be assigned this role in addition to at least one of the other four roles.</p>

# ViNO Homepage

The menu options on the homepage navigation bar differ depending on the role ([administrator](#), [designer](#), [operator](#), or [provisioner](#)) your username is assigned. For example, a user assigned only the **operator** role cannot view the **Service Manager**, **Activate a Service**, and **Application Settings** menus.

Table 3 describes the menus located in the top navigation bar of the homepage. Figure 1 shows an example of the homepage.

**Table 3. ViNO Homepage Navigation Bar**

Menu	Description
<b>Main – Homepage</b>	Displays the <b>Service Activations</b> and the <b>Service Detail</b> sections. Click <b>Main</b> to refresh or return to the homepage.
<b>Service Manager</b>	Displays the flow editing tool ( <b>Service Manager</b> ). Click ViNO-Service Manager to return to the home page.
<b>Activate a Service</b>	Enables you to select a service from the drop-down, enter customer information, and then progress through the configuration steps to activate the service.
<b>Application Settings</b>	Includes the following options: <ul style="list-style-type: none"><li>▪ <b>Settings Management</b> – Enables you to add, remove, or modify settings categories and modify default scalar values and overrides. See <a href="#">Application Settings Menu–Settings Management</a>.</li><li>▪ <b>User Management</b> – Enables you to access the Keycloak Admin console. See <a href="#">Application Settings–User Management</a>.</li></ul>
<b>Help</b>	Includes:  <b>Installed Docker Containers Details</b> – Displays the installed version of each ViNO component: vino-core, vino-database, vino-keycloak, and vino-proxy.  <b>Web Service Documentation</b> – Displays the ViNO REST services in Swagger. If you build software to call ViNO, use this option for a machine-to-machine interface. Use the browser Back button to return to the ViNO home page.
<i>Username</i> (Displays the name of the user who is logged in)	Includes the <b>Account Settings</b> menu and the Log Out button. <ul style="list-style-type: none"><li>▪ <b>Account Settings</b> menu – Provides access to Keycloak. See <a href="#">Accessing Keycloak Account Settings</a>.</li><li>▪ Log Out button – Click to log out of ViNO.</li></ul>

## Accessing Keycloak Account Settings

The **Account Settings** option (in the *username* drop-down) enables you to access the Keycloak account manager. Keycloak displays the **Edit Account** screen by default and pre-populates your username in the **Username** field. The navigation bar at the top of the screen includes:

- **English** – Default language. Click the drop-down to select a different language that applies to the options in the **Account Settings** menu.
- **Back to vino-api** – Returns you to the ViNO homepage.
- **Sign Out** – Logs you out of Keycloak and ViNO.

The **Edit Account** screen includes the following options. Refer to the [Keycloak](#) documentation for information on using these options.

- **Account**
- **Password**
- **Authenticator**
- **Sessions**
- **Applications**

## Service Activations and Service Detail Sections

The homepage contains the following two sections, which are read-only:

- **Service Activations** – Enables a designer or provisioner to deactivate and reactivate services.
- **Service Detail** – Enables all users to view service details and details for the specific steps in a service. See [Displaying Service Details](#).

The **Service Activations** section lists active and deactivated services. To filter the list of entries, enter criteria in the Search field (you do not have to click Enter). Search criteria applies to the data in all columns. Clear the criteria to re-display all entries.

Figure 1 shows an example of the homepage. To display information in the **Service Detail** section for a specific service, highlight the entry in the **Service Activations** section.

Figure 1. ViNO Homepage

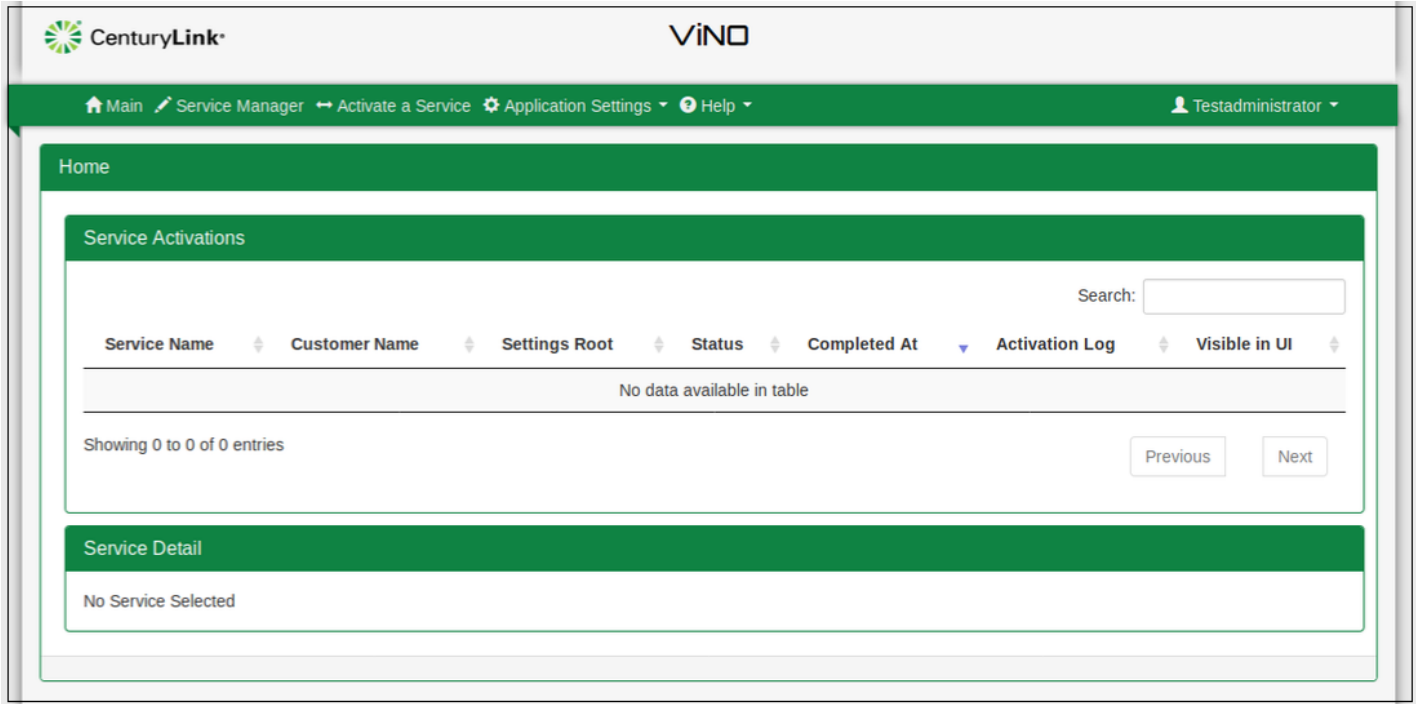


Table 4 describes the buttons and columns in the **Service Activations** section. Each column can be toggled to list the entries in either alphabetical or numerical order (ascending or descending).

**Table 4. Service Activations Section**

<b>Component</b>	<b>Description</b>
<b>Deactivate Button</b>	Deactivates a service. Highlight the service entry in the <b>Services</b> section and click Deactivate. See <a href="#">Deactivating a Service</a> .
<b>Reactivate Button</b>	Reactivates a service. Highlight the service entry in the <b>Services</b> section and click Reactivate. See <a href="#">Reactivating a Service</a> .
<b>Show All Activations</b>  <b>Filter All Activations</b>	Displays all activations (service entries) irrespective of the <b>Visible in UI</b> column setting (Yes or No).  Hides service entries that have the <b>Visible in UI</b> column set to Yes. See <a href="#">Displaying and Hiding Service Entries</a> .
<b>Search Field</b>	Filter the list of entries. Search criteria applies to data in all columns. When the Search field is blank, ViNO displays all entries. When the Search field contains criteria, ViNO displays only those entries that match the search criteria. Clear the criteria to re-display all entries.
<b>Service Name Column</b>	Name of the service as defined in the associated flow.
<b>Customer Name</b>	Name of the customer associated with the service as defined in the associated flow.
<b>Settings Root</b>	Identifies the root group to use for all constants in the service activation.
<b>Status</b>	Status of the service (Activated or Terminated).
<b>Completed At</b>	Date and time stamp when the service activation or deactivation was completed. The time stamp is UTC (Coordinated Universal Time).
<b>Activation Log</b>	Configuration log of a service activation. Click Close to close the log or click Download Log to open and save the file.
<b>Visible in UI Column</b>	Displays (Yes) or hides (No) an entry. The default is Yes. See <a href="#">Displaying and Hiding Service Entries</a> .

## Displaying Service Details

The **Service Detail** section displays information about a service and about each step in the flow. This information is historical and is helpful for troubleshooting. To display information for a service, select the service in the **Services** section.

Table 5 describes the fields in the **Service Detail** section.

**Table 5. Service Details Field Descriptions**

Field	Description
<b>Job ID</b>	Identifier for the service Activation log. ViNO automatically assigns and updates the job ID.
<b>Customer Name</b>	Name of the customer associated with the service as defined in the <b>Customer Info</b> section when you activate a service ( <b>Service Activation</b> screen).
<b>Settings Root</b>	Identifies the root group to use for all constants in the service activation.
<b>Activation Status</b>	Displays the value listed in the <b>Status</b> column ( <b>Service Activations</b> section).
<b>Activation Notes</b>	Notes associated with the service as defined in the <b>Customer Info</b> section when you activate a service ( <b>Service Activation</b> screen).
<b>Select Step for Details</b>	Drop-down that lists each step in a flow as defined by the designer.
<b>Input Parameters</b> and <b>Output Parameters</b>	Select an entry from the <b>Select Step for Details</b> drop-down to display the Input and Output parameters defined by the designer for that specific node. Both parameter sections include the following columns: <ul style="list-style-type: none"><li>▪ <b>Parameter Name</b> – Name of the parameter defined in the flow.</li><li>▪ <b>Parameter Description</b> – Description of the parameter defined in the flow.</li><li>▪ <b>Parameter Type</b> – Type of parameter (for example, string, Boolean, number, enumeration) defined in the flow.</li><li>▪ <b>Parameter Value</b> – Value of the parameter defined in the flow.</li></ul>

## Deactivating a Service

When you deactivate a service, ViNO tears down the service configuration, but retains the configuration information enabling you to easily reactivate the service if needed.

**Note:** A flow must have a deactivation path defined to in order to deactivate the service.

To deactivate a service:

1. Select the service that you want to deactivate.
2. Click Deactivate. ViNO displays a confirmation window.
3. Click OK to proceed with deactivating the service. The Deactivate Service log opens and displays the progress. When the process is complete, the last entry in the log displays: Service deactivation completed successfully.
4. Click OK to close the log. ViNO updates the service status to Terminated.

## Reactivating a Service

To reactivate a service:

1. Select the service that you want to reactivate.
2. Click Reactivate. ViNO displays a confirmation window.
3. Click OK to proceed with reactivating the service. The Activating Service log opens and displays the progress. When the process is complete, the last entry in the log displays: Service activation completed successfully.
4. Click OK to close the log. ViNO adds a new activation entry to the **Service Activations** section.

## Displaying and Hiding Service Entries

The [Visible in UI](#) column (homepage > **Services** section) enables you to display or hide service entries. The default setting is Yes (display an entry). To hide an entry, change the setting to No. To display all entries irrespective of the entry setting, click Show All Activations. To hide entries that are set to No, click the button, which changes to Filter Activations.

## Activating a Service

The **Activate a Service** menu displays the **Service Activation** screen, which enables a user assigned the [administrator](#) or the [provisioner](#) role to select a service to be activated. Table 6 describes the sections and fields in the **Service Activation** screen.

**Table 6. Service Activation Screen**

Section	Description
Service Info	Includes the <b>Service Name</b> and the <b>Description</b> fields, which ViNO populates.
Customer Info	Includes the following fields: <ul style="list-style-type: none"><li>▪ <b>Settings Root</b> – (Required) Select the root group to use for all constants in the service activation.</li><li>▪ <b>Customer Name</b> – Enter the customer name associated with the service.</li><li>▪ <b>Notes</b> – Enter specific information about the service.</li></ul>
Service Steps	Lists the steps in a flow that are required to create a service. Select a node to display its parameters (default, custom, or pre-configured). Each node includes the following fields: <ul style="list-style-type: none"><li>▪ <b>Name</b> – Name of the node as entered by the designer.</li><li>▪ <b>Node ID</b> – System-generated unique identifier.</li><li>▪ <b>Description</b> – Text entered by the designer in the Edit panel of a node.</li></ul>
Parameters	Displays parameters assigned by a designer for each node in a flow. <ul style="list-style-type: none"><li>▪ <b>Show Pre-Configured Parameters</b> – Displays node parameters that have pre-assigned values. To enter or update a parameter, double-click the field. A field can still be updated when it is greyed-out. To cancel an update, refresh the browser (which returns you to the home page).</li><li>▪ <b>Hide Pre-Configured Parameters</b> – Displays parameters (with or without assigned values) that apply to a node and hides parameters displayed by the <b>Show Pre-Configured Parameters</b> setting.</li></ul> <p><b>Note:</b> When a parameter has the <b>Final</b> checkbox selected (located in the <b>Edit Input Parameter</b> window), it is not displayed in the <b>Parameter</b> section in order to protect the integrity of a finalized flow. Parameters that have the <b>Final</b> checkbox selected can only be updated from the <b>Edit Input Parameter</b> window.</p>
Back Button	Displays parameters for the previous service step.
Next Button	Displays parameters for the next service step.



To activate a service:

1. Select a service from the **Service** drop-down (**Activate a Service** menu). ViNO auto-populates the **Service Name** and the **Description** fields with the information that was entered when the flow was created. Figure 2 shows the Create NBS VM service selected with information populated in the **Details** section.

**Note:** ViNO only auto-populates the **Service** drop-down with services that start with a **service entryptoint** node.

**Figure 2. Service Activation Screen**

The screenshot shows the 'Service Activation' screen. At the top, there's a green navigation bar with links: Main, Service Manager, Activate a Service, Application Settings, and Help. The user 'Testadministrator' is logged in. The main area has a green header 'Service Activation'. Below it, there's a 'Service' dropdown menu with 'Core Node Test' selected. Underneath is a 'Details' section with a green header. It contains two text input fields: 'Service Name' (filled with 'Core Node Test') and 'Description' (filled with 'Tests each type of Parameter Wrapper, each type of Loop, Conditionals, and Status Messages.'). A green 'Select Service' button is at the bottom right of the details section.

2. Click Select Service. The **Service Activation Details** screen displays the **Services Info**, **Customer Info**, and the **Service Steps** sections as shown in [Figure 3](#).
3. Select an entry from the **Settings Root** drop-down (**Customer Info** section). The Settings Root identifies the root group to use for all constants in the service activation. ViNO populates the values in this drop-down from the Settings Management entries.
4. (Optional) Enter the customer name in the **Customer Name** field. If you do not enter a name, ViNO auto-populates the value as *ViNO*.
5. (Optional) Enter notes pertaining to the customer or site (for example, site address) in the **Notes** field.
6. Select each step (or click the Next button) to display parameters for each step (**Service Steps** section). The parameters for each step correspond to a node in the flow (that contains values).
7. Populate parameters as needed by toggling through each step. Required fields are highlighted in red text and change to green text when you enter a value. Pre-populated variables appear greyed out in the **Service Activation** screen. To override a default value, double-click the field and confirm that you want to override the value, then enter the new value.
8. After you have toggled through each step, click Next to display the **Review and Activate** screen. This screen displays the service name, description, and the steps including their parameters (see [Figure 3](#)).

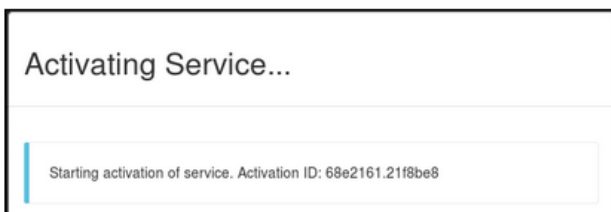
To activate a service without first checking the parameters for each step, select the last step in the flow, and then click Activate. In the **Review and Activate** screen, click Activate. If a step is missing a required parameter, an error message is displayed at the top of the screen.

**Figure 3. Service Activation Details Screen**

- To proceed with activating the service, click Activate. To include ViNO debug messages in the activation log, select the **Run in Debug Mode** checkbox, then click Activate. To make additional changes to parameters, click Edit Service Parameters to return to the **Service Activation** screen.

When you click Activate, ViNO displays the following warning message:

10. Click Activate to activate the service. ViNO displays a status log as shown below.



11. Click OK at the end of the status log to close the window.

## Application Settings Menu—Settings Management

The **Settings Management** screen (**Application Settings** menu) enables you to create, group, and store settings, elements, and scalars that can be used in services. Storing elements can provide variable default values to driver nodes. For example, a service may call for a different VM image or size based on the needs of a customer. The **Settings Management** screen enables you to store a group of settings that defines images and sizes, which can be passed in at service activation time rather than having to manually configure them each time or hard coding them into the flow.

To use constants in a ViNO service, see [Constants](#).

The following sections describe each element in Settings Management:

- [Using Settings Categories](#)
- [Using Groups](#)
- [Using Scalars](#)
- [Using Scalar Lists](#)
- [Downloading and Uploading a JSON File](#)
- [Deleting Elements](#)

**Note:** Only a user assigned the administrator role can edit and save values in the **Settings Management** screen. A user assigned the designer role can only view the **Settings Management** screen. If a designer updates this screen and tries to save the changes, an error message is displayed and the changes are not saved.

A new instance of ViNO does not include categories. You must create them manually or create them by loading the data using a JSON file. Settings management data is stored in JSON files. See [Downloading and Uploading a JSON File](#).

An example of the **Settings Management** screen with three expanded categories is shown in Figure 4. Element names that are preceded with an asterisk ( \* ) are default elements. When you select an element, a window displays information for that element including options to add a group, a scalar list, or a scalar. If the element you select contains additional elements (such as groups, scalars, or scalar lists), the tree expands to display all the elements.

**Figure 4. Settings Management Screen**

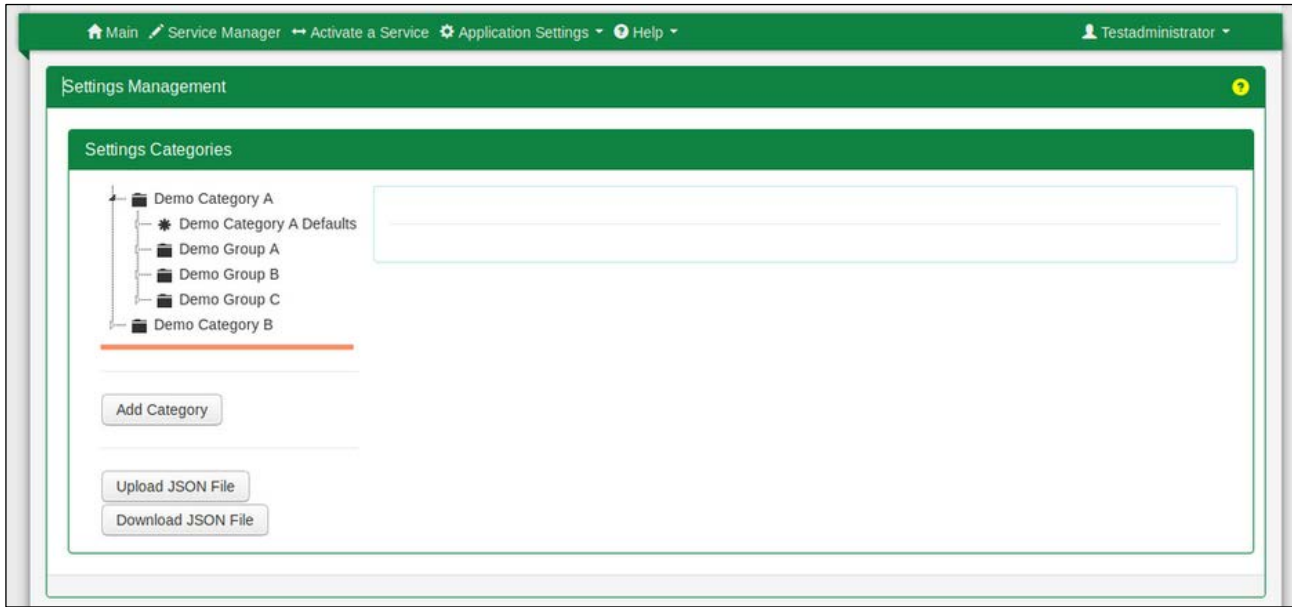


Table 7 describes the section and buttons on the **Settings Management** screen.

**Table 7. Settings Management Screen**

Element	Description
<b>Settings Categories</b>	Top level element in <b>Settings Management</b> . Categories can contain groups, scalars, and scalar lists, which are displayed in a tree structure.
<b>Add Category</b>	Add a new category.
<b>Upload JSON File</b>	Load settings data from a JSON file to an instance of ViNO.
<b>Download JSON File</b>	Select and save categories to a JSON file. Use this file to load your saved categories to a new instance of ViNO.
<b>Apply Button</b>	Apply and save changes made to elements.
<b>Add Group</b>	Add a group to an element.
<b>Add Scalar List</b>	Add a scalar list, which functions similarly to a group except that you can only add scalars. Scalar lists do not allow groups to be added.
<b>Add Scalar</b>	Add values used in services.

# Using Settings Categories

When you create a settings category, ViNO automatically creates a Defaults group and names it with the category display name. Each group that is added inherits the values from the initial Defaults group. Defaults groups cannot be deleted. However, you can add groups, scalars, and scalar lists to a Defaults group. Elements added to a Defaults group are automatically inherited by all groups for that category. Default values are identified by an asterisk ( \* ).

- [Adding a Settings Category](#)
- [Modifying the Display Name in a Settings Category](#)

## Adding a Settings Category

To add a new settings category:

1. Click Add Category. A window displays that enables you to enter values in the following fields:
  - **Name** – Category name that is displayed in service definitions or during service activation. A name must be one word and can consist of letters, numbers, and hyphens. The **Name** field is a unique identifier and once set, cannot be changed.
  - **Display Name** – Category name that is displayed in the UI (such as, the **Settings** screen and the **Service Activation** screen) to provide a display-friendly and readable name for the category. The **Display Name** field can contain multiple words (including letters, numbers, spaces, and hyphens).
2. Click Apply to save the new category.

## Modifying the Display Name in a Settings Category

To modify the display name for a category:

1. Highlight the category in the **Settings Categories** section. Highlighting a category expands the tree in addition to displaying information related to that category.
2. Click in the **Display Name** field, and then overwrite the existing name with the new name.
3. Click Apply.

## Using Groups

Groups are used to store other groups, scalars, and scalar lists. Groups are similar to categories, but they do not contain their own Defaults group.

Attributes for groups include:

- **Name** – Name that is displayed in service definitions or during service activation. A name must be one word and can consist of letters, numbers, and hyphens. The **Name** field is a unique identifier and once set, cannot be changed.
- **Display Name** – Name that is displayed in the UI (such as, the **Settings** screen and the **Service Activation** screen) to provide a display-friendly and readable name for the group. The **Display Name** field can contain multiple words (including letters, numbers, spaces, and hyphens).

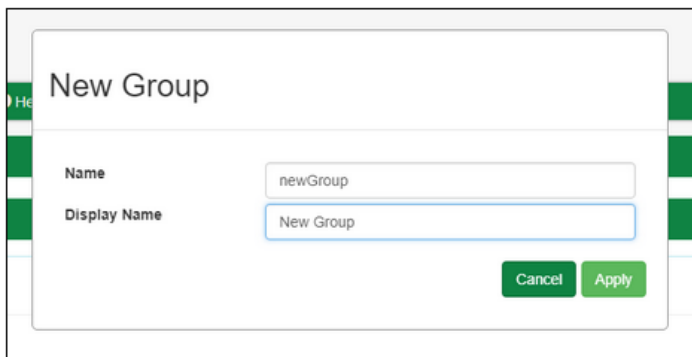
Refer to the following sections for information on using groups:

- [Adding a Group](#)
- [Modifying the Display Name for a Group](#)

## Adding a Group

To add a new group:

1. Select a category or group in the tree. The properties window for that element is displayed.
2. Click the Add Group button. Alternatively, you can right-click a category or group and select Add > Group.
3. Enter a name in the **Name** field.
4. Enter a name in the **Display Name** field.
5. Click Apply to save the new group.

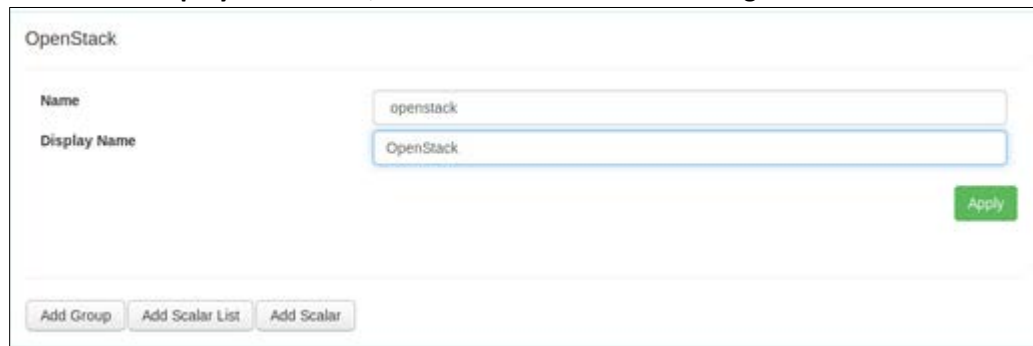


The image shows a 'New Group' dialog box. It contains two text input fields. The first field is labeled 'Name' and contains the text 'newGroup'. The second field is labeled 'Display Name' and contains the text 'New Group'. Below the fields are two buttons: 'Cancel' and 'Apply'.

## Modifying the Display Name for a Group

To modify the display name for a group:

1. Select the group in the tree to open the **Group** window.
2. Click in the **Display Name** field, and then overwrite the existing name with the new name. For example:



The screenshot shows a web interface for configuring a group named 'OpenStack'. It features two text input fields: 'Name' with the value 'openstack' and 'Display Name' with the value 'OpenStack'. The 'Display Name' field is highlighted with a blue border. A green 'Apply' button is located to the right of the 'Display Name' field. At the bottom of the window, there are three buttons: 'Add Group', 'Add Scalar List', and 'Add Scalar'.

3. Click Apply to save your changes.

## Using Scalars

Scalars contain values that are used by ViNO services. Scalars can contain string, Boolean, or number values. You can also designate these values as required for service activation.

Scalars that are added to a:

- Defaults group are inherited by all groups.
- Non-default groups override inherited default scalars with the same name.

Refer to the following sections for information on using scalars:

- [Adding a Scalar](#)
- [Modifying a Scalar](#)

## Adding a Scalar

To add a new scalar:

1. Select a category, group, or scalar list in the tree, and then click Add Scalar. Alternatively, right-click on an element and select Add > Scalar.
2. Enter values in the **Name**, **Display Name**, **Required**, **Type**, and **Value** fields.
3. Click Apply to save the new scalar. For example:

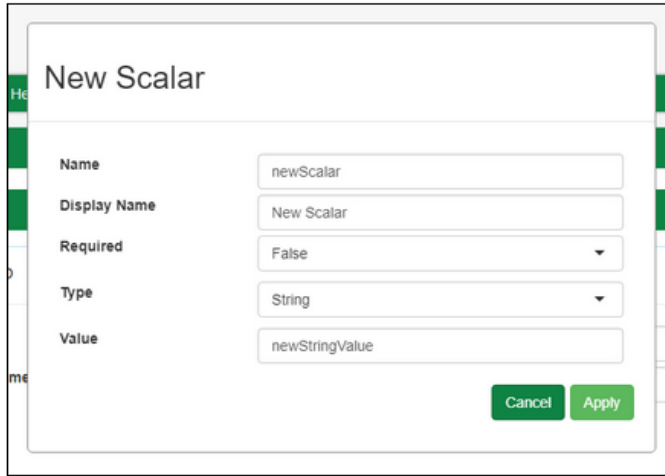


Table 8 describes the fields in the New Scalar window.

**Table 8. New Scalar Field Descriptions**

Field	Description
<b>Name</b>	Scalar name that is displayed in service definitions and during service activation. A name must be one word and can consist of letters, numbers, and hyphens. The Name field is a unique identifier and once set, cannot be changed.
<b>Display Name</b>	Scalar name that is displayed in the <b>Settings</b> screen and the <b>Service Activation</b> screen to provide a display-friendly and readable name for the category. This field can contain multiple words (including letters, numbers, spaces, and hyphens).
<b>Required</b>	Determines whether a field is required (True) or optional (False). Select the appropriate option from the drop-down.
<b>Type</b>	Determines the value for the scalar. Select <b>String</b> , <b>Number</b> , or <b>Boolean</b> from the drop-down as appropriate.
<b>Value</b>	Value used by a ViNO service that references this scalar. The type of value that can be entered in this field is determined by the setting in the <b>Type</b> field for this scalar.



## Modifying a Scalar

To modify a scalar:

1. Select the scalar in the tree. The values for the scalar attribute are displayed. The name in the **Name** field for a scalar is a unique identifier and once set, cannot be changed.
2. Enter changes by backspacing over existing values or by selecting a value from a drop-down.
3. Click Apply to save your changes. To cancel changes, refresh the page.

## Using Scalar Lists

Scalar lists enable you to categorize and store scalars. While scalars are not required to be stored in a scalar list, you may want to organize certain scalars together. For example, environment variables related to a specific system or a set of systems that share common architecture, such as the IP address, SSH username, SSH key, and password for a given system.

Scalar lists function similarly to a group except that you can only add scalars. Scalar lists do not allow groups to be added.

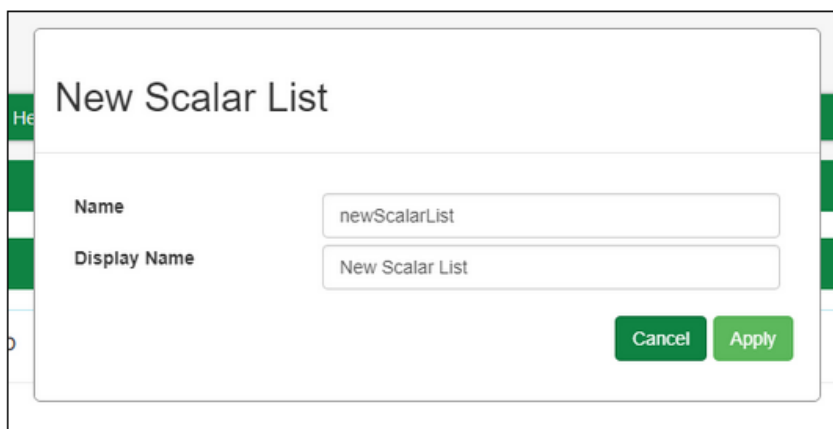
Refer to the following sections for information on managing scalar lists:

- [Adding a Scalar List](#)
- [Modifying the Display Name for a Scalar List](#)

## Adding a Scalar List

To add a new scalar list:

1. Select a category or group in the tree. The properties window for the selected element is displayed.
2. Click Add Scalar List. Alternatively, right-click the selected element and select **Add > Scalar List**.
3. Enter values in the [Name](#) and the [Display Name](#) fields in the **New Scalar List** window.
4. Click Apply to save the new scalar list.



The screenshot shows a 'New Scalar List' dialog box. It has a title bar with the text 'New Scalar List'. Below the title bar, there are two input fields. The first field is labeled 'Name' and contains the text 'newScalarList'. The second field is labeled 'Display Name' and contains the text 'New Scalar List'. At the bottom right of the dialog, there are two buttons: 'Cancel' and 'Apply'.

## Modifying the Display Name for a Scalar List

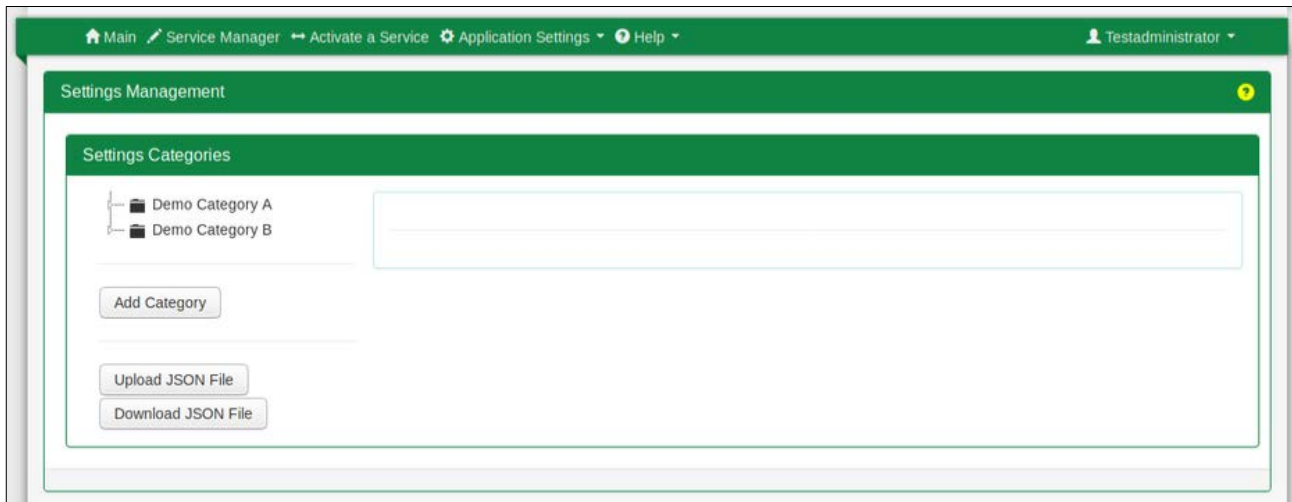
To modify the display name for a scalar list:

1. Click the scalar list in the tree. The scalar list properties window is displayed.
2. Click in the **Display Name** field, and then overwrite the existing name with the new name.
3. Click Apply to save your changes. To cancel the change, refresh the page.

## Downloading and Uploading a JSON File

ViNO enables you to save elements into a JSON file and then upload the file to another ViNO. Figure 5 shows the JSON file buttons on the **Settings Management** screen.

**Figure 5. JSON Upload and Download Buttons**



Refer to the following sections for information on using JSON files:

- [Downloading a JSON File](#)
- [Uploading a JSON File](#)

## Downloading a JSON File

Downloading enables you to save elements to a JSON file and then upload the file to another instance of ViNO.

To save elements to a JSON file:

1. Click the **Download JSON File** button (see [Figure 5](#)). A window opens enabling you to select which elements to save to a JSON file.
2. Navigate to the element you wish to save and select it.
3. Click Download.



4. In the **Opening vino-settings.json** window:
  - a. Select the **vino-settings.json** checkbox. This is the default name. You can change this name as needed after the file is saved.
  - b. Select the action the browser should take.
    - In Firefox, save the file to a location or open it.
    - In Chrome, the filename appears in the bottom left of the screen. Click the down arrow to open the file or display it in your Download folder.

## Uploading a JSON File

Uploading a JSON file enables you to re-use previously saved elements in a settings category.

To upload a JSON file:

1. Click the **Upload JSON File** button (see [Figure 5](#)). The **Select a file to load** window is displayed.
2. Click Choose File, and then navigate to the location of the JSON file you wish to upload.



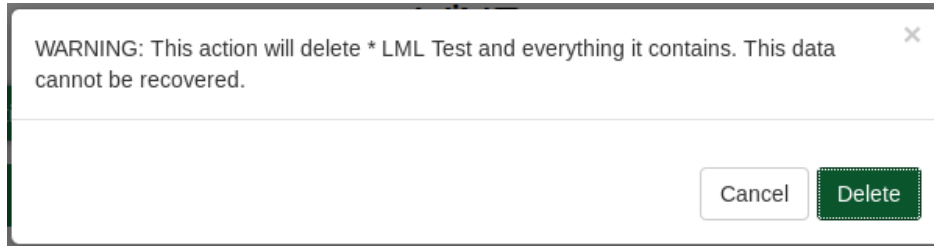
3. Select the file and then click Open. The file is displayed.
4. Scroll to the bottom of the file and click **Upload**. The new elements are uploaded and displayed in the **Settings Categories** section.

## Deleting Elements

**Caution:** All elements stored within a deleted settings category, group, scalar, or scalar list are also deleted. This process cannot be undone and the deleted values cannot be retrieved. Default groups cannot be deleted directly, but are deleted if the category is deleted.

To delete a settings category, group, scalar, or scalar list:

1. Right-click the element in the tree.
2. Select the Delete option. A warning confirmation message is displayed and requires you to confirm the action. The following example shows the confirmation message to delete the scalar list *LML Test*:



3. Click Delete to delete the element.

## Application Settings—User Management

The **User Management** option (**Application Settings** menu) opens the Keycloak homepage where you can access the Admin Console and the [Keycloak](#) documentation.

## Service Manager

The **Service Manager** enables a [designer](#) to create reusable network services by designing flows that represent the steps required to create a service. The main difference between a ViNO service and a Node-RED flow is that a ViNO service must always start with a [service entrypoint](#) node and end with a [service endpoint](#) node.

**Note:** The term *flow* is used for a single set of connected nodes and for the tabs in the Service Manager, which can contain multiple flows (sets of connected nodes). A *project* includes all the flow tabs and subflows in the Service Manager.

The underlying technology is Node-RED with several additions to:

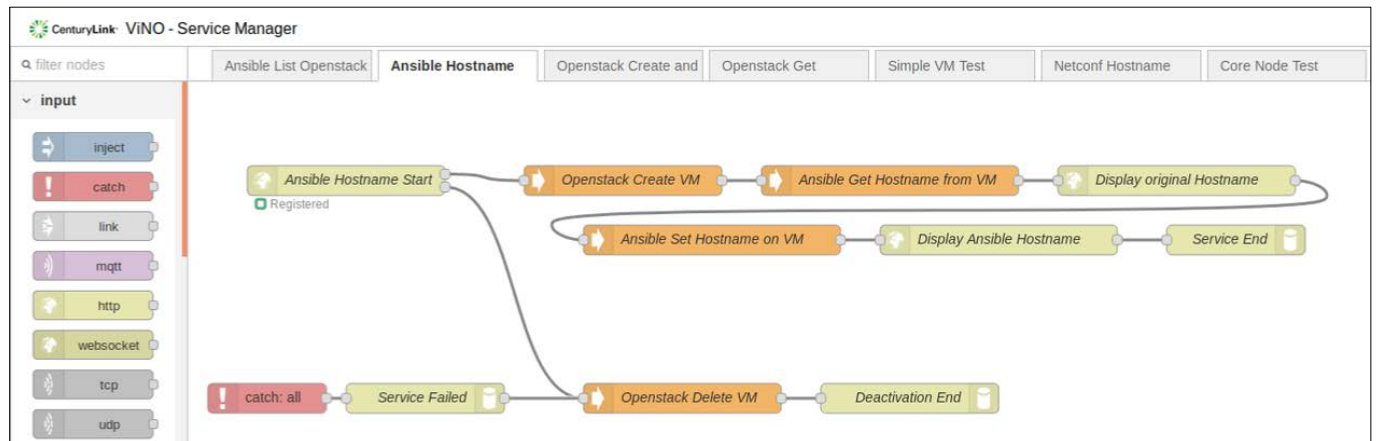
- Interact with Openstack.
- Configure services using Ansible and Netconf.
- Define how data is passed between steps within a flow.

ViNO can also leverage the nodes built in to Node-RED as well as an array of third-party nodes developed by the Node-RED community.

**Caution:** To avoid being logged out after 10 minutes while working in the Service Manager, you must open the Service Manager in a separate window. If you only have the Service Manager open, you are logged out of ViNO after 10 minutes with no warning. Even after being logged out, it appears that you can work and save in the Service Manager, but your changes are not saved.

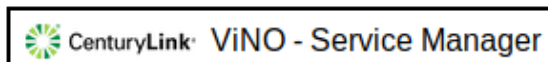
Figure 6 shows a partial **Service Manager** screen that has the Ansible Hostname tab displayed.

**Figure 6. Service Manager Screen**



To display a description of a node in the Palette, hover your cursor over it.







To return to the homepage, click VINO-Service Manager located in the top left of the **Service Manager** screen.



# Service Manager Components

Table 9 describes the components on the **Service Manager** screen.

**Table 9. Service Manager Components**

Component	Description
<b>Save icon</b> 	Saves all flows for the project. This icon briefly changes into a loading icon during the process of saving the flows.
<b>Menu icon</b> 	Drop-down that includes the following options: <b>Projects, View, Import, Export, Search Flows, Configuration Nodes, Flows, Subflows, Manage Palette, Setting, Keyboard Shortcuts, Node-RED Website,</b> and <i>Node-RED software version</i> . Refer to the <a href="#">Node-RED documentation</a> for information on these options.
<b>Designer workspace</b>	Create flows in this space.
<b>Right sidebar</b>	<p>Includes the following informational displays. See <a href="#">Working with the Sidebar</a>.</p> <ul style="list-style-type: none"> <li>  <b>info</b> – Displays information for the node currently highlighted.         </li> <li>  <b>history</b> – Displays the history of your changes and enables you to revert or push changes to a Git repository.         </li> <li>  <b>debug</b> – Displays debug messages from the <b>debug</b> node.         </li> <li>            Informational drop-down that includes the following options. When you select an option, its icon is added to the sidebar.           <ul style="list-style-type: none"> <li><b>Node information</b> – Same description as the <b>info</b> icon.</li> <li><b>Project history</b> – Same description as the <b>history</b> icon.</li> <li><b>Debug messages</b> – Same description as the <b>debug</b> icon.</li> <li><b>Configuration nodes</b> – N/A.</li> <li><b>Context data</b> – N/A.</li> </ul> </li> </ul>
<b>Node palette</b>	Located in the left panel of the screen. Includes the node categories listed below.
subflows	Set of connected nodes that can be reused within a larger flow. A subflow is represented as a single node in the workspace that can be opened in a tab to display the underlying flow. The <b>subflow</b> node category does not exist until a subflow is created.

Component	Description
input	Includes nodes that cover the basic communications mechanisms that applications are likely to use. The nodes range from UDP and TCP to the higher-level HTTP and the publish/subscribe MQTT (Message Queuing Telemetry Transport).
output	Essentially the mirror images of the basic set of input nodes that provide a way to send data on the same set of protocols (for example, MQTT, HTTP, UDP).
function	Nodes that carry out specific processing functions. Functions range from the simple <b>delay</b> and <b>switch</b> nodes to the programmable <b>function</b> node that can be adapted to most programming needs.
social	Basic social media nodes that support interaction with email and Twitter. These nodes enable flows to send or receive email or to send or receive tweets.
storage	Targeted at devices such as the Raspberry Pi. This category is limited and focuses on file-based storage.
analysis	Performs standard analyses on incoming messages. The only node provided is the <b>sentiment</b> node, which can be used to try to determine the sentiment of an incoming message based on the words used in the message (for example, an email or a tweet).
advanced	Miscellaneous nodes offering various types of functionality.
ViNO	Nodes used to create ViNO services.  The additional nodes included in the software enhance the functionality of Node-RED by enabling ViNO to programmatically and automatically generate activation templates that have full compatibility with existing Node-RED default nodes, as well as many of the community-contributed nodes available on the internet.
Raspberry Pi	Nodes specific to Raspberry Pi usage.
abacus	Nodes used to create web services with Node-RED.

# Service Manager Concepts

The Service Manager enables you to access the designer functionality to create nodes and flows. Table 10 describes the concepts of the Service Manager functionality.

**Table 10. Service Manager Concepts**

Concept	Description
<b>Flow</b>	Describes a single set of connected nodes. Click the <a href="#">Save</a> icon to save a flow. This term is also used to describe a tab in the Service Manager, which can contain multiple flows (sets of connected nodes).
<b>Parent Flow</b>	A parent flow contains subflows.
<b>Flow (Tab)</b>	A flow is represented as a tab at the top of the Service Manager workspace and is the main way to organize nodes. Use the <b>Project History</b> option ( <a href="#">Informational drop-down</a> ) to save all flow tabs and subflows in the Service Manager (known as a <i>project</i> ).
<b>Project</b>	Includes all the flow tabs and subflows in the Service Manager for the active project. Only one project may be active at a time.
<b>Subflow</b>	A subflow is a collection of nodes that are collapsed into a single node in the workspace. You can use subflows to reduce some visual complexity of a flow or to package a group of nodes as a reusable component used in multiple places. See <a href="#">Working with Subflows</a> .
<b>Workspace</b>	The Workspace is the main area where you develop flows by dragging nodes from the palette and wiring them together. The workspace has a row of tabs along the top; one for each flow and subflow. See <a href="#">Working in the Service Designer</a> .
<b>Node</b>	A node is the basic building block of a flow. Nodes are triggered by either receiving a message from the previous node in a flow or by waiting for an external event, such as an incoming HTTP request or a timer. A node processes the message or event and then may send a message to the next node in the flow. A node can have only one input port and as many output ports as it requires. See <a href="#">Working with Nodes</a> .
<b>Configuration Node</b>	A <b>config</b> node is a special type of node that holds reusable configuration that can be shared by regular nodes in a flow. <b>Config</b> nodes do not appear in the main workspace, but can be seen by opening the Configuration Nodes option in the <a href="#">Menu</a> icon drop-down (next to the Save button). See <a href="#">Working with the Sidebar</a> .
<b>Message</b>	Messages are passed between the nodes in a flow. They are plain JavaScript objects that can have a set of properties. Messages are often referred to as <b>msg</b> within the editor. See <a href="#">Working with Messages</a> .
<b>Wire</b>	Wires connect the nodes and represent how messages pass through the flow. See <a href="#">Working with Wires</a> .

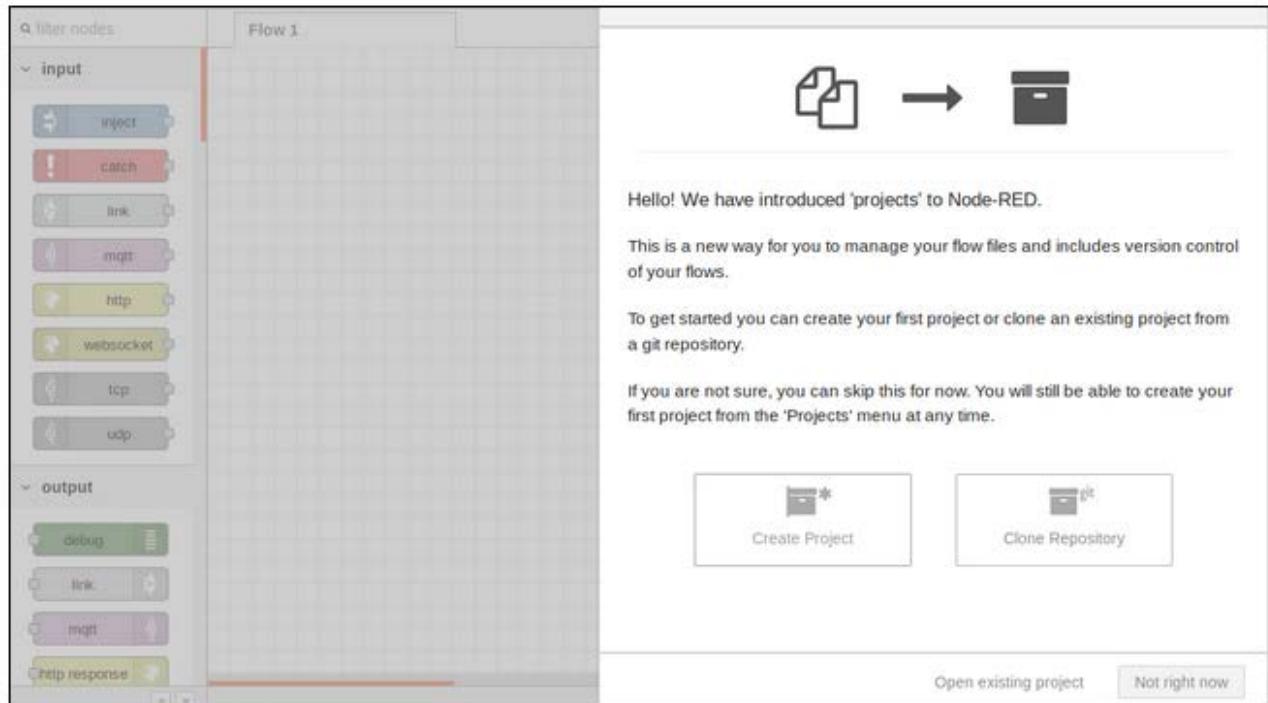


## Creating or Cloning a Project

When you access the Service Manager on a new ViNO, you are presented with the options to create a project or clone an existing project from a Git repository. Cloning a repository maps your ViNO to the correct Git repository used to save your projects.

Figure 7 shows the **Service Manager** screen in a new ViNO.

**Figure 7. Service Manager Screen in a New ViNO**



Refer to the following sections for information on creating and cloning projects:

- [Creating a Project](#)
- [Cloning a Project Repository](#)

## Creating a Project

To create a new Node-RED project:

1. Click Create Project.
2. Enter your username and email for your Git repository in the **Setup your version control client** window. Both fields are required.
3. Click Next.
4. Enter your project name (required) and an optional description in the **Create your project** window.
5. Click Next.
6. (Optional) You can configure a name for your flow file and credentials file that will be uploaded to your repository in the **Create your project files** window.
7. Click Next.
8. (Optional) You can provide a custom encryption key to protect the project files in your repository in the **Setup encryption of your credentials file** window.
9. Click Create Project. A success message is displayed.
10. Click Done.

## Cloning a Project Repository

Cloning a repository maps your ViNO to the correct Git repository used to save your projects.

To clone a repository:

1. Click Clone Repository (See [Figure 7](#)).
2. Enter your username and email for your Git repository in the **Setup your version control client** window. Both fields are required.
3. Click Next.
4. Enter the following values in the **Clone a project** window. All fields except for the credentials encryption key are required.
  - a. Your project name.
  - b. Repository URL for the project you wish to clone.
  - c. Enter your username and password for your Git repository.
  - d. Enter the credentials encryption key.
  - e. Click Clone Project. The project is loaded.

## Saving a Project

It is likely that the services you develop on your instance of ViNO will need to be deployed or shared with other ViNO instances. Therefore, it is important to save the flows that you create so they can be transferred easily between ViNO instances. Use the Project History icon or option in the [Informational drop-down](#) to save flow tabs and subflows by pushing the changes to a Git repository. This option also enables you to revert changes instead of saving them.

When you clone a Git repository, your ViNO Service Manager maps directly to a Git repository to provide a version control system for saving projects. The Git repository mapping is done when you clone a repository on a new instance of ViNO.

**Caution:** Do not use the **Import** or **Export** options ([Menu](#) icon drop-down) to share or save a project. These functions lose parameter mappings. See [Using Parameter Mapping](#).

## Working in the Service Designer

The main workspace is where you develop service flows by dragging nodes from the palette and wiring them together. The workspace has a row of tabs along the top: one for each flow and subflow that is open.

The footer of the workspace includes buttons to zoom in and out as well as the ability to reset the default zoom level. The footer also includes a toggle button for the view navigator. The view provides a scaled down view of the entire workspace that highlights the area currently visible. You can drag this area around the navigator to quickly jump to other parts of the workspace. You can also customize the workspace view using the **View** tab in the **User Settings** window ([Menu](#) icon > **Settings** option).

Refer to the following sections for information on using the workspace components.

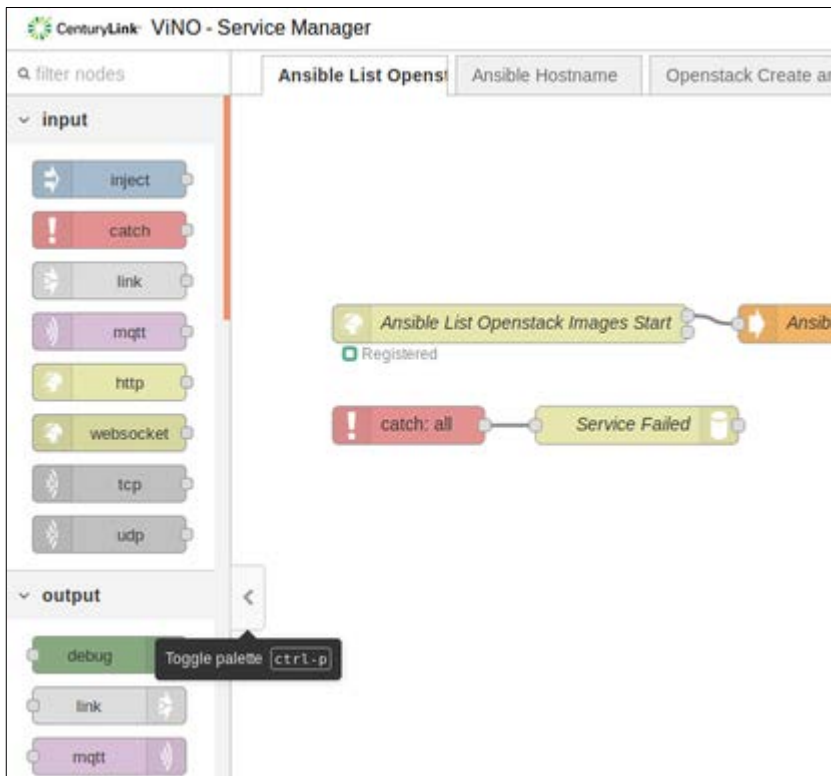
- [Working in the Palette](#)
- [Managing Nodes in the Palette Manager](#)
- [Working with Flow Tabs](#)
- [Working with Nodes](#)
- [Working with Wires](#)
- [Working with Subflows](#)
- [Working with Messages](#)
- [Working with the Sidebar](#)
- [Using Parameters](#)
- [Using Loops](#)
- [Using Parallel Execution](#)

## Working in the Palette

The Palette is located on the left of the Service Manager workspace and lists nodes (by section) that are available to use in flows. Nodes are organized into categories, with inputs, outputs and functions listed at the top. Subflows (when they exist) appear in a category at the top of the palette. Click a category to expand or collapse the node list.

Above the palette is the **filter nodes** field that enables you to search for nodes. To hide the entire palette, click the Toggle Palette arrow as shown in Figure 8.

**Figure 8. Filter Node Field and Toggle Palette**



To display a complete description of a node, hover your cursor over it. You can install additional nodes into the palette using either the command-line or the Palette Manager. See [Managing Nodes in the Palette Manager](#).

# Managing Nodes in the Palette Manager

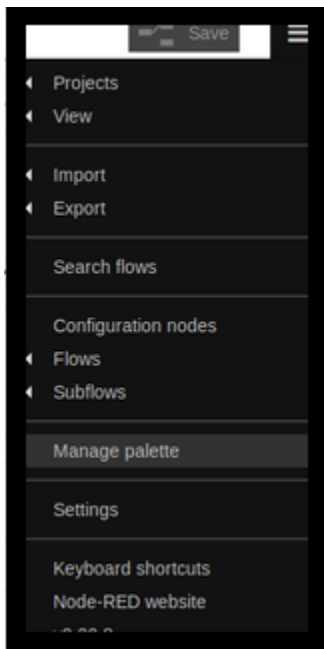
The Palette Manager enables you to manage nodes and install new nodes into the palette. See the following sections for managing nodes:

- [Accessing the Palette Manager](#)
- [Managing Nodes](#)
- [Adding Nodes to the Palette](#)
- [Using Built-In or Third-Party Node-RED Nodes](#)

## Accessing the Palette Manager

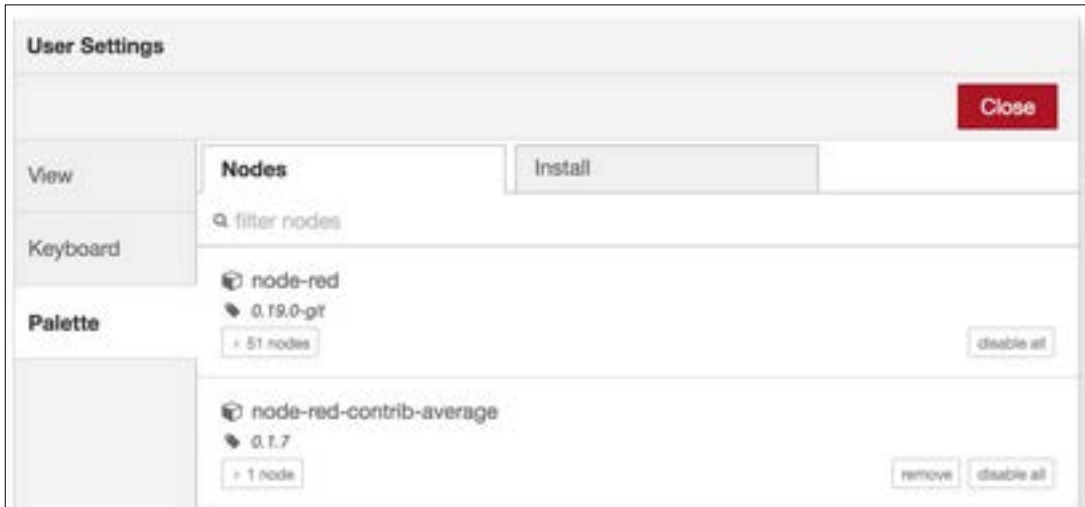
To access the Palette Manager:

1. Click the [Menu](#) icon drop-down (  ).
2. Select **Manage Palette**.



The **User Settings** window opens and displays the **Palette** tab by default as shown Figure 9.

Figure 9. User Settings Window



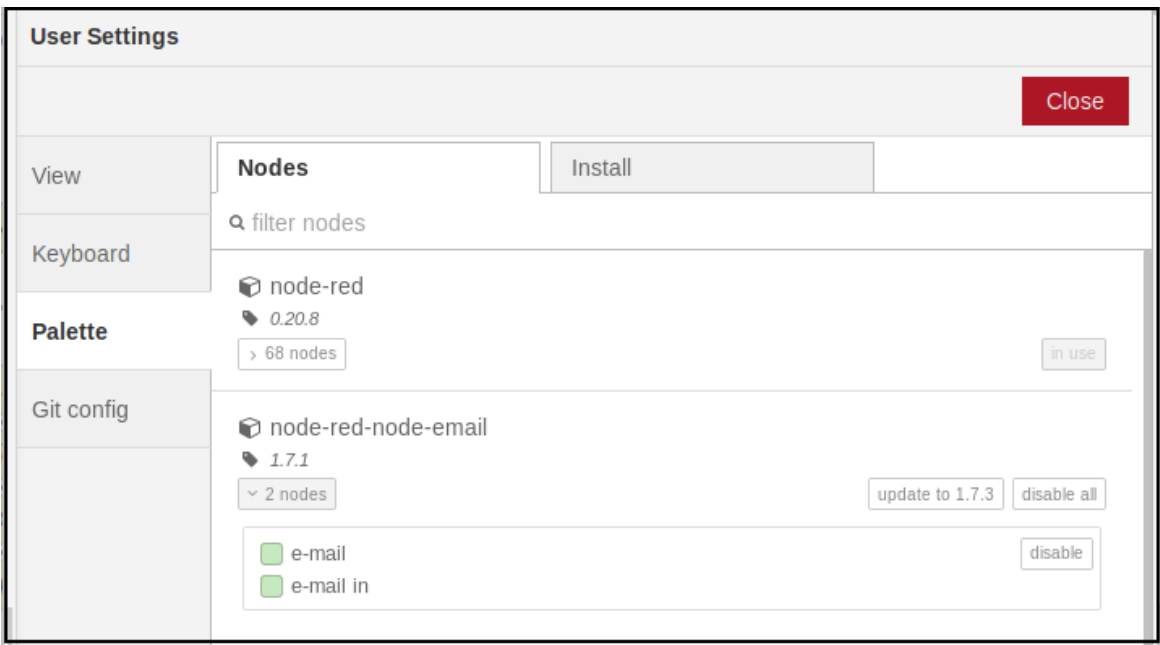
## Managing Nodes

Each entry in the **Nodes** list (**Menu** icon > **Settings** option > **Palette** tab > **User Settings** window) displays the name and version of each module (bundled nodes) in addition to a list of the individual node types the module provides. Use the Remove, Disable All, and Update buttons as needed to manage modules.

**Note:** A node that is currently in use cannot be removed or disabled.

Figure 10 shows the **Nodes** list in the **Palette** tab.

Figure 10. User Settings–Nodes List in Palette Tab



## Adding Nodes to the Palette

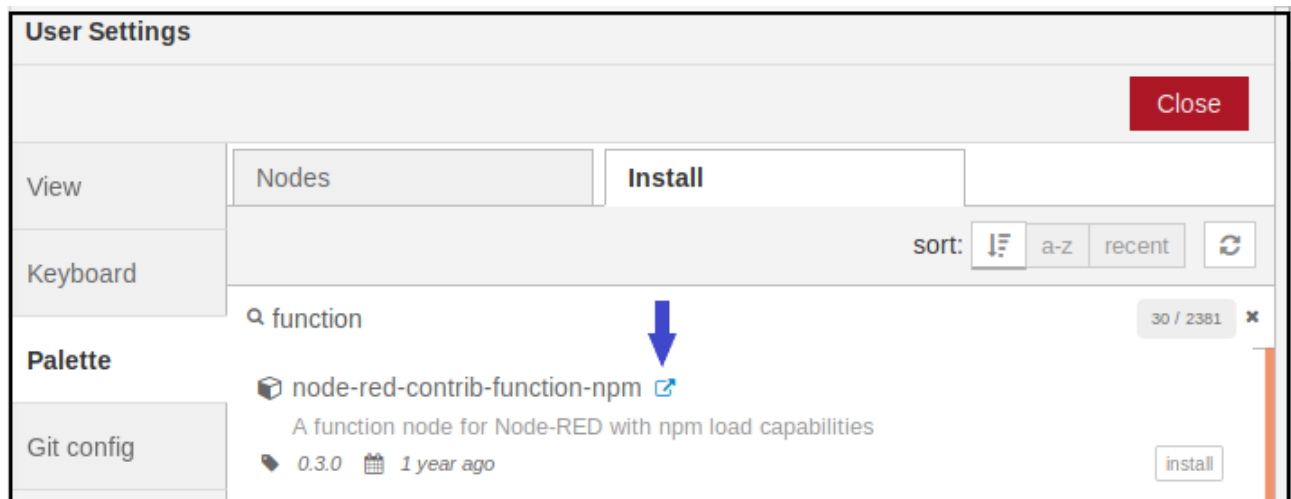
The **Install** button (**Palette** tab) enables you to search for available modules (bundle of nodes) and install them.

**Caution:** Third party nodes may not be compatible with ViNO if the nodes do not follow Node-RED standards.

To search for a module, enter its name in the Search field. The search results display the details of the modules, including when it was last updated and a link to its documentation. Figure 11 shows the **Install** tab with an arrow pointing to the documentation icon.

To install the module, click Install.

**Figure 11. Palette Install Tab**



## Using Built-In or Third-Party Node-RED Nodes

As part of a service definition, it may be necessary to incorporate nodes that are not included with ViNO. While other nodes can be freely mixed in within ViNO service flows, additional steps may be taken to make them useful. The output of non-ViNO nodes can only interface with ViNO input and output parameters using the utility nodes provided with ViNO (for example, the [service status](#), [parameter wrapper](#), and [throw](#) nodes).

Due to limitations in the Node-RED flow activation procedure, not all third-party nodes function correctly in ViNO service flows. A third-party node that replaces the message passed into it with its own object rather than modifying it prevents a service flow from activating correctly. Consider the implications before you modify the **msg** object mid-flow to prevent ViNO service failures.

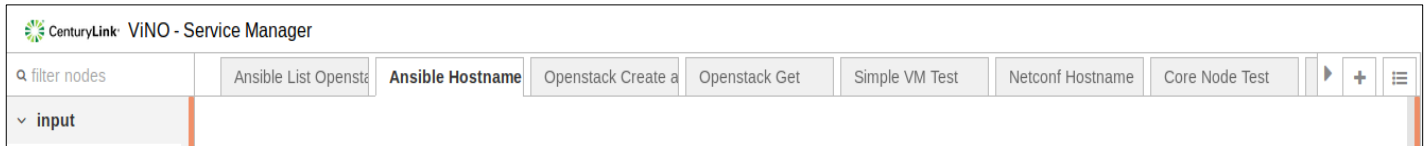
# Working with Flow Tabs

A flow is represented as a tab (a flow tab) within the Service Manager workspace and is the main way to organize nodes.

**Note:** The term *flow* is used for a single set of connected nodes and for the tabs in the Service Manager, which can contain multiple flows (sets of connected nodes). A *project* includes the flow tabs and subflows in the Service Manager.

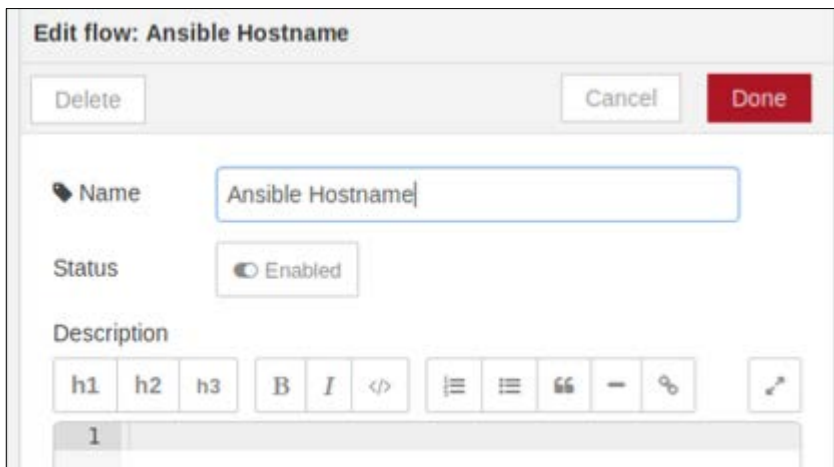
Figure 12 shows seven flow tabs in the Service Manager workspace.

**Figure 12. Service Manager Flow Tab Navigation Bar**



Double-click a flow tab to open the **Edit Flow** sidebar, which displays the name, status, and description (if provided) of the flow. You can enter a description as needed. Figure 13 shows an example of the **Edit Flow** sidebar.

**Figure 13. Edit Flow Sidebar**




Refer to the following sections for information on managing flow tabs:

- [Adding a Flow Tab](#)
- [Editing Flow Tab Properties](#)
- [Enabling or Disabling a Flow Tab](#)
- [Deleting a Flow Tab](#)
- [Switching Between Flows](#)



## Adding a Flow Tab

To add a new flow, either click the Plus icon (  ) in the flow tab navigation bar (see [Figure 12](#)) or double-click any free space in the tab bar. A flow tab can be reordered by dragging it in the tab bar.

## Editing Flow Tab Properties

To edit the properties for a flow tab or to enable or disable a flow, double-click the tab in the top bar to open the **Edit Flow** sidebar as shown in [Figure 13](#). Enter a name and description for the flow. The description can use Markdown syntax for formatting and appears in the Information sidebar.

To change the status of a flow, toggle the current status in the **Status** field. If a flow tab is disabled, none of the nodes it contains are created when the flow tab is deployed.

## Enabling or Disabling a Flow Tab

To enable or disable a flow tab, double-click the tab in the top bar to open the **Edit Flow** sidebar (see [Figure 13](#)). To change the status, toggle the current status in the **Status** field.

When a flow tab is disabled, none of the nodes it contains are created when the flow tab is deployed. For example, if you have an unfinished flow and save it, the save process fails and displays errors. If you first disable the flow, you can save successfully.

## Deleting a Flow Tab

To delete a flow tab, click Delete in the **Edit Flow** sidebar. See [Figure 13](#).

## Switching Between Flows

To display a list of the available flow tabs, click the [Menu](#) icon in the top bar. Click the flow tab you wish to open. The display only lists flow tabs. It does not list subflows (even though they are listed in the tab bar).

## Working with Nodes

The Service Manager palette includes a set of nodes that are the basic building blocks for creating flows. Nodes are joined by wires using their ports. A node can have only one input port but many output ports. A port may have a label that is displayed when you hover your mouse over it. A node may specify labels (for example, the **switch** node shows the rule that matches the port). You may also customize the name of a node by double-clicking the icon (in the workspace) and populating the **Name** field.

When a node has undeployed changes, it displays a blue circle above it. If there are errors with its configuration, the node displays a red triangle.

Certain nodes include a button on either the left or right edge. These buttons enable interactions with the node from within the editor. The **inject** and **debug** nodes are the only core nodes that have buttons.

**Note:** To display a complete description of a node, hover your cursor over it in the palette.

Refer to the following sections for information on core nodes:

- [Using the Inject Node](#)
- [Using the Debug Node](#)
- [Using the Function Node](#)
- [Using the Change Node](#)
- [Using the Switch Node](#)
- [Using the Template Node](#)
- [Using ViNO Nodes](#)
- [Using Conditional Start and Conditional End Nodes](#)
- [Adding Status Messages to Built-In Nodes](#)
- [Using the Function Node to Manipulate Parameters](#)

## Using the Inject Node

The **inject** node enables you to manually trigger a flow by clicking the node's button within the editor. This node can also be used to automatically trigger flows at regular intervals.



The message sent by the **inject** node can have its payload and topic properties set. The payload can be set to one of the following types:

- Flow or global context property value.
- String, number, Boolean, buffer, or object.
- Timestamp (in milliseconds) since January 1st, 1970.

## Using the Debug Node

The **debug** node enables you to display messages in the **debug** sidebar within the editor.



The sidebar provides a structured view of the messages it is sent, which makes it easy to explore the message. In addition to the message, the debug sidebar includes information about the time the message was received and which **debug** node sent it.

A **debug** node can also be configured to send all messages to the runtime log or to send a short (32 characters maximum) message to the status text under the **debug** node in a flow. Use the button on the node to enable or disable its output.

## Using the Function Node

The **function** node enables JavaScript code to be run against the messages that are passed through it.



## Using the Change Node

The **change** node enables you to modify the properties of a message and set context properties without having to resort to a **function** node.



Each **change** node can be configured with multiple operations that are applied in order. The available operations are:

- **Set** – Set a property. The value can be a variety of different types (displayed in the drop-down in the node Properties window) or can be taken from an existing message or context property.
- **Change** – Search and replace parts of a message property.
- **Move** – Move or rename a property.
- **Delete** – Delete a property.

## Using the Switch Node

The **switch** node enables messages to be routed to different branches of a flow by evaluating a set of rules against each message.



The **switch** node is configured with the property to test, which can be either a message property or a context property. The node routes a message to outputs corresponding to matching rules. You can also configure the **switch** node to stop evaluating rules when it finds one that matches. There are four types of rules:

- **Value** rules are evaluated against the configured property.
- **Sequence** rules can be used on message sequences, such as those generated by the **split** node.
- **JSON Data Expression** rules can be evaluated against the entire message and matches if the expression returns a true value.
- **Otherwise** rules can be used to match if none of the preceding rules have matched.

## Using the Template Node

The **template** node enables you to generate text using a message's properties to fill out a template.



This node uses the Mustache templating language to generate the result.

## Using ViNO Nodes

Table 11 describes the custom nodes provided with ViNO.

**Table 11. ViNO Nodes**

Node	Description
<b>service status</b>	Enables a designer to output arbitrary status messages during activation. Status messages are visible while they are activating and deactivating.
<b>conditional end</b>	Place this node at the end of a conditional branch. Parameters in nodes on conditional branches that need to be accessed by subsequent steps should be defined in the output parameters of this node.  <b>Note:</b> This node is deprecated and will be removed in a later version.
<b>conditional start</b>	Manages a conditional branching of service flow logic. This node provides parameters to define the <a href="#">Left Hand Side</a> and the <a href="#">Right Hand Side</a> operands of a Boolean expression in addition to the operation to perform and the data type of the operands.  <b>Note:</b> This node is deprecated and will be removed in a later version.
<b>deactivation endpoint</b>	When the service finishes with the <b>deactivation endpoint</b> node, the database elements related to that activation are updated (for example, with status changes).
<b>loop end</b>	Used to connect back to the input of a loop start node. Place this node at the end of a loop body.
<b>loop start</b>	Manages a loop in the service flow. This node contains the parameter that controls the number of iterations to perform. Each <b>loop start</b> must have a matching <b>loop end</b> to connect back to the <b>loop start</b> .
<b>parameter wrapper</b>	Enables arbitrary input parameters to map to properties of the <b>msg</b> object that is passed to a following node. Parameter wrappers can be used to: <ul style="list-style-type: none"><li>▪ Inject ViNO parameters into the Node-RED <b>msg</b> object.</li><li>▪ Extract attributes from the <b>msg</b> object and convert them into ViNO parameters.</li></ul>
<b>service endpoint</b>	When the service finishes with the <b>service endpoint</b> node, the database elements related to that activation are updated (for example, with status changes).  <b>Note:</b> Each ViNO service must end with a <b>service endpoint</b> node (unless your service needs to end with a <b>service failure</b> node or a <b>deactivation endpoint</b> node).

Node	Description
<b>service endpoint</b>	Creates an HTTP end-point for a service. <b>Note:</b> Each ViNO service must start with a <b>service endpoint</b> node.
<b>service failure</b>	When the service finishes with the <b>service failure</b> node, the database elements related to that activation are updated (for example, with status changes).
<b>throw</b>	Throws an error. This node is useful when used in conjunction with a <b>conditional</b> node to check whether a particular value is valid. This node generates a service failure or a rollback (depending whether a deactivation flow is defined). The <b>throw</b> node is typically paired with a <b>catch</b> node to facilitate proper error handling.
<b>vino driver ansible</b>	<p>ViNO driver node that performs Ansible operations. This node runs an Ansible playbook on the provided target system. The playbook provided can be templated using the Mustache templating syntax. All input parameters on the Ansible node are available for use within the playbook template and can be referenced with <code>{{parameter_key}}</code> anywhere in the playbook.</p> <p>To use the key-init features of this node with Openstack, you must generate a public and a private SSH key for ViNO to interact with Openstack.</p> <p>This node enables the use of a private key to authenticate requests against a remote server. For this to work, the private key must be readable by the root user of the vino-core docker container. To accomplish this, place the private key in a docker volume accessible to the vino-core container. The easiest way to do that is after the ViNO services are running, place the private key file in <b><code>/var/lib/docker/volumes/&lt;project name&gt;_ViNO-Common/_data/</code></b></p> <p>Files placed in this directory will be available at the path <b><code>/opt/vino/common</code></b> inside the vino-core container. The <i>project name</i> is the value configured when ViNO was initialized.</p> <p>The <b>vino driver ansible</b> node contains the input parameter <b>SSH Private Key</b> that should be configured with the full path to your private key inside the vino-core container. As an option to using the SSH Private Key, you can use the username and password parameters available on this node.</p>

Node	Description
<b>vino driver netconf</b>	<p>ViNO driver node that performs Netconf operations. This node performs a Netconf get or edit on the provided target system. The node takes in a Netconf template that uses the Mustache templating syntax similar to the Ansible node.</p> <p>To use the key-init features of this node with Openstack, you must generate a public and a private SSH key for ViNO to interact with Openstack.</p> <p>This node enables the use of a private key to authenticate requests against a remote server. For this to work, the private key must be readable by the root user of the vino-core docker container. To accomplish this, place the private key in a docker volume accessible to the vino-core container. The easiest way to do that is after the ViNO services are running, place the private key file in <b>/var/lib/docker/volumes/&lt;project name&gt;_ViNO-Common/_data/</b></p> <p>Files placed in this directory will be available at the path <b>/opt/vino/common</b> inside the vino-core container. The <i>project name</i> is the value configured when ViNO was initialized.</p> <p>The <b>vino driver netconf</b> node contains the input parameter <b>SSH Key</b> that should be configured with the full path to your private key inside the vino-core container. As an option to using the SSH Key, you can use the username and password parameters available on this node.</p>
<b>vino driver openstack</b>	ViNO driver node that performs OpenStack operations.

## Using Conditional Start and Conditional End Nodes

In some cases, it may be necessary to only perform actions in a service under a certain condition. For example, if you need to perform operations on a VM that may or may not exist, you can use a condition to determine whether the VM exists.

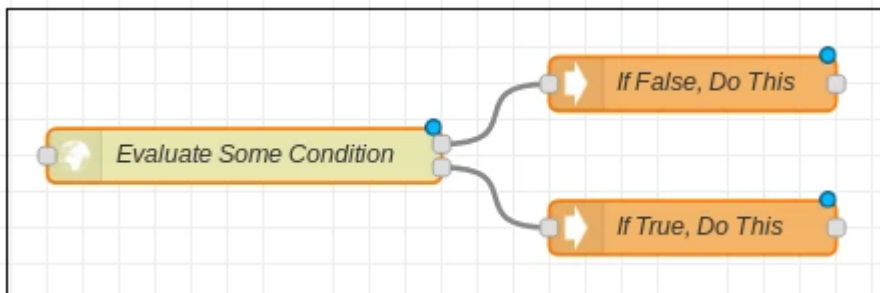
**Note:** The **conditional start** and the **conditional end** nodes are deprecated and will be removed in a later version.

While setting up conditional logic is possible in a standard Node-RED instance, ViNO provides the **conditional start** and the **conditional end** nodes to simplify this process and make it more compatible with the concept of input and output parameters. The **conditional start** node represents the beginning of a conditional branch of execution. This node includes the following input parameters that control how the Boolean expression is to be evaluated at activation time. The **conditional start** node compares the two values in the **Left Hand Side** and the **Right Hand Side** using the specified operator and executes one of two paths depending on the result.

- **Left Hand Side** – Left hand side operand of the expression.
- **Operation** – Comparison operator to use (equals, less than, less than or equal, greater than, greater than or equal).
- **Operand Data Type** – Data type of the two operands (string, number, or Boolean).
- **Right Hand Side** – Right hand side operand of the expression.

The top output of the node is intended to connect to the steps to take if the condition evaluates to false while the bottom output connects to the nodes to activate if it evaluates to true. Figure 14 shows an example of using a **conditional start** node.

**Figure 14. Example of Using the Conditional Start Node**



The data type of the **Left Hand Side** and the **Right Hand Side** of input parameters is always **string** and the operand data type parameter controls which type those values are converted to at runtime. Therefore, it is important to ensure the values in those parameters are valid for the configured operand data type. If the values are invalid, they trigger an error during activation.

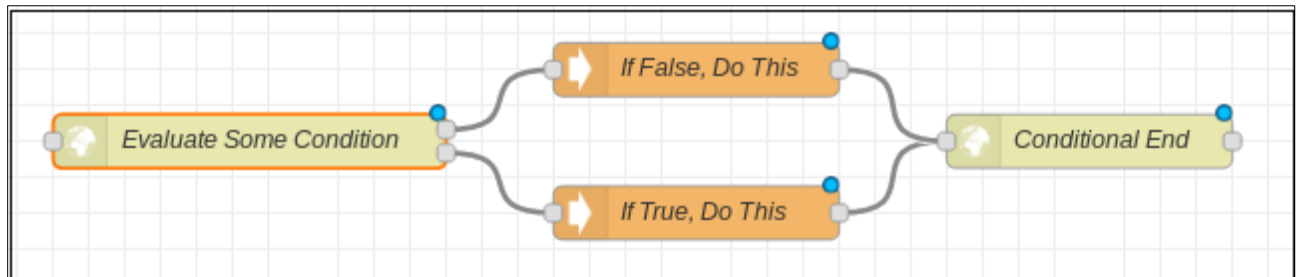
**Note:** When the operand data type is set to string or Boolean, only the **eq** (Equals Comparison) operation can be used.

The **conditional start** node also contains a single output parameter (Evaluation Result) that contains the result of the comparison.



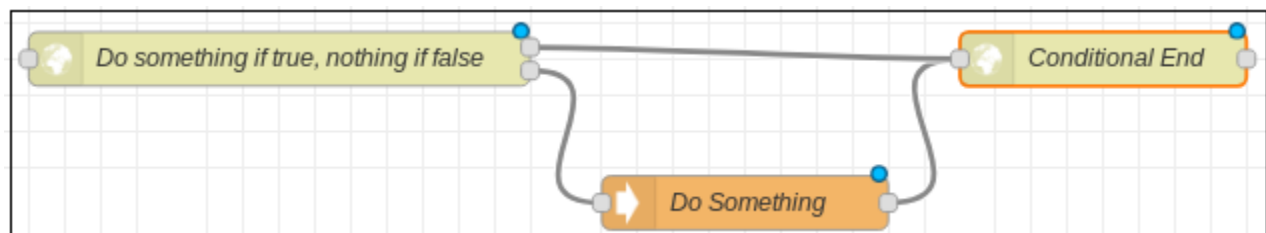
At the end of both the true and false branches, the last nodes in each branch must be joined back into a single **conditional end** node (shown in Figure 15). This node provides a clear indication where the conditional branching has ended and the service resumes a normal path. The **conditional end** node also serves an important role in [parameter mapping](#).

**Figure 15. Example of Using the Conditional End Node**



It is a requirement that both branches of the **conditional start** node eventually connect back to a single **conditional end** node. If only one of the branches needs to perform actions, the other branch should have its output connected directly to the **conditional end** node as shown in Figure 16.

**Figure 16. Example of Using Conditional Start and Conditional End Nodes**



## Adding Status Messages to Built-In Nodes

It may often be helpful to display status information about operations being performed by built-in nodes. To display status information, use the [service status](#) node, [parameter mapping](#), and the built-in Mustache templating in the status message to display information extracted from the node.

The **service status** node enables you to output arbitrary status messages during activation. The message can use Mustache-style templating syntax to inject the input parameters defined on this node into the status message. This is useful for including output data from a previous step in the status by:

1. Creating a new input parameter.
2. Mapping it to a previous node's output.
3. Referencing it in the **Message** template.

Properties of the message object can also be referenced in the template by using `msg.nameOfProperty`.

## Using the Function Node to Manipulate Parameters

There may be situations in which parameters need to be manipulated during an activation. For example, it may be necessary to concatenate two or more input parameters or perform some arithmetic on a value. This can be achieved using the **function** node. This node enables you to write JavaScript functions and directly manipulate the message object passed in Node-RED.

By injecting parameters into the message object and then accessing them in a **function** node, you can leverage the power of a full-featured scripting language to manipulate input parameters. After the parameter values are manipulated, the parameters can be injected on to the message object and extracted into an output parameter.

An example to use the **function** node to concatenate objects would be:

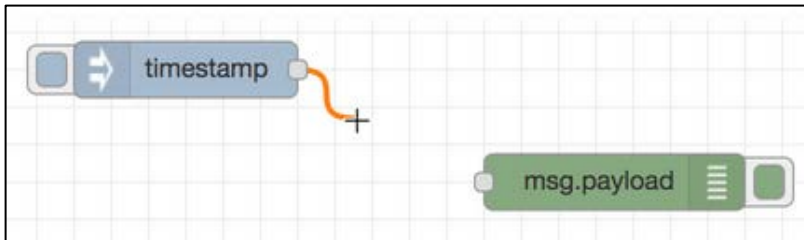
```
var first= 'Jane ';  
var last = 'Doe ';  
msg['full_name'] = first + ' ' + last;  
return msg;
```

**Note:** Within the **function** node the message object is referenced with **msg**. The original **msg** object (or a copy of that object) must always be returned at the end of a function. Returning a new object that lacks information injected into it by the ViNO nodes breaks the service flow and causes errors in activation.

## Working with Wires

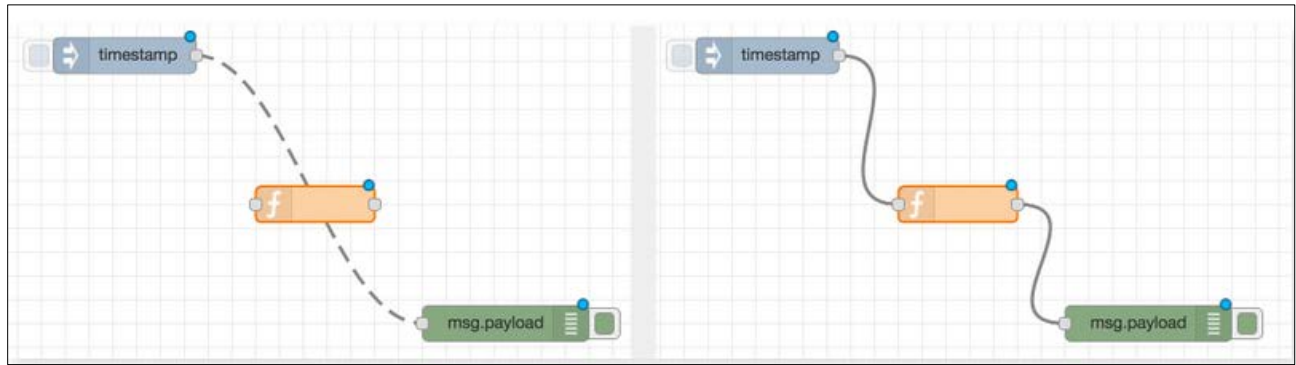
Nodes are wired together by clicking the left-mouse button on a node's port, dragging to the destination node, and then releasing the mouse button as shown in Figure 17.

**Figure 17. Working with Wires**



When you drag a node with both an input and output port over the mid-point of a wire, the wire is drawn with a dash. If the node is then dropped, it is automatically inserted into the flow at that point. Figure 18 shows how to drop a node on a wire to insert it mid-flow.

**Figure 18. Dropping a Node on a Wire**



## Moving Wires

To disconnect a wire from a port, select the wire by clicking on it, then press and hold the Shift key when the left-mouse button is pressed on the port. When you drag the mouse, the wire disconnects from the port and can be dropped on another port. If you release the mouse button over the workspace, the wire is deleted.

## Deleting Wires

To delete a wire, select it by clicking on it and then press the Delete key.

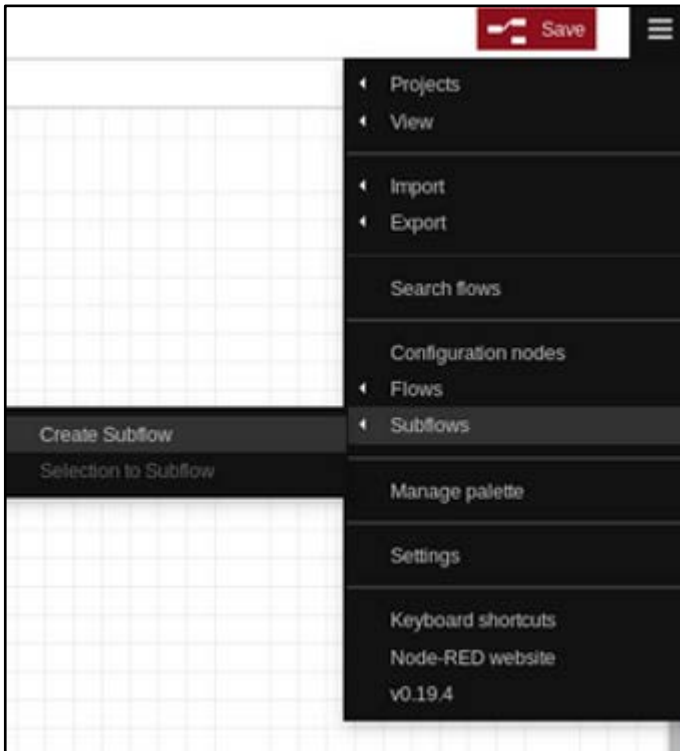
## Working with Subflows

Subflows enable a group of nodes to be packaged into a reusable piece of logic that is represented as another node in the palette. These subflows can then be used as part of parent flows. A [parent flow](#) is a flow that contains subflows. Conceptually, subflows are the equivalent of subroutines in traditional programming languages.

To create a subflow:

1. Open the Service Manager.
2. Click the [Menu](#) icon (see Figure 19).
3. Navigate to **Subflows**.
4. Select **Create Subflow**.

**Figure 19. Create Subflow Option**



This action opens a new tab that is almost identical to a normal flow tab with the addition of a small tool bar at the top. The tool bar enables you to:

- Control the number of inputs and outputs in the subflow.
- Open a menu to change subflow properties.

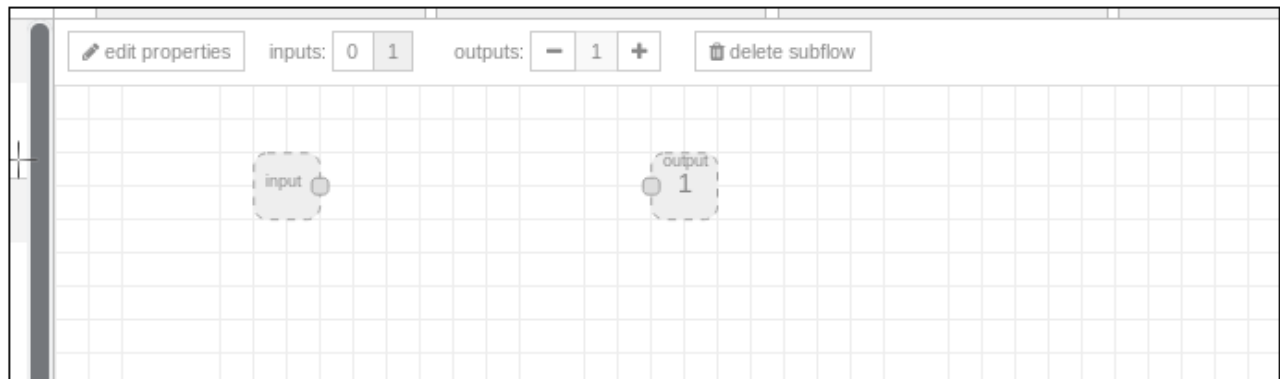
Figure 20 shows a new subflow (Subflow 1) and the toolbar.

**Figure 20. Example of New Subflow and Toolbar**



In most cases, subflows should have a single input and output that are represented as special nodes in the subflow editor as shown in Figure 21.

**Figure 21. Subflow Example**



## Designing a Subflow

To design a subflow:

1. Connect nodes starting from the input.
2. When the flow is complete, connect the last node to the output.

**Note:** Subflows should not contain **service endpoint** or **service endpoint** nodes because the subflow does not represent a ViNO service, but rather a small unit of work that is intended to be included in other services.

## Working with Messages

A flow works by passing messages between nodes. The messages are simple JavaScript objects that can have a set of properties. Messages usually have a **payload** property, which is the default property (or object) that most nodes work with. Messages also include an identifier property called **\_msgid** that traces a node's progress through a flow. The value of a message property can be a valid JavaScript type, such as:

- Boolean
- Number
- String
- Array
- Object
- Null

## Changing Message Properties

A common task in a flow is to modify the properties of a message as it passes between nodes. For example, the result of an HTTP request may be an object with many properties, of which only some are needed. There are two main nodes for modifying a message:

- **function** node – Enables you to run JavaScript code against the message. This gives you complete flexibility in what is done with the message, but does require familiarity with JavaScript and is unnecessary for many simple cases. See [Using the Function Node](#).
- **change** node – Provides functionality without the need to write JavaScript code. This node modifies message properties and also accesses flow- and global-context. See [Using the Change Node](#).

## Using the Service Status Node

The [service status](#) node enables you to inject custom status messages at any stage into a service. The message displayed is provided in the node configuration and can be templated using the Mustache syntax to display input parameters defined on the node by referencing them with `{{parameter_key}}`. This feature is useful for displaying status for non-ViNO nodes that do not write status messages.

## Working with the Sidebar

The sidebar contains icons that provide tools within the Service Manager:





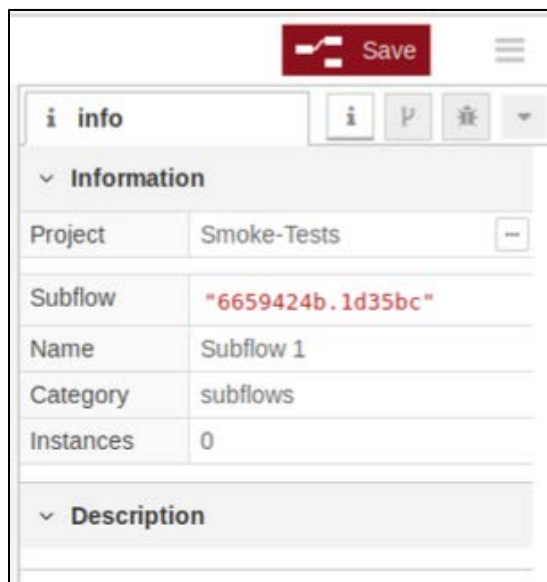
-  Node information – View information about nodes.
-  Project history – Displays changes to local files and changes to commit.
-  Debug messages – View messages passed to debug nodes.
-  Informational dropdown that lists the display options.

Figure 22 shows the display screen for Node Information.

**Figure 22. Service Manager Sidebar**



To open a display, click an icon in the header of the sidebar or, select it from the **Informational** drop-down



# Using Parameters

One of the main features of ViNO is how it handles the parameters used to drive service activations. There are two types of parameters that apply to ViNO nodes:

- **Input parameters** – Control how each step activates a service.
- **Output parameters** – Hold information that is output by a step in a flow that may be useful later in the service.

Parameters are unique to the [custom nodes provided with ViNO](#). However, you can use these parameters to manipulate other Node-RED nodes using the **parameter wrapper** node with the **Mode** set to **Parameter Injection/Extraction** to inject and extract data into a Node-RED message.

Refer to the following sections for information on using parameters.

- [Configuring Parameters on Nodes](#)
- [Editing Parameters](#)
- [Using Parameter Mapping](#)
- [Using Parameter Mappings and Conditionals](#)
- [Using Parameter Wrapper Injection and Extraction](#)
- [Using Parameter Combiner Modes](#)
- [Passing Parameters Between a Parent Flow and Subflow](#)



## Configuring Parameters on Nodes

Parameters are configured on a node in the **Property Edit** window. Double-click a node to open the **Property Edit** window specific to that node. Figure 23 shows an example of the **Property Edit** window for the **loop start** node. Depending on the node selected, additional fields may be displayed.

Figure 23. Property Edit Window

Edit loop start node

Delete

Cancel

Done

⚙ Properties

⚙

📄

🔍

🔍 Name

Loop Start Delete Inside Networks

🔍 Description

🔍 Mode

For Each Loop (String)

Status Message Options

Input Parameters

Add Parameter

Edit Parameter

Remove Parameter

Restore Defaults

Unset Parameter

Search:

Parameter Name	Parameter Description	Parameter Key	Parameter Value	Display Order
Input List	List to iterate over	inputList	Mapped from previous node Copy Inside Network ID List from Activation : insideNetworkIdList	

Showing 1 to 1 of 1 entries

Previous

1

Next

Output Parameters

Add Parameter

Edit Parameter

Remove Parameter

Restore Defaults

Search:

Parameter Name	Parameter Description	Parameter Key	Parameter Format	Extraction Method	Inject Into msg as
Current Value	The current value in the list being iterated over	currentValue	unused	CUSTOM	

Showing 1 to 1 of 1 entries

Previous

1

Next

Table 12 describes the fields, buttons, sections, and columns in the **Property Edit** window.

**Table 12. Property Edit Window**

<b>Component</b>	<b>Description</b>
Top navigation bar	<ul style="list-style-type: none"><li>▪ <b>Delete</b> – Deletes the node.</li><li>▪ <b>Cancel</b> – Cancels changes you made in Edit mode.</li><li>▪ <b>Done</b> – Saves changes and closes the Edit Parameters window.</li></ul>
<b>Name</b> field	Displays the name of the node in the flow.
<b>Description</b> field	Displays a description of the node (if one was entered when the parameter was added).
<b>Command</b> field	Determines the action the node takes for a step.
<b>Status Message Options</b> button	This button enables you to limit the type of messages per node that are displayed at activation. Select <b>Always</b> , <b>Debug mode only</b> , or <b>Never</b> from each drop-down as needed: <ul style="list-style-type: none"><li>▪ <b>Display Node Starting Messages</b></li><li>▪ <b>Display Node Completed Messages</b></li><li>▪ <b>Display Retry Messages</b></li></ul>
<b>Input Parameters</b> section	Displays the parameters configured for the node. Provides buttons to add, edit, or remove parameters as well as restore defaults and unset parameters. These functions are listed in the following rows.

Component	Description
<b>Add Parameter button</b>	<p>Opens the <b>Edit Input Parameter</b> window enabling you to add a new parameter. Depending on the option selected in the <b>Type</b> and <b>Parameter Source</b> fields, other fields may be displayed.</p> <p>The following fields apply to all nodes:</p> <ul style="list-style-type: none"> <li>▪ <b>Name</b> – Enter a descriptive name for the parameters.</li> <li>▪ <b>Key</b> – This field auto-populates as you type in the <b>Name</b> field.</li> <li>▪ <b>Description</b> – Enter a description of the parameter.</li> <li>▪ <b>Type</b> – Select a type from the drop-down.</li> <li>▪ <b>Parameter Source</b> – Select a source from the drop-down: <ul style="list-style-type: none"> <li>– <b>Node Configuration</b> – Requires you to add a value in the <b>Value</b> field.</li> <li>– <b>Message</b> – Requires you to add a name in the <b>Message Property Name</b> field. Enables you to get a value from the Node-RED message object and use it as the parameter source.</li> <li>– <b>File Reference</b> – Select a file from the <b>Project Files</b> drop-down or upload a file. Enables you to use a value in a file as the parameter source.</li> <li>– <b>Constants</b> – Enables you to use a constants file as the parameter source. Requires you to enter a path in the <b>Constants Path</b> field using the value in the <b>Name</b> fields for the settings. For example, <b>/categoryName/groupName/scalarName</b>. Several constants are pre-populated as you type.</li> <li>– <b>Mapped from another Node (ViNO Service Only)</b> – Enables you to select the node that has the parameter you want to map from (in the <b>Mapped From Node</b> field). You must then select the key you want to map from in the <b>Mapped from Key</b> field.</li> </ul> </li> <li>▪ <b>Display Order Index</b> – Use the arrow buttons to determine the order in which parameters are displayed. The lower the number, the higher the entry appears in the display.</li> <li>▪ <b>Optional</b> – Select this button to make the parameter optional. If <b>Optional</b> is not selected, the parameter is required by default to activate the service.</li> <li>▪ <b>Final</b> – Select this button to disable the ability to change the parameter at activation. Parameters that have this option selected are not displayed in on the <b>Activation</b> screen.</li> </ul>
<b>Edit Parameter button</b>	<p>Opens the <b>Edit Input Parameter</b> window enabling you to edit parameters for a specific node. Highlight a node entry and then click Edit Parameter (or double-click a parameter). Includes similar fields as the <a href="#">Add Parameter</a> button.</p>
<b>Remove Parameter button</b>	<p>Enables you to remove user-added parameters. You cannot remove built-in parameters.</p>

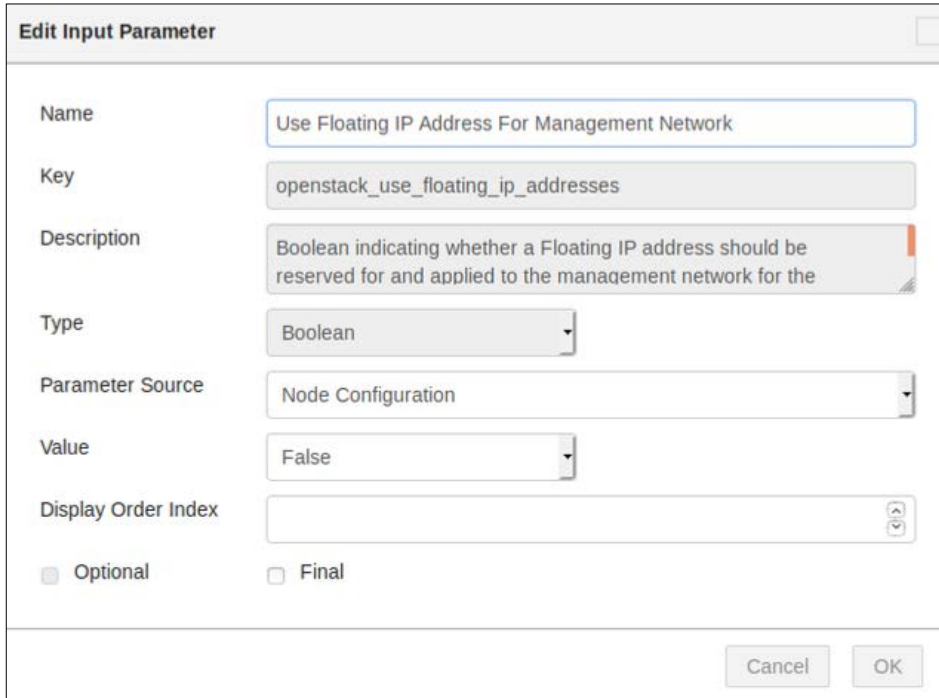
Component	Description
<b>Restore Defaults</b> button	Restores the default value (as displayed in the <b>Parameter Value</b> column) for the selected parameter.
<b>Unset Parameter</b> button	Clears the value (as displayed in the <b>Parameter Value</b> column) for the selected parameter.
<b>Parameter Name</b> column	Displays the name of the input parameter.
<b>Parameter Description</b> column	Displays a description of the input parameter if one was entered in the <a href="#">Description</a> field.
<b>Parameter Key</b> column	Displays the input parameter key for the node.
<b>Parameter Value</b> column	Displays the input parameter value if one has been configured on the node or displays the source for the value.
<b>Display Order</b> column	Displays a parameter in a specific order if the <a href="#">Display Order Index</a> field is set.
<b>Output Parameters</b> section	Displays the parameters that are configured to contain the output of the node. Provides buttons to add, edit, or remove output parameters as well as restore defaults. These functions are listed in the following rows.
<b>Add Parameter</b> button	<p>Open the Edit Output Parameter window enabling you to add an attribute to the Node-RED message object that contains the value of the output parameter.</p> <ul style="list-style-type: none"> <li>▪ <b>Name</b> – Enter a descriptive name for the parameters.</li> <li>▪ <b>Key</b> – This field auto-populates as you type in the <b>Name</b> field.</li> <li>▪ <b>Description</b> – Enter a description of the parameter.</li> <li>▪ <b>Type</b> – Select a type from the drop-down.</li> <li>▪ <b>Extraction Method</b> – Select a method from the drop-down to determine how data is extracted from the output of a node. If you select REGEX, XPATH, or JSONPath, you must provide a valid rule in the <b>Format</b> field.</li> <li>▪ <b>Format</b> – Enter a valid rule if you selected REGEX, XPATH, or JSONPath as an extraction method.</li> <li>▪ <b>Inject into msg as</b> – Enables you to inject the ViNO output parameter into the Node-RED msg object using the identifier entered in this field.</li> <li>▪ <b>Private</b> – This checkbox prevents other nodes from being able to use parameter mapping to reference this output parameter.</li> </ul>
<b>Edit Parameter</b> button	Opens the <b>Edit Output Parameter</b> window enabling you to edit parameters for a specific node. Highlight a node entry, then click Edit Parameter. Or, double-click a parameter. Includes the same fields as the <a href="#">Add Parameter</a> button.

Component	Description
<b>Remove Parameter</b> button	Enables you to remove user-added parameters. You cannot remove built-in parameters.
<b>Restore Defaults</b> button	Restores the default value (as displayed in the <b>Parameter Value</b> column) for the selected parameter.
<b>Parameter Name</b> column	Displays the name of the output parameter.
<b>Parameter Description</b> column	Displays a description of the output parameter if one was entered in the <a href="#">Description</a> field.
<b>Parameter Key</b> column	Displays the output parameter key for the node.
<b>Parameter Format</b> column	Displays the rule entered in the <a href="#">Format</a> field. This value displays when REGEX, XPATH, or JSONPath was selected as an extraction <a href="#">method</a> .
<b>Extraction Method</b> column	Displays the method as set in the <a href="#">Extraction Method</a> field.
<b>Inject into msg as</b> column	Displays the identifier entered in the <a href="#">Inject into msg as</a> field.

## Editing Parameters

To edit a parameter, select it from the **Input Parameters** section of the [Property Edit](#) window, and then click **Edit Parameter** or double-click the parameter name. Figure 24 shows an example of the **Edit Input Parameter** window.

**Figure 24. Edit Input Parameter Window**



The screenshot shows the 'Edit Input Parameter' dialog box. It contains the following fields and controls:

- Name:** A text field containing 'Use Floating IP Address For Management Network'.
- Key:** A text field containing 'openstack\_use\_floating\_ip\_addresses'.
- Description:** A text area containing 'Boolean indicating whether a Floating IP address should be reserved for and applied to the management network for the'.
- Type:** A dropdown menu set to 'Boolean'.
- Parameter Source:** A dropdown menu set to 'Node Configuration'.
- Value:** A dropdown menu set to 'False'.
- Display Order Index:** A text field with up and down arrow buttons on the right.
- Optional:** A checkbox that is currently unchecked.
- Final:** A checkbox that is currently unchecked.
- Buttons:** 'Cancel' and 'OK' buttons at the bottom right.

The **Edit Input Parameter** window includes fields and checkboxes that control the behavior of parameters and the **Value** field in which a fixed value can be set. Because this parameter is built into the action, some fields and options are disabled. When the parameter is configured to come from constants or is mapped from another node, additional fields appear that enable you to configure which constant to lookup or to which node and parameter to map.

- The **Final** checkbox controls whether the value can be overridden at activation time. When mapping from another node, the **Final** checkbox cannot be selected.
- The **Optional** checkbox controls whether the service can activate with or without that parameter.

## Using Parameter Mapping

Parameter mapping enables you to reference a parameter value from a node that had been activated previously. For example, you may reference a VM ID in a deactivation step in order to delete the VM. When mapping to previous nodes, map to output parameters (not input parameters).

**Caution:** Do not use the **Import** or **Export** options ([Menu](#) icon drop-down) to share or save a project. These functions lose parameter mappings.

## Using Parameter Mappings and Conditionals

ViNO conditional nodes (conditional start and conditional end) present a unique challenge when it comes to parameter mapping. This is because the conditional statement is not evaluated until activation time and its input values may be based on previous steps or user input. The result of the evaluation cannot be known ahead of time. Therefore, there is no guarantee when designing a service that the steps in one particular branch will be executed. For that reason, nodes outside of a particular conditional branch are not permitted to map to parameters on nodes within a conditional branch. However, you may need to use output parameters from steps within a conditional later in the service flow after the conditional branch has completed.

To accomplish this, configure the [conditional end](#) node with a unique type of output parameter that handles this situation in a safe manner. Output parameters in the **conditional end** node are configured to map to parameters in both the true and false branches. This functionality imposes the limitation that both branches must contain a parameter that logically makes sense to expose in both the true and false scenarios.

## Using Parameter Wrapper Injection and Extraction

The **parameter wrapper** node has several different purposes. The **Mode** setting controls the behavior of the node.

When a **parameter wrapper** node has the mode set to **Parameter Injection/Extraction**, the node injects input parameters into the Node-RED message object so that the parameters can be accessed with **msg.parameter\_key**. This node also inspects output parameters defined on it and tries to extract the parameters from the Node-RED message object. This functionality is necessary to facilitate the use of non-ViNO nodes in a service flow. The **parameter wrapper** node can also be used to pass parameters into subflows.

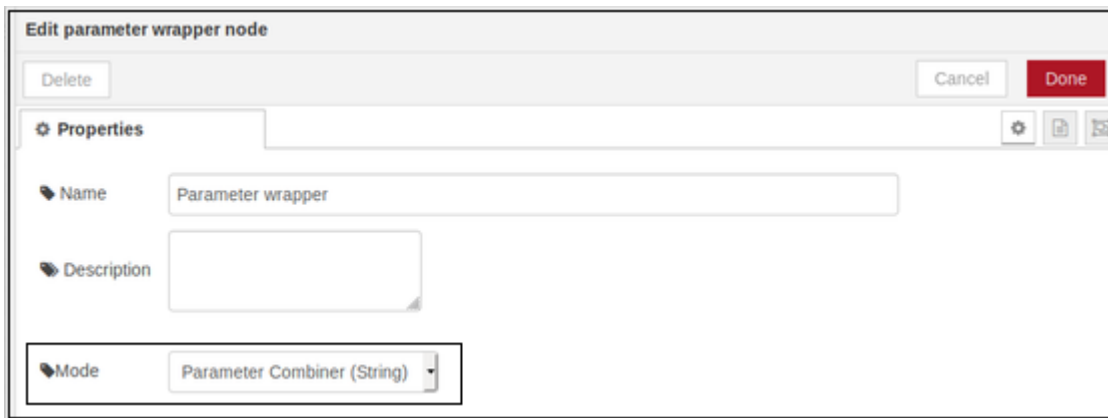
## Using Parameter Combiner Modes

The **Parameter Combiner** modes (**Mode** drop-down in the **Property Edit** window) provide a simple way to transform individual scalar parameters in to a single list type parameter. Input parameters added in this mode are included in the **Combined Output List** parameter. The **Parameter Combiner** modes available in the **Mode** drop-down include:

- **Parameter Combiner (String)**
- **Parameter Combiner (Number)**
- **Parameter Combiner (Boolean)**

Figure 25 shows the **Mode** field in the **Property Edit** window.

**Figure 25. Mode Field in Property Edit Window**





## Passing Parameters Between a Parent Flow and Subflow

Parameters can be passed between a subflow and the [parent flow](#) that is using it. When accessing the output parameters of nodes within the subflow from the parent flow, ViNO combines them and presents them as output parameters of the subflow node within the parent flow. However, passing parameters into the subflow is complex because nodes within a subflow cannot use the ViNO parameter mapping functionality to directly map to any parameters in a parent flow. This is because subflows can be used in different flows. To work around this, use the [parameter wrapper](#) node with the **Mode** set to **Parameter Injection/Extraction**.

At the start of the subflow, add a **parameter wrapper** node and define output parameters to contain any values you want to pass into the subflow from the parent flow.

Nodes within the subflow can now map to those output parameters as needed. To pass those parameters into the subflow:

1. Add another **parameter wrapper** node immediately in the parent flow *before* the subflow.
2. Define input parameters on the **parameter wrapper** node that exactly match the parameter keys from the output parameters created within the subflow.

When the service is activated, the two input parameters in the parent flow are put into the message object, which gets passed into the subflow where they are immediately extracted into two corresponding output parameters. When a flow uses a subflow that requires certain parameters to be passed in and the parent flow fails to do so, the subflow triggers an activation failure due to missing input.

**Note:** When the subflow requires the parameters for use in built-in nodes, you can omit the **parameter wrapper** node at the start of the subflow. This is because the parent flow has already placed the parameters on to the message object so that they can be directly accessed by built-in nodes as is.

Figure 26 shows an example of the Edit Property window for a parameter wrapper node.

Figure 26. Using a Parameter Wrapper Node

Edit parameter wrapper node

Delete

Cancel

Done

Properties

Name

Inject Status Message Parameter

Description

Mode

Parameter Injection/Extraction

Status Message Options

Input Parameters

Add Parameter

Edit Parameter

Remove Parameter

Restore Defaults

Unset Parameter

Search:

Parameter Name	Parameter Description	Parameter Key	Parameter Value	Display Order
statusMessage	The value of this parameter will be printed in the status later in the graph	statusMessage	Parameter Wrapper injection success!	

Showing 1 to 1 of 1 entries

Previous

1

Next

Output Parameters

Add Parameter

Edit Parameter

Remove Parameter

Restore Defaults

Search:

Parameter Name	Parameter Description	Parameter Key	Parameter Format	Extraction Method	Inject Into msg as
No data available in table					

Showing 0 to 0 of 0 entries

Previous

Next

## Using Loops

It may be necessary to repeat a step or a group of steps in a service multiple times. The number of times to repeat these steps may be variable or the input for each iteration may need to vary. The **loop start** and the **loop end** nodes provide two ways to perform a set of actions repeatedly:

- A basic loop that repeats a fixed number of times
- A **for each** loop that iterates over a list and exposes a single element from that list upon each iteration.

Refer to the following sections for information on using loops:

- [Using Basic Loops](#)
- [Considerations for Each Loop](#)
- [Using Input Parameters in a Loop Step](#)
- [Using Parameter Mapping and Loops](#)

## Using Basic Loops

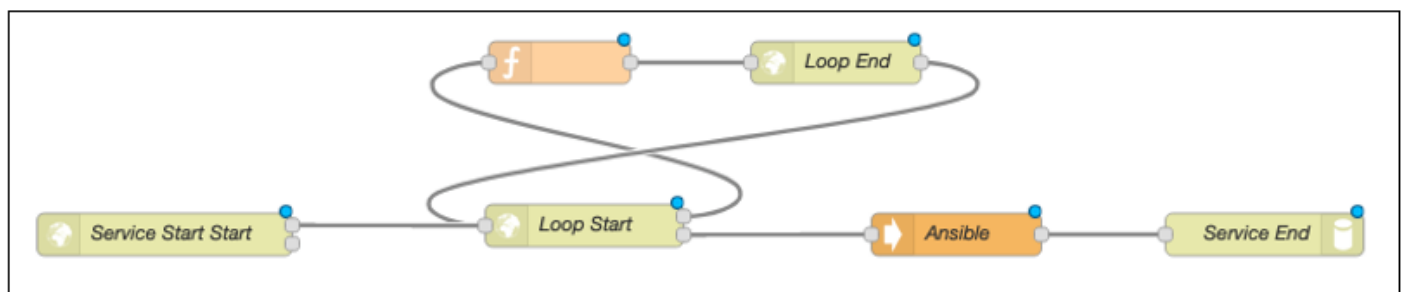
A basic loop begins with a **loop start** node that contains a single input parameter (Iterations) that controls the number of times the loop is to repeat. The [parameter source](#) can be configured using the same sources available to other input parameters. The **loop start** node contains a single output parameter that always contains the current iteration number of the loop starting from 0. A node within a loop can map any of its parameters to the Iterations input parameter in the **loop start** node.

The **loop start** node has two output ports:

- The top output connects to the start of the steps that are included in the loop (the loop body).
- The bottom output connects to the next step to take after the loop has run to completion.

As shown in Figure 27, at the end of the loop body, the last node must connect to a **loop end** node that must connect back to the input of the **loop start** node.

**Figure 27. Example of a Basic Loop**



# Considerations for Each Loop

In addition to the basic loop mode, ViNO provides another looping mode that resembles the **for each** construct of many programming languages. This loop mode operates identically to the basic loop mode, but rather than taking a number that controls the number of iterations in the **start** node, the loop takes a list type input parameter called **Input List**. The type of this list is determined by which variation of the **for each** mode is selected (String, Number, or Boolean). There is also a single output parameter (**Current Value**) that contains the value for the current iteration of the loop.

Figure 28 shows the **loop start** node with the **Input List** parameter and the mode set to **For Each Loop** with a string input list. The input list contains three values (VM1, VM2, and VM3), indicating that the loop is to run a total of three times. The **Current Value** output parameter has its value set to VM1 on the first iteration, VM2 on the second iteration, and VM3 on the third iteration.

Figure 28. Using Input List in a Loop

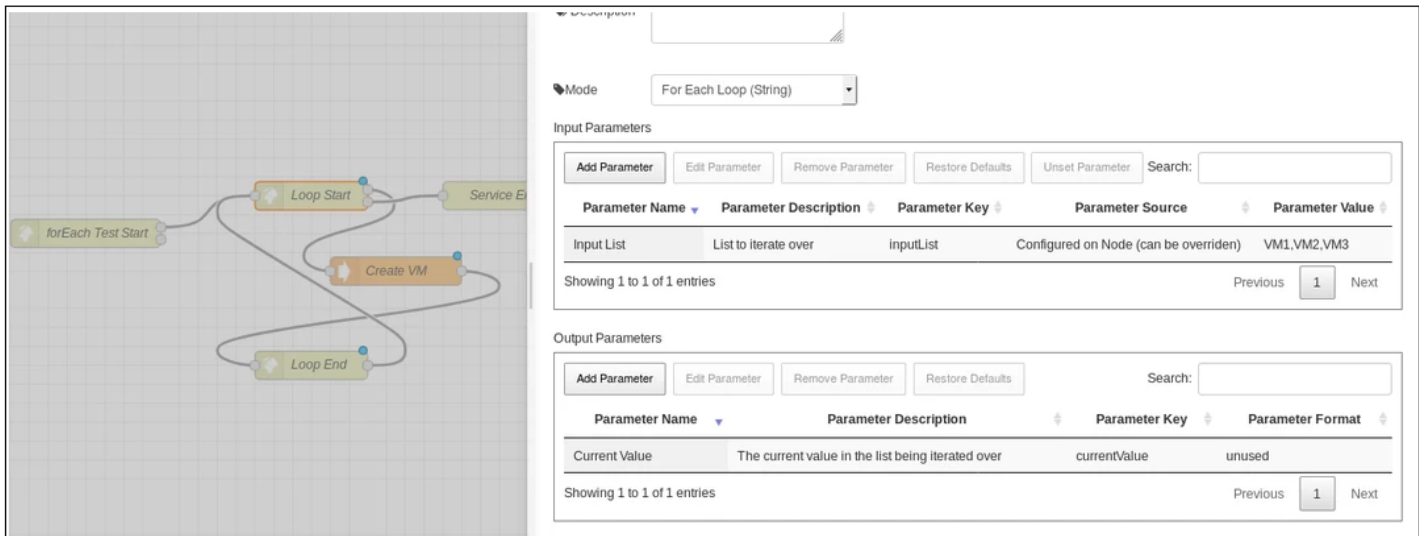
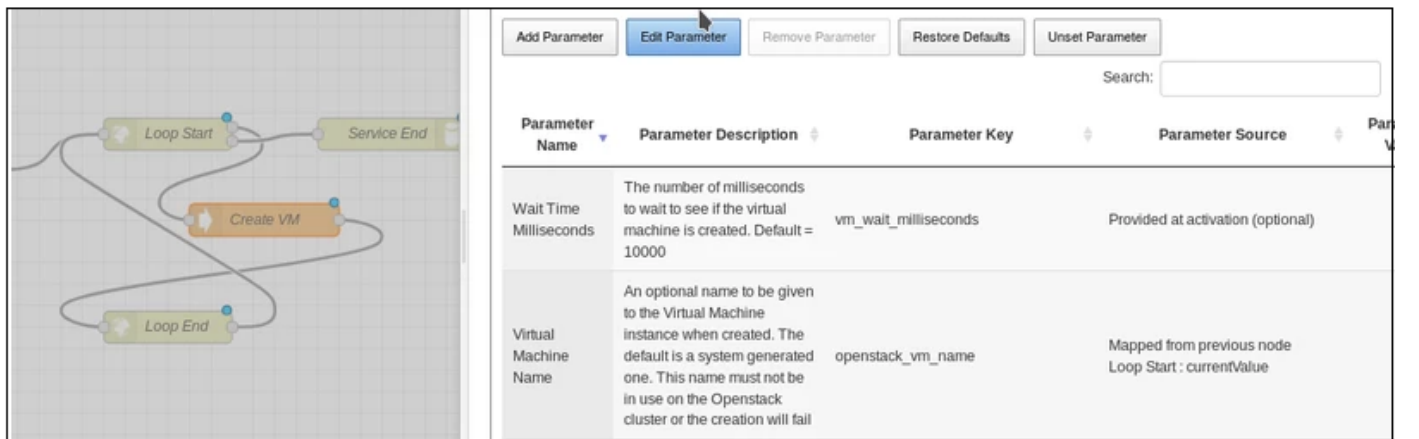


Figure 29 shows the **Create VM** node with the **Virtual Machine Name** input parameter value mapped to the **Current Value** output from the preceding **loop start** node. Because that parameter changes with each iteration, the end result of this loop would be three VMs with the names VM1, VM2, and VM3.

Figure 29. Loop Start Node Example



## Using Input Parameters in a Loop Step

To provide flexibility, the input parameters of a node have a unique behavior when the parameters are located inside the body of a loop. When a parameter is configured in a node and not overridden, it uses that fixed value for every iteration of the loop. However, if the input parameter is provided at activation time, its type is converted into a corresponding list type (for example, a string parameter becomes a **stringList parameter**) and the step uses the next value available in that list upon each iteration.

For example, if the “Create VM” step was placed inside a loop, the **VM Name** input parameter would be converted to a **stringList**. The loop could then be configured to run for three iterations and the **VM Name** parameter could be given a list of three different names. This would result in the creation of three virtual machines, each with one of the names provided in the **VM Name** input parameter list. The loop would also wrap around while iterating through input lists, if necessary. Therefore, if a loop runs six times, but a particular input list only contains two values, the loop alternates between those two values for that particular parameter.

## Using Parameter Mapping and Loops

For nodes that map to parameters of nodes within the same loop, [parameter mapping](#) works as expected and the mapping copies the value from the node activated within the current iteration. However, if a parameter in a node outside of the loop maps to a parameter in a node within the loop, it always receives the value used or created on the last iteration of the loop.

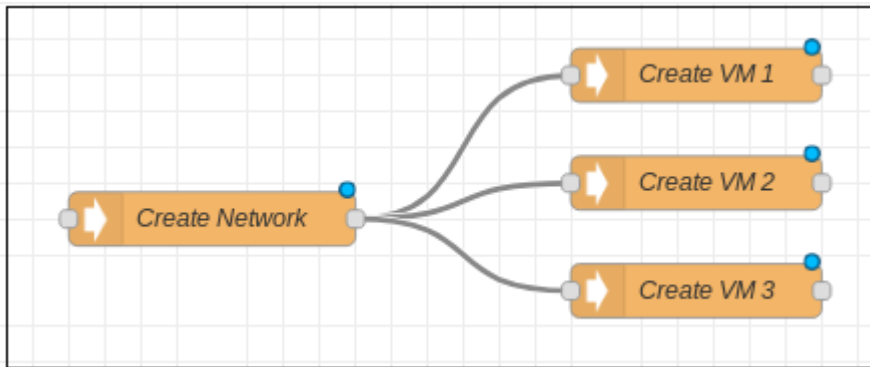
## Using Parallel Execution

Node-RED and ViNO natively support executing independent steps in parallel. Each node output can have multiple connections to subsequent nodes. To create a parallel branch, create links from one node output to two or more nodes.

**Note:** Because nodes within parallel branches cannot map to nodes in other branches, parallel logic can only be employed when a series of actions are truly independent.

As shown in Figure 30, after the network is created, the service splits into three branches and begins creating three VMs in parallel.

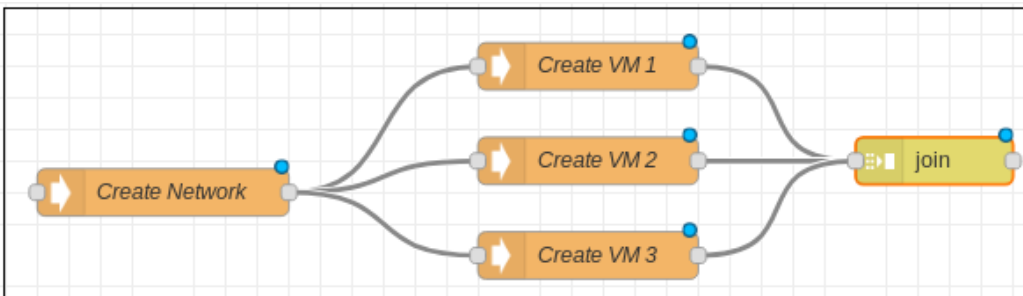
**Figure 30. Example of Creating a Parallel Branch**



## Joining Parallel Branches

Before a service completes, it must join the parallel branches together at some point. This is a point that halts execution of the service until all branches are finished completely. The last node of each branch must be connected to a single **join** node as shown in Figure 31.

**Figure 31. Example of Joining Parallel Branches**



The **join** node must then be configured to expect message from the number of branches that were created. To accomplish this:

1. Open the **Property Edit** window for the **join** node. See Figure 32.
2. Select **manual** mode.
3. (Optional) Select values for the following fields:
  - **to create**
  - **joined using**
  - **After a timeout following the first message**
4. Enter the number of branches being joined in the field **After a number of message parts**.
5. Click Done.

After the branches have been joined, the service can continue as normal and all nodes within the branches can be referenced using [parameter mapping](#).

**Note:** Failure to configure the **join** node correctly results in undefined behavior.

**Figure 32. Example of Join Node Properties**

The screenshot shows the 'Edit join node' dialog box. At the top, there are three buttons: 'Delete', 'Cancel', and 'Done'. Below these is a section titled 'node properties' with a dropdown arrow. The configuration options include: 'Mode' set to 'manual'; 'Combine each' set to 'msg. payload'; 'to create' set to 'a String'; 'joined using' set to 'a\_z \n'; and 'Send the message:' with three radio button options. The first option, 'After a number of message parts', is selected and has a value of '3' in a text box. The second option, 'After a timeout following the first message', has a value of 'seconds' in a text box. The third option, 'After a message with the msg.complete property set', is not selected. At the bottom, there is a 'Name' field with a placeholder text 'Name'.

# Creating a Service Deactivation Flow

To create a service deactivation flow, use the [service endpoint](#) node. This node has two outputs available:

- Top output – Used for the service activation flow.
- Bottom output – Meant to be connected to a corresponding flow that would deactivate an instance of the service.

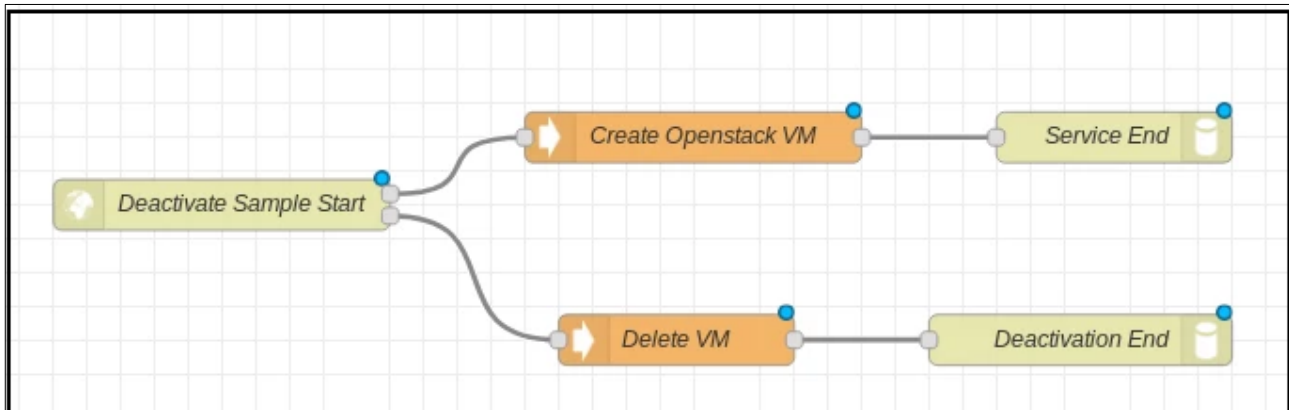
**Note:** A service flow must have a deactivation path defined in order for the service to be deactivated.

A service deactivation flow works similarly to how the activation flows work with several key differences:

- At the end of a deactivation flow, the last node must be connected to a [deactivation endpoint](#) node rather than a [service endpoint](#) node.
- Nodes within the deactivation flow can map to parameters from a node in the activation flow. This enables the deactivation flow to access the input and output information that was used or created during a particular activation as part of the deactivation logic. For example, if a VM was created in an activation flow, the flow would need the VM identifier output as part of that step in the deactivation flow to delete the correct VM.
- The deactivation flow ignores errors encountered during deactivation and continues deactivating. This is to facilitate easier rollback logic and ensure that ViNO tears down as much of the service as possible, even if some resources are missing or have been modified since the service activation. Errors in the deactivation flow are still reported in status messages.

Figure 33 shows an example of a simple service with a corresponding deactivation flow.

**Figure 33. Service with Corresponding Deactivation Flow**





When a deactivation is called on an instance of a service, the inputs and outputs from the activation are retrieved and available for use. Figure 34 shows a deactivation flow that maps to the VM ID output parameter from the activation flow, which deletes the VM associated with that particular activation.

**Figure 34. Deactivation Flow Example**

The screenshot displays the ViNO interface. On the left, a flow diagram shows a sequence: 'Deactivate Sample Start' (green node) connects to 'Create Openstack VM' (orange node), which then connects to 'Delete VM' (orange node). The right panel, titled 'Edit vino-driver-openstack node', contains configuration options for the 'Delete VM' node. It includes a 'Delete' button, a 'Cancel' button, and a 'Done' button. Below these are 'node properties' and a dropdown menu set to 'OpenStack' with 'Delete Virtual Machine' selected. The 'Input Parameters' section features buttons for 'Add Parameter', 'Edit Parameter', 'Remove Parameter', 'Restore Defaults', and 'Unset Parameter', along with a search bar. A table lists the parameters for the node:

Parameter Name	Parameter Description	Parameter Key	Parameter Source	Parameter Value
Virtual Machine Identification	The ID of the Openstack VM to destroy	openstack_vm_id	Mapped from previous node Create Openstack VM : openstack_vm_id	
Openstack Username	The username to use when authenticating	openstack_username	From constants openstack/user	
Openstack Project Name	The project to scope to	openstack_project_name	From constants openstack/projectName	
Openstack Password	The password to use when authenticating	openstack_password	From constants openstack/password	
Openstack Host	The host that is running the Openstack cluster	openstack_host	From constants openstack/server	
Openstack Domain Name	The domain to scope to	openstack_domain_name	From constants openstack/domainName	
Openstack Auth URL	The endpoint URL for authenticating and running web service commands	openstack_auth_url	From constants openstack/authUrl	

At the bottom of the panel, it indicates 'Showing 1 to 7 of 7 entries' and includes 'Previous', '1', and 'Next' navigation links.

# Error Handling

Due to the complexity of certain service activation flows, errors may occur. For example, if there are invalid or missing inputs or unreachable systems, service activations will fail to complete successfully. It is important to handle cases where failure is a possibility. Node-RED and ViNO include the ability to automatically handle errors using the:

- **catch** node
- **throw** node

## Using the Catch Node

If a node encounters a problem, it triggers an error and stops passing messages to subsequent nodes. The Node-RED **catch** node (located under the [input](#) category) can intercept these errors and start executing a flow from an entirely independent location. You can configure the **catch** node to either catch errors from anywhere on the current flow or to only catch errors from specified nodes. This functionality provides flexibility for handling errors.

However, it is necessary to communicate to ViNO in the case of Node-RED errors, which is the function of the [service failure](#) node. When a **service failure** node executes, ViNO records that the service activation was a failure and reports this in a status message. When a service encounters an error that is not directed to a **service failure** node, ViNO does not record the activation attempt. Therefore, services should have a minimum of one **catch** node connected to a **service failure** node. Both nodes can be placed anywhere on the same tab as the activation flow.

The **catch** node can also be configured to catch errors only from specified nodes. This can be useful in defining rollback logic for a service. See [Adding Rollback Capability](#).

## Using the Throw Node

The ViNO [throw](#) node simply throws an error upon reaching a node. This node functions as a valid end (such as the **service endpoint** and the **deactivation endpoint** nodes) to a ViNO service, which allows the service to complete even in the case of an error.

## Adding Rollback Capability

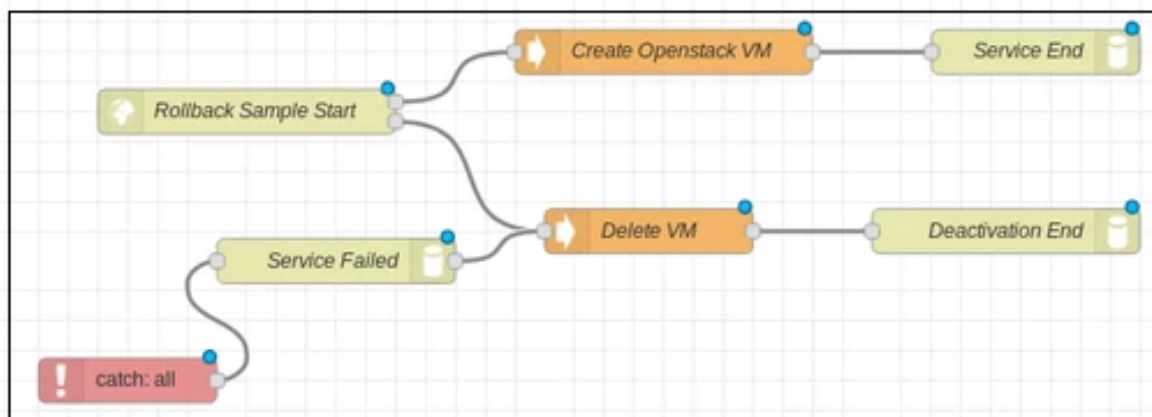
If a service runs into an error at some point in the activation process, it may be necessary to clean up or revert some of the actions that completed prior to the failure. This can be done by combining the error handling capabilities (see [Error Handling](#)) with the deactivation flow.

The **service failure** node contains an output that may optionally be connected to a node in the deactivation flow. If a service encounters an error and ViNO detects that the **service failure** node is connected to the deactivate flow, it enters rollback mode and indicates that it is now trying to clean up the failed activation. The easiest way to implement rollback is to:

1. Use a single **catch** node and select **all nodes** from the **Catch Errors From** drop-down.
2. Connect it to a **service failure** node.
3. Connect **service failure** node to the first node of the deactivate flow.

Figure 35 shows how to implement a rollback.

**Figure 35. Implementing a Rollback**



Because nodes in a deactivation flow ignore errors, a rollback always runs to completion even if some of the actions attempting to be deactivated were never activated.