PagerDuty
Process Automation On Premise
Cluster Deployment Guide

July 2023

Table of Contents

Introduction

PagerDuty Process Automation (PA) is an enterprise-grade commercial software application based on the open source Rundeck project. PA can be deployed in a data center or the cloud and can manage resources in local or remote environments.

One of the key features that separates Process Automation from Rundeck is that PA can be deployed in a clustered mode, providing high availability (HA) functionality and failover for running workflows. The center of the PA architecture is the Cluster, consisting of multiple parallel instances[1] running on Linux or Windows servers. The cluster members will take input from users and run automation on nodes in the environment.

Cluster instances have some unique files on each server (server logs and configuration files) but most actions are based on shared resources, such as an external relational database and a common location to store execution history logs as jobs occur. Other than execution logs, most data about the Process Automation environment is stored in the shared database.

In conjunction with the back-end database, the Cluster provides load balancing of job executions, spreading the load of running jobs among available cluster instances at runtime. Most customers will still deploy a load balancer in front of the cluster to control and balance load in terms of access to the system interfaces. This is often the point in the architecture where SSL may terminate.

Access to the Process Automation interface is controlled with authentication that is customizable and should be considered as part of the deployment architecture. Though there is a built-in user and group management system in PA, most enterprises choose instead to integrate with their existing enterprise directory system, typically something like LDAP, Microsoft Active Directory or an SSO (Single Sign-On) application.

The last key piece of the general PA architecture that should be considered are the nodes to be managed by the automation system. Nodes are typically servers or network infrastructure that will be targeted by the cluster instances as jobs are run. Workflows or jobs will have some steps that are run on the cluster instances themselves and other steps that are run on each of several nodes targeted in the job. Communication between an instance and a node allows for customization but generally involves common protocols such as SSH or WinRM.

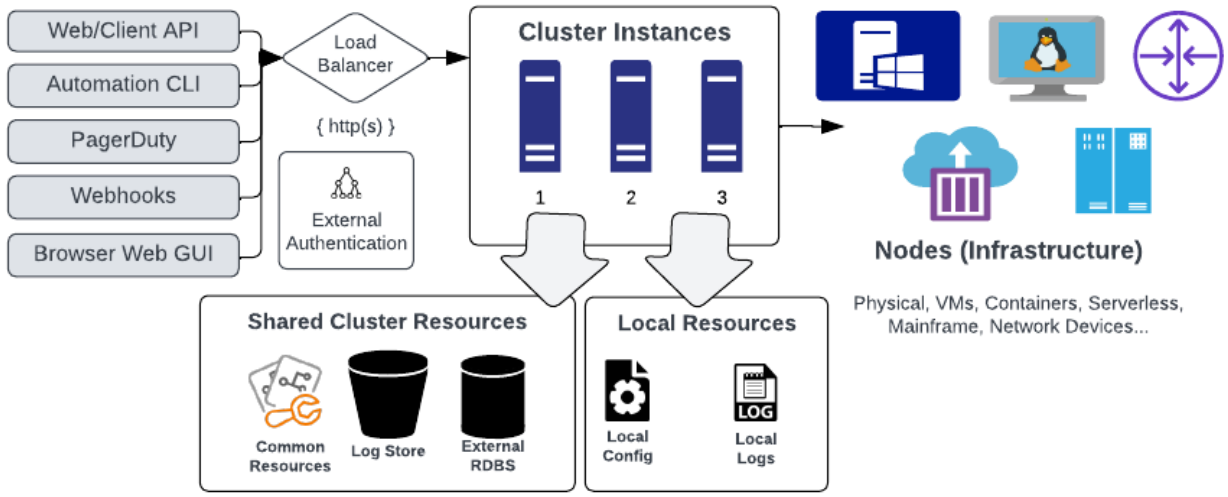
One new element in the PA architecture is the Enterprise Runner. Runners can be added to a deployment as a way of allowing secure communication to remote networks that might otherwise be inaccessible. Generally, one or more Runners are installed in each remote network to manage communication to nodes in that network. Rather than the cluster instances reaching out to the remote nodes directly, each Runner polls the cluster to find assigned work and dispatches Job steps to nodes within their own network.

[1] An instance is a physical or virtual server that is a member of the PA Cluster.

Cluster Architectures

Basic Cluster

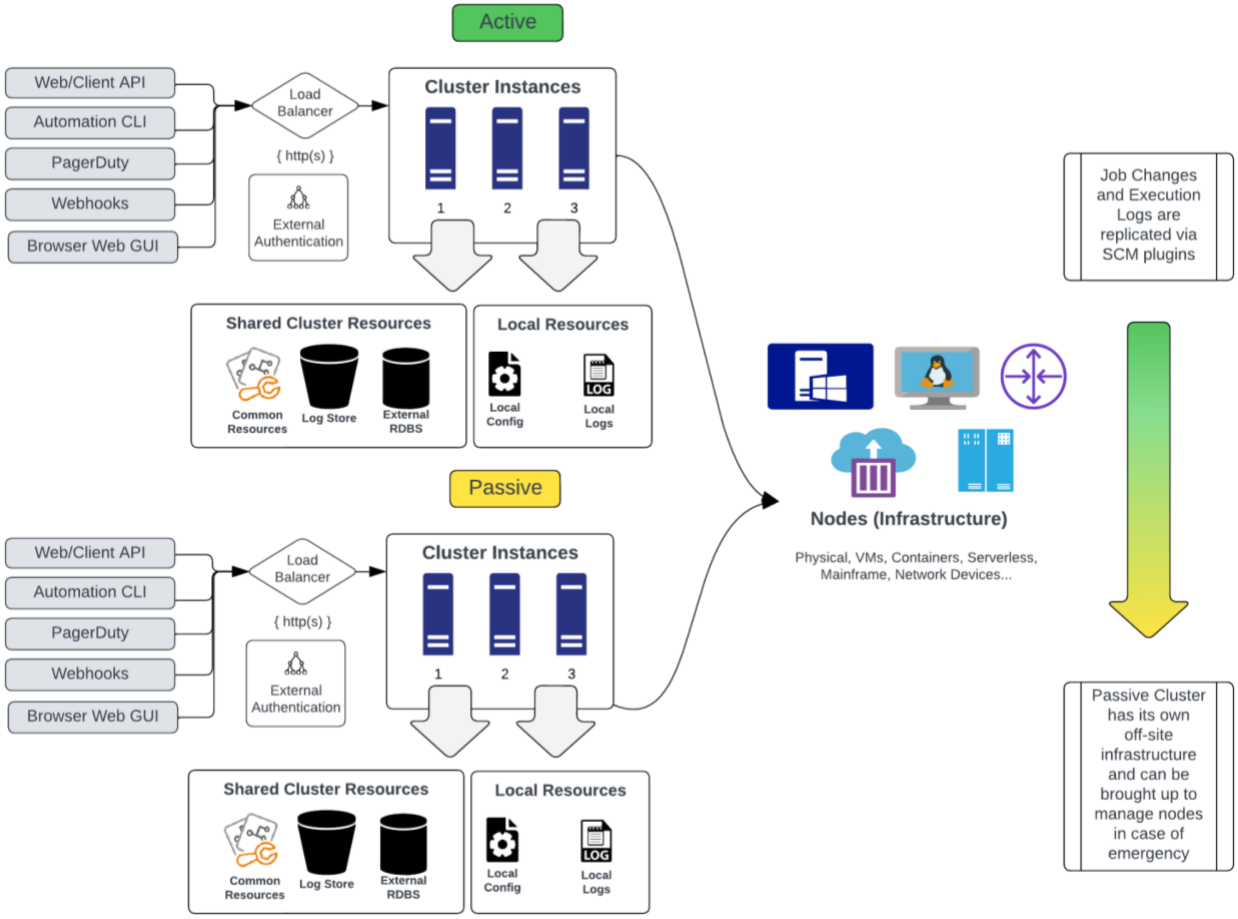
A cluster consists of as few as two server members. However, best practice is to start with a Cluster of three to ensure multiple servers will be available even when applying patches to a Cluster member. All server instances must connect to shared resources such as a backend database. It is recommended that all instances be largely homogenous in terms of server OS and resources. By default, whichever instance a user connects to via the UI will run any jobs initiated by that user. If server access is being managed through a load balancer, this will likely work fine for small clusters.



A basic PA Cluster, with three cluster members

Active and Passive Cluster

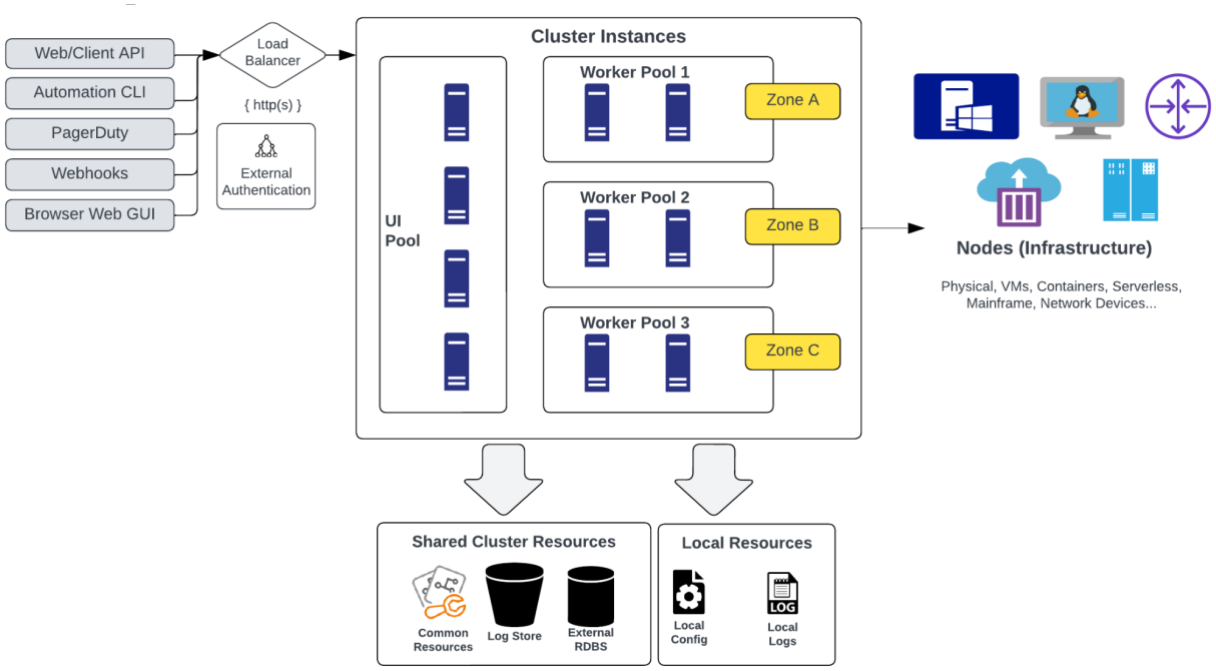
For enterprise organizations who are proactively planning for Disaster Recovery, it is possible to deploy two PA clusters in an Active/Passive mode. Each cluster has its own infrastructure and database as well as being deployed in a separate physical or network location. In this scenario, the Passive cluster can have its Jobs and its Execution History kept up to date by included plugins that will sync changes from the Active cluster as updates occur. In this way, the Passive cluster can be made active with up-to-date Jobs and logs in case of disaster.



Two Clusters configured with one Active and one Passive for Disaster Recovery

Larger Clusters with Specialized Servers

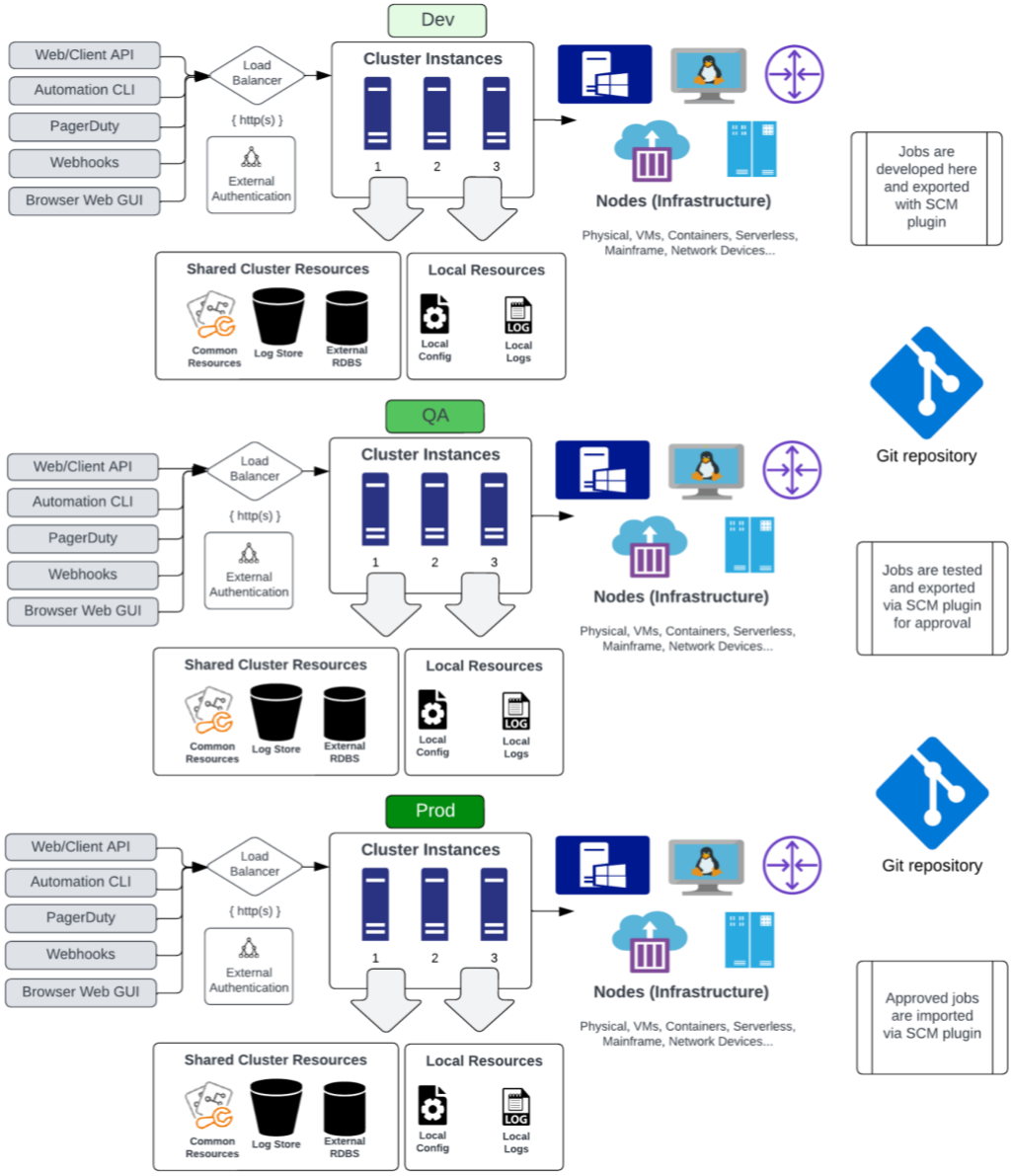
For organizations that have a particularly large load of Jobs to run, it is possible to deploy a Cluster with a larger number of instances. In these cases, it can be beneficial to set specialized roles for instances in the Cluster. Typically, this involves separating User Interface (UI) servers from those that will do the work of running Jobs. Handled this way, the UI servers can maintain high response times for users while the Worker servers are focused on efficiently processing concurrent Jobs. It would also be possible to have a further separation of duties within the Worker servers based on specialized Jobs or resources. For example, a subset of the Worker servers might be enabled for Ansible and would run all Ansible-specific Jobs.



A single larger cluster with 10 instances (4 handling user requests and 6 running jobs)

Multiple Clusters Deployed to Manage Jobs as Code with Source Code Management

For organizations that are far along the maturity curve with automation, it can be very beneficial to manage PA jobs as code. By deploying separated clusters for each environment, new jobs can be authored in the Dev environment and then submitted to a Git repository to go through an approval process before being promoted for testing and eventually Production. The number of environments and their names would be based on the organization's processes for source code management (SCM).



Multiple clusters deployed to manage Jobs using source code management

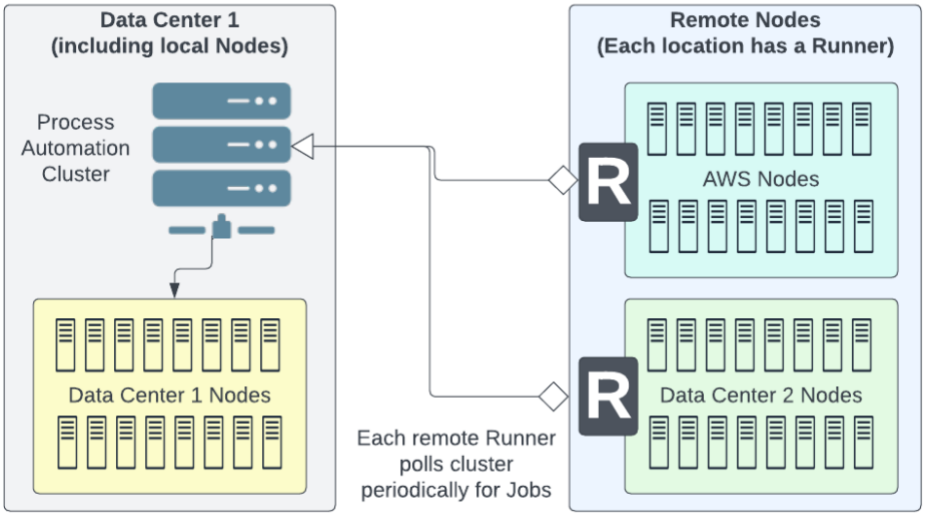
Deploying multiple clusters by environment in this way can also be used to manage different sets of nodes for access separation (Dev environment manages Dev nodes, QA manages QA nodes, Prod only manages production environments). Architecturally, these would look the same.

Notes on Cluster Architectures

The included examples are not the only ways to deploy a Cluster but are intended to provide examples of the most common variations. Though documented individually, these examples could be mixed as needed. For example, an Active/Passive Cluster configuration could include more than three instances with specialized roles (or not).

Notes on accessing remote nodes using Enterprise Runners

Enterprise Runners are a recent addition to the PagerDuty Process Automation architecture. They are primarily used as a method for orchestrating automation in secure, remote environments where direct access to infrastructure and services is restricted. This is accomplished by using a "reverse proxy" architecture: Runners regularly query (outbound only) the Process Automation Cluster for tasks. Runners are assigned to one or more Projects, and then Jobs within those Projects can use the Runner to dispatch the Job steps to be executed through the Runner in the remote environment.



Example showing Runner architecture (Each remote Runner could be a Runner set)

When Runners are enabled for a job (based on job and Runner tags), the execution of Job steps will be delegated to a specific Runner for management. This model offers resiliency as multiple Runners could be enabled for a particular Project so that each Job can be assigned to a specific Runner during high usage windows.

Process Automation Supported System Requirements

Included below are the basic system requirements to be considered when deploying PagerDuty Process Automation. Following these requirements will allow a commercial customer to receive support from the PA technical support team while under contract.

Where relevant, the requirements include links to relevant documentation related to installation and/or configuration.

Server Operating System

All servers in the Cluster should be of the same OS family (Linux or Windows).

Red Hat Enterprise Linux/Amazon Linux/Oracle Linux
Ubuntu/Debian Linux
Windows Server

Server (and Runner) Profile

Recommended:	**Minimum**:
32GB RAM	16GB RAM
(24GB JVM Heap)	(12GB JVM Heap)
8 CPUs per instance	4 CPUs per instance
Equivalent to m4.2xlarge in AWS EC2	*Equivalent to m4.xlarge in AWS EC2*

Java

Java 11 installed on each instance/Runner

Install Method

.rpm, .deb, Java servlet (.war) or Docker

Back-end Database

MariaDB/MySQL 8+ or PostgreSQL

Admin Access

Root (or Administrator on Windows) is not required or recommended. Using a dedicated user account such as "rundeck" is recommended. If root access is needed, please set up the dedicated user to have access via sudo.

Log Store

S3-compatible object store or Azure blob storage

Default Network Ports

For access to interface	4443 (https), 4440 (http)
For access to most nodes	22 (ssh)
For access to Windows nodes via WinRM	5986 (https), 5985 (http)

Which ports are used often changes with configuration of the load balancer but all these ports are configurable as needed.

Browser

Accessing automation typically requires an HTML5 compliant browser. Currently supported versions of Mozilla Firefox or Google Chrome are recommended.

Installation Checklist (in approximate order)

Server/Cluster Setup

Cluster Size

Strictly speaking, it is possible to have a Cluster with as few as two server members. However, best practice is to start with a Cluster of three to ensure multiple servers will be available even when applying patches to a Cluster member.

Though many customers will start with 3 instances in the Cluster, how many instances are needed over time is not a simple question to answer. In general terms, determining the number of instances is dependent on the largest number of concurrent Jobs that would be running and the resources needed for that.

Each Job will be managed at runtime by a single server in the Cluster, including connections to all Nodes for that Job run. Effectively, that means that for a job targeting 10,000 nodes, the server will keep running the Job until all those nodes have executed the Job (including server-side Job steps) and reported back success or failure. How many nodes will be processed in parallel will be dependent on Job settings as well as threads and other resources available to the server. Most Jobs will be considerably smaller than this example but it is worth looking at the expected peak times for Job execution (maintenance windows, for example) to estimate how many Jobs might be running and how many Nodes could be targeted in a given time-frame.

Database Configuration

A fresh install will include H2, a local sandbox database. This is fine for testing but is not recommended for production deployments. Instead, cluster instances should be connected to an external database from the supported databases listed above. All cluster members must connect to the same active database instance, which will be accessed regularly by cluster members. It is vital that all instances can connect to the external database with low latency. In addition to the general instructions for connecting a single server to the database, the first cluster member installed and connecting to the database will establish encryption settings that must be copied to the configuration files for remaining cluster members. If the encryption settings aren't copied, only the first server will successfully maintain a connection to the database.

User Authentication and Authorization

All users who attempt to login to the automation web interface must be authenticated to determine who they are and what they are entitled to. For non-production or test instances, this can be handled with built-in user and group management. However, most commercial customers will configure Process Automation to authenticate users using their enterprise's directory system, such as LDAP/Microsoft AD or an SSO provider. For SSO, bear in mind that the SSO plugin currently supports SSO that uses OpenID v2 and doesn't support SAML. When a user successfully authenticates, their group memberships are shared with Process Automation and those groups are reviewed to determine what projects they can access and what they can do within those projects.

Load Balancing in front of Cluster

Though the Process Automation cluster will manage loads related to job execution and node communication, best practice is to enable a load balancer in front of the cluster to manage which cluster instances will respond to user requests and to balance that load as needed. Recommended/supported options for a load balancer would be AWS ALB or NGINX.

Centralized Resources and Logging

For a clustered deployment, it is necessary to set up a shared location to store job execution logs, which would otherwise only be stored locally on the server where the Job runs. This storage should be configured to use an AWS S3 bucket for logs or an S3-compatible object store, such as Minio. When configuring the S3 log storage plugin, be sure to use `com.rundeck.rundeckpro.amazon-s3` in place of `org.rundeck.amazon-s3`. Doing so will enable checkpoint log storage which enables viewing execution logs from any cluster member while the execution is running.

There may be other resources that need to be shared between cluster instances in which case it is advisable to set up an additional shared location for these items. The most common examples that come up would be file-based node sources or files to be referenced in Jobs.

Job Load Balancing

By default, if a Job is triggered by a user, the Job will be executed on the server where the user is logged in at the time the Job is triggered. Scheduled Jobs will be run on whichever server the schedule was created or last modified on.

This Job behavior can be modified to follow other rules if desired. For example, rules could be configured to run in a Round Robin (alternates executions between Cluster members), Load-Balanced or Random pattern instead. This sort of configuration could be used to cause cluster servers to be specialized by routing all Job executions to run on a subset of Cluster members rather than including all servers as is most common.

Configuration of Job execution behavior is controlled by two features, Remote Job Execution and Autotakeover. The former is used, if configured, to manage all job execution behavior. The latter only affects scheduled Jobs.

Cluster Maintenance

There are a few cluster elements worth addressing when setting up a new deployment. There are local logs on each server that are valuable for audit tracking regarding changes to the system over time. These should be configured for log rotation and can be managed with Log4J2. If you have a single cluster, establish a regular process for backup and restore. If you have a dedicated Disaster Recovery cluster, use replication to keep the DR cluster up to date. Lastly, check out the tuning instructions to understand how to tune the servers for greater throughput.

Project Setup

Node Management and Communication

Most Jobs will be targeting one or more nodes. Though nodes are central to the automation solution, they are configured and accessed in the context of a project. In each project, nodes are identified using a node resource model. The resource model either consists of a file with node data defined in it or a plugin that connects to a third-party tool like Ansible, ServiceNow CMBD or Amazon EC2 to gather that data from an existing source.

In addition to metadata about the nodes, the cluster must also be supplied with a method of connecting to those nodes. Node communication is carried out using what is referred to as a Node Executor and files are copied using a File Copier. These two elements for node communication are customizable but most Linux-based nodes will be configured to connect using an SSH-based executor and copier. Windows nodes are usually reached using a WinRM or PowerShell-based executor.

Storing Keys and Passwords

PagerDuty Process Automation includes built-in key storage. Though keys can be stored and accessed at the global level, it is best practice to manage necessary keys and passwords at the project level. This sensitive data is stored in an encrypted form and can be accessed as needed by a plugin or to allow connection to a node. PA also includes plugins that allow customers to store data in three different third-party password managers: Hashicorp Vault, Thycotic Secret Server and Cyberark Key Storage.

Purging Old Executions

By default, job execution history is written locally on the server that runs a Job and also in a shared location for the cluster when that has been configured. That history will be maintained indefinitely if not pruned. Best practice is to configure each project to purge history based on the needs of the project.

Synchronizing Jobs using Source Code Management

The Git SCM plugin can be used to export jobs to a Git-like repository or import Jobs from one. In this way, it is possible to configure a code promotion process to manage Jobs as code. A particular project or cluster is set up for development of new Jobs. Those can be exported to Git for approval and then imported into other clusters for testing and eventually production.

Project Replication Between Servers

PagerDuty Process Automation includes plugins that will replicate jobs and execution history, per project, from a primary cluster to a backup one, such as a Disaster Recovery cluster.

Encrypting Config Properties

Many settings that control how Process Automation will work are stored in the primary configuration file, `rundeck-config.properties`. Though that file and its contents are only accessible to someone who has the dedicated user credentials, many customers wish to encrypt some or all of those settings in the file. Instructions on how to do so are included here.

Configuring Enterprise Runners to access remote nodes

As noted above, Runners are used to manage communication to nodes in remote networks in a secure manner. To utilize Runners, there are four general steps:

System Configuration

1. Enable Runners at the system level
   Though Runners will primarily be used and managed in the context of projects, the feature must first be enabled by a system admin.
2. Create new Runners at the system level
   Before installing in the remote networks, it is necessary to configure each Runner within the web interface. For each Runner to be created, you will be prompted for three things:

A name	Using clear naming can help with Runner management as the number of Runners increases.
Define one or more tags	Tags determine which Runner set will execute which Jobs.
Assign Runner to Projects	Each Runner must be associated with at least one Project in order to be used.

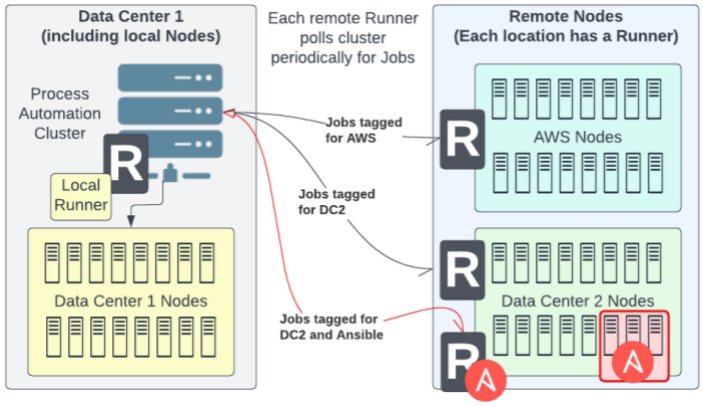
After entering the necessary details, you'll have an option to download a preconfigured jar file for each Runner.

Runner Installation

3. Install Runner software in remote networks for each Runner that has been created
   Though Runners have been defined by this point, it is still necessary to install software in each of the remote locations. Use the jar file downloaded in the previous step. If there is more than one runner in a single remote network, use the unique jar file for each as credentials are included for each.

Configure Jobs for Runners

4. Add Runner tags to Jobs
   New jobs will prompt authors to choose tags which represent Runners or Runner sets by tags. It is important to review existing Jobs to ensure that they have appropriate tags selected as well.



Jobs are assigned to remote Runners based on tags. Jobs on nodes in the cluster's local network can be handled through the Local Runner.

Resources
Advanced Runner Setup Options
Runner Custom Logging
Runner FAQ