

Automated Denodo Deployment using Puppet

submitted by

Debnil Sarkar

Class Roll No: 002210503005

Exam Roll: MCA244005

Master of Computer Application

under the supervision of

Prof. Debotosh Bhattacharjee

in the

Department of Computer Science and Engineering

Faculty of Engineering and Technology

Jadavpur University

2024

Faculty of Engineering and Technology
Jadavpur University

CERTIFICATE OF RECOMMENDATION

To Whom It May Concern

I hereby recommend that the project report entitled “Automated Denodo Deployment using Puppet” has been carried out by Debnil Sarkar (Reg. No.: 163597 of 2022-23, Roll No.: 002210503005, Exam Roll: MCA244005), under my guidance and supervision and be accepted in partial fulfilment of the requirement for the degree of Master of COMPUTER APPLICATION in DEPARTMENT of COMPUTER SCIENCE and ENGINEERING, JADAVPUR UNIVERSITY during the academic year 2023-2024.

Prof. Debotosh Bhattacharjee
Project Supervisor
Dept. of Computer Science and
Engineering
Jadavpur University

Prof. (Dr.) Nandini Mukhopadhyay
Head of the Department
Dept. of Computer Science & Engineering
Jadavpur University

Prof. Dipak Laha
Dean, Faculty Council
of Engineering & Technology
Jadavpur University, Kolkata-032

Faculty of Engineering and Technology
Jadavpur University

CERTIFICATE OF APPROVAL

This is to certify that the project report entitled “**Automated Denodo Deployment using Puppet**” is a bonafide record of work carried out by DEBNIL SARKAR, in fulfillment of the requirements for the award of the degree of Master of Computer Application in the Department of Computer Science and Engineering, Jadavpur University. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein but approve the project report only for the purpose for which it has been submitted.

Signature of Examiner 1

Date :

Signature of Examiner 2

Date :

Faculty of Engineering and Technology
Jadavpur University

**DECLARATION OF COMPLIANCE WITH
ACADEMIC ETHICS**

I hereby declare that this project report entitled "Automated Denodo Deployment using Puppet" is an outcome of my genuine and independent work, as part of my Master of Computer Application program.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, all materials and results that are not original to this work have been fully cited and referenced.

Debnil Sarkar

Registration No.: 163597 of 2022-23

Examination Roll No.: MCA244005

Thesis Title :- Automated Denodo Deployment using Puppet

Signature with Date

ACKNOWLEDGEMENT

I wish to record my heartfelt gratitude towards all the people who have helped me in the completion of this project.

I offer sincere thanks to my internal guide Prof. Debotosh Bhattacharjee and my external guide, Mrs. Shradha Sharma of TI, for their kind support, inspiration, and thoughtful insights.

I am extremely thankful to Texas Instruments and the Department of Computer Science and Engineering at Jadavpur University for providing all the facilities and necessary help towards the development of this project.

I would like to thank all the faculty members of the Department of Computer Science & Engineering of Jadavpur University for their continuous support.

Debnil Sarkar

Class Roll No.: 002210503005

Examination Roll No.: MCA244005

Department of Computer Science & Engineering

Jadavpur University

Contents

1. Introduction	1
1.1 What is Automated Application Deployment?	1
1.2 How does Automated Application Deployment work?	1
1.3 Characteristics of Automated Application Deployment	2
2. Introduction to Denodo	3
2.1 What is Data Virtualization?	3
2.2 Key Features of Denodo	3
2.3 Different Services of Denodo.....	4
3. Puppet- Configuration Management Tool	7
3.1 Core Functionality and Architecture	7
3.2 Key Components or Key Terminologies used in Puppet.....	8
3.3 How Puppet works	10
3.4 Puppet module structure and environment	11
4. Manual vs Automated Deployment process in TI?	12
5. Problem Statement	14
5.1 Background.....	16
5.2 Challenges.....	16
5.3 Project Objectives.....	16
5.4 Specific Goals.....	16
5.5 Expected Outcomes.....	16
6. Design of Solution	15
6.1 Installation and configuration of Denodo Platform on a cluster.....	17
6.2 Installation and configuration module designed in puppet.....	17
7. Results and Analysis	36
8. Scope of Future Work	37

ABSTRACT

Automated application deployment refers to the process of automatically deploying software applications to a target environment using specialized tools or scripts. This automation aims to streamline the deployment process, reduce the risk of human errors, ensure consistency across environments, and enable faster release cycles. In the context of this project, Puppet, as a configuration management tool, is used to automate the deployment of Denodo. Puppet would handle the installation, configuration, and management of the Denodo platform across infrastructure.

Denodo is a data virtualization platform that allows organizations to integrate data from various sources—such as databases, web services, and cloud storage—without the need to move or copy the data physically. It creates a unified view of data that can be accessed in real-time by applications, enabling more agile data management and analytics practices.

Puppet is an open-source configuration management tool for automating software and systems infrastructure management and deployment. Puppet uses a declarative language to specify system configuration, which it then enforces automatically, ensuring that the system remains desired. It can manage infrastructure ranging from small-scale setups to large, complex systems, making it a versatile tool for IT automation.

Automated application deployment benefits significantly from tools like Puppet, particularly in complex environments involving platforms like Denodo. Automation accelerates the deployment process and enhances reliability, scalability, and efficiency in managing and operating software applications.

1. Introduction

1.1 What is Automated Application Deployment?

Automated application deployment is a process that involves using software to automatically deploy and manage applications in various environments, from development through production. This automation minimizes human intervention, making the deployment process more efficient, consistent, and error-free.

1.2 How does Automated Application Deployment work?

Automated application deployment streamlines the process of installing, updating, and managing applications across various environments (development, testing, production). It eliminates manual, error-prone steps, leading to faster releases, enhanced reliability, and better standardization.

The Role of Configuration Management Tools

Configuration management (CM) tools are the backbone of automated application deployment. Here's how they play a critical part:

- **Defining Desired State:** CM tools (e.g., Ansible, Puppet, Chef) allow to codify the desired state of infrastructure and application. This includes:
 - System packages and dependencies
 - Configuration files and settings
 - Application components and their relationships
- **Enforcing Consistency:** The CM tool continuously monitors your environments. It automatically takes corrective actions if it detects any deviations from the desired state (known as "configuration drift"). This ensures consistency across all your deployment targets.
- **Orchestrating Deployments** CM tools integrate with other automation tools within a CI/CD (Continuous Integration/Continuous Deployment) pipeline to manage the complete deployment process. They can handle tasks like:
 - Provisioning new servers or virtual machines
 - Installing pre-requisite software
 - Configuring application settings
 - Deploying application code artifacts
- **Handling Complex Deployments** Modern CM tools support advanced deployment patterns like:
 - **Rolling updates:** Gradually updating servers to minimize downtime.
 - **Blue-green deployments:** Deploying an updated version alongside the old one for seamless switchover.
 - **Canary deployments:** Testing updates on a small subset of users before a full rollout.

Example: Deploying a Web Application

1. **Define Configuration:** Using a CM tool, you define a configuration recipe that includes:
 - Operating system packages (web server, database, runtimes)
 - Web server configuration (virtual hosts, ports)
 - Database settings (usernames, passwords)

- Application code dependencies
- 2. **CI/CD Trigger:** A new code commit in your version control triggers your CI/CD pipeline.
- 3. **Build and Package:** Your CI/CD tool builds the application code and creates an artifact (e.g., ZIP, container image).
- 4. **CM Tool Executes:** The CM tool takes over:
 - Provisions necessary infrastructure if needed.
 - Installs required packages and dependencies.
 - Configures the web server and database as per your recipe.
 - Deploys the new application artifact from your CI/CD tool.

1.3 Characteristics of Automated Application Deployment

Automated application deployment encompasses a series of practices and technologies aimed at streamlining the process of getting software from development into end-users' hands efficiently and reliably. Here are some key characteristics of automated application deployment:

1. Speed

Automation significantly speeds up the software deployment process by reducing the manual steps involved. This means updates and new features can be delivered to users faster, facilitating a faster response to market demands.

2. Consistency

Automated processes ensure that deployments are performed consistently every time, reducing the chances of human error. This is particularly important when managing deployments across multiple environments (development, testing, staging, production) or geographically dispersed data centers.

3. Repeatability

Automation allows the deployment process to be repeated under the same conditions every time. This is crucial for maintaining the integrity of the software across different environments and versions. It also simplifies the process of rolling back to a previous version if something goes wrong.

4. Scalability

Automated deployment processes are designed to scale seamlessly with the infrastructure. Whether deploying to a few servers or thousands, the process remains the same, enabling businesses to scale up or down as needed without additional complexity.

5. Reduced Downtime

Automated deployments can be done in a way that minimizes or even eliminates downtime. Techniques such as blue-green deployments, rolling updates, and canary releases ensure that deployment activities do not disrupt the user experience.

2. Introduction to Denodo

Denodo is a technology platform specializing in data virtualization, providing businesses with the ability to integrate data from various sources, such as databases, web services, and flat files, without replicating the data physically. Data virtualization presents a unified view of the data from these disparate sources to end-users and applications, facilitating more efficient data management, quicker access, and the ability to derive insights more effectively. Here's an introduction to Denodo and the concept of data virtualization:

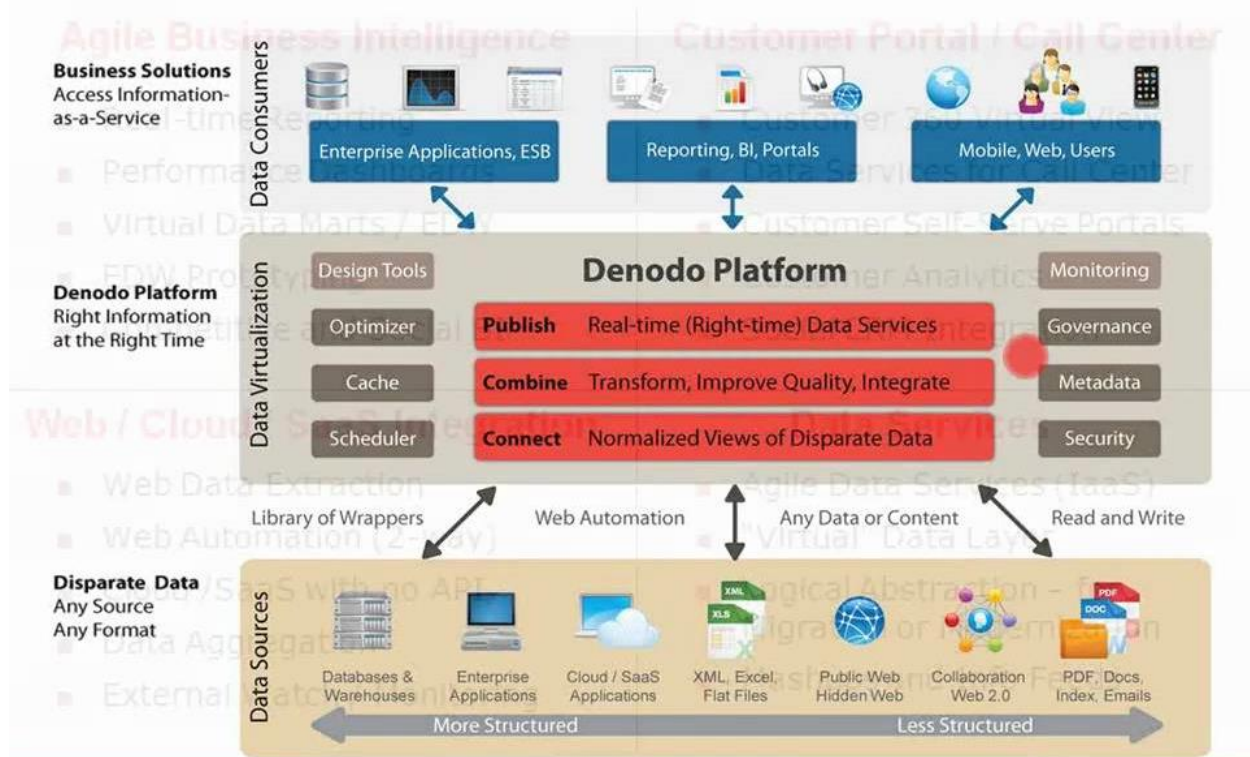
2.1 What is Data Virtualization?

Data virtualization is a method of data management that allows an application to retrieve and manipulate data without requiring technical details about the data, such as how it is formatted or where it is physically located. This approach offers a layer of abstraction over the physical implementation of data, enabling dynamic data integration, real-time data delivery, and enhanced business agility.

2.2 Key Features of Denodo

- a) **Unified Data Access:** Denodo integrates data from diverse data sources, including traditional databases, big data repositories, cloud platforms, and even data streams. This integration happens in real-time, without duplicating the data, thus offering a consolidated view across the organization.
- b) **Real-time Performance:** Through advanced optimization techniques like query rewriting and caching, Denodo provides high-performance access to various data sources. It ensures that users and applications receive data in a timely manner, which is crucial for decision-making processes.
- c) **On-demand Data Delivery:** Users can access data in the format they need for their specific applications or analytics tools. Denodo transforms and delivers real-time data, supporting different use cases from reporting and business intelligence to machine learning and operational applications.
- d) **Advanced Security and Governance:** Denodo includes robust security features that ensure data is accessed securely and in compliance with organizational policies and external regulations. It manages security at the data virtualization layer, allowing fine-grained access control and auditing.
- e) **Scalability and Flexibility:** The platform can scale to handle large volumes of data and queries without compromising performance. It also offers flexibility in deployment options, supporting on-premises, cloud, and hybrid environments.

Denodo Platform Architecture



2.3 Different Services of Denodo

Denodo Platform

The Denodo Platform is the primary environment for data virtualization provided by Denodo. It integrates data from diverse sources—from databases to web services—and presents it in a unified form to end-users and applications without the need to physically move or replicate the data.

Core Components:

1. Virtual DataPort server (VDP server)

The core of the Denodo Platform is the Virtual DataPort (VDP), which serves as the data virtualization engine. VDP allows users to integrate data from diverse sources such as databases, XML files, web services, and more without physically moving the data. It creates a virtual abstraction layer that users can query using SQL to access the underlying data sources.

2. Denodo Scheduler

Denodo Scheduler is used to schedule and execute data extraction and publishing processes. It helps in automating tasks such as data refreshment, loading data into caches, or exporting data

to other formats or systems at specified intervals, thus supporting operational workflows and real-time data delivery.

3. Data Catalog

Denodo's Data Catalog provides tools for searching, annotating, and organizing data assets available through the Denodo Platform. It enhances data governance by enabling users to understand data through rich metadata management, semantic search capabilities, and lineage tracking, which are essential for regulatory compliance and data quality initiatives.

4. ITPilot

ITPilot allows for the automation of web data extraction and the procedural integration of this data into data virtualization scenarios. It can navigate and interact with web applications to retrieve data, making it possible to include web-derived data sources within the Denodo virtual datasets.

5. Diagnostic & Monitoring Tool

This tool is crucial for maintaining the health and performance of the Denodo Platform. It helps administrators by providing detailed monitoring capabilities and diagnostics to troubleshoot and optimize performance across all components of the platform.

6. Denodo Express

Denodo Express is a free version of the Denodo Platform designed for developmental and educational purposes. It offers many of the features of the full platform but with limitations on scale and some functionalities, ideal for smaller projects or those new to data virtualization.

7. Connectors

Denodo includes a wide array of connectors that allow it to link with various data sources, including major SQL databases, NoSQL databases, web services, data lakes, and big data systems. This universal connectivity is key to its flexible integration capabilities.

Denodo Solution Manager

The Denodo Solution Manager is designed to manage and orchestrate the various instances of the Denodo Platform deployed across an organization. It focuses more on the operational aspects of data virtualization projects. It also manages the license of all machines running the Denodo Platform.

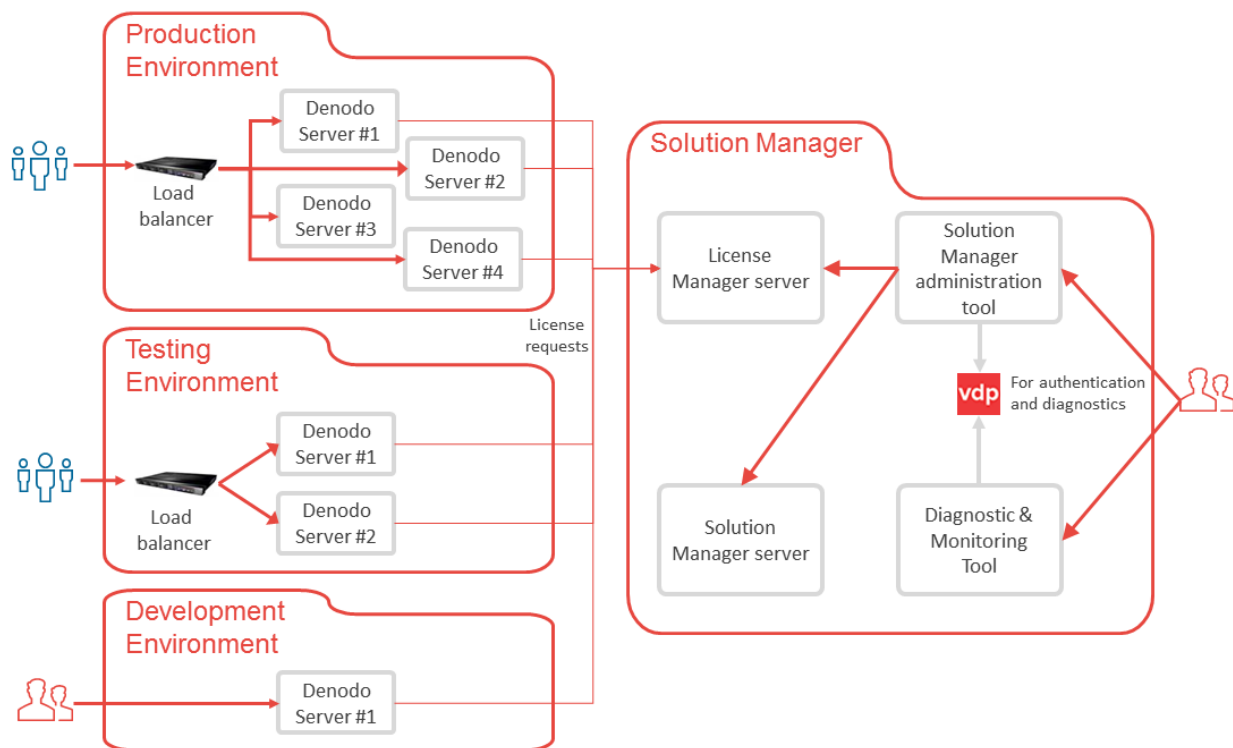
Core Features:

- a) **Lifecycle Management:** Helps manage the full lifecycle of data virtualization projects, including development, testing, deployment, and upgrades.

- b) **Automated Deployment:** Facilitates the automated deployment of Denodo configurations across different environments, making it easier to maintain consistency and efficiency in large-scale deployments.
 - c) **Monitoring and Diagnostics:** Provides tools to monitor the health and performance of Denodo deployments, offering insights into usage patterns and potential bottlenecks.
1. **Security and Governance:** Includes features to manage security policies and compliance requirements effectively across all Denodo instances.

How They Work Together

- **Integration and Management:** While the Denodo Platform provides the data integration capabilities, the Solution Manager oversees the deployment and ongoing management of these integrations across multiple environments and instances.
- **Operational Efficiency:** The Solution Manager enhances the operational efficiency of data virtualization projects by automating routine tasks and managing deployments, which allows data engineers and architects to focus more on strategic aspects of data integration.
- **Scalability and Control:** Together, they ensure that data virtualization scales effectively with organizational needs and that all deployments are under stringent governance and control protocols to meet business and regulatory requirements.



3. Puppet- Configuration Management Tool

Puppet is a robust configuration management tool developed by Puppet Labs, designed to automate the management and configuration of server infrastructures. It leverages a client-server architecture and a custom declarative language to ensure systems are in their desired state. This overview provides detailed insights into how Puppet works and its various components, useful for your thesis on configuration management tools.

3.1 Core Functionality and Architecture

The core functionality and architecture of Puppet, a leading configuration management tool, are designed to automate and streamline the management of various computing environments, from servers to desktops. Below is a detailed discussion of the key elements that define Puppet's architecture and functionality:

1. Client-Server Architecture

Puppet employs a client-server model where the server component is known as the Puppet master and the client components are known as Puppet agents.

- **Puppet Master:** The master node serves as the central control unit. It compiles and manages all the configuration data for the nodes (or agents) that are connected to it. The master processes the Puppet code (manifests), compiles them into catalogs, and serves these catalogs to each client.
- **Puppet Agents:** These are the client machines managed by Puppet. Each agent periodically pulls configuration catalogs from the Puppet master and applies them locally. The frequency of these pulls can be configured according to the needs of the organization.

2. Declarative Configuration Language

At the heart of Puppet's configuration management is its declarative Domain-Specific Language (DSL). This allows users to define the desired state of their hardware and software configurations, which Puppet then works to achieve and maintain:

- **Resources:** These are the basic units for modeling system configurations (e.g., services, packages, files). Users specify resources and the state they should be in without needing to outline the steps to achieve that state.
- **Manifests:** These are Puppet programs composed of Puppet code, which defines actions and configurations using the Puppet DSL.

3. Resource Abstraction Layer (RAL)

Puppet abstracts system configurations through a Resource Abstraction Layer, which allows the software to work across different operating systems and platforms without needing to know the specifics of each platform's configuration commands. This layer simplifies the management of diverse

systems by allowing administrators to focus on the properties of resources rather than their underlying implementations.

4. Transactional and Idempotent Behaviour

Puppet's operations are both transactional and idempotent:

- **Transactional:** Changes to system configurations are made as if they are transactions. Puppet can revert to the previous state if a configuration change fails, ensuring stability.
- **Idempotent:** Puppet's configurations are idempotent, which means that no matter how many times you apply a configuration, the result will be the same as if you had applied it once. This prevents redundancy and unnecessary changes during each configuration run.

5. Modular Architecture

Puppet's functionality is organized into modules, and each module encapsulates a specific task or service. This modularity ensures that Puppet's management capabilities can be extended and customized to fit the needs of any environment:

- **Modules:** Collections of related resources that can be configured as a unit and reused. These modules can be shared within the Puppet community through the Puppet Forge.
- **Classes and Defined Types:** Used to group and encapsulate configurations. Classes are named blocks of Puppet code and can be declared in node definitions and included in modules.

6. Reporting and Compliance

Finally, Puppet provides detailed reporting capabilities that help administrators assess the compliance of their systems against the desired configurations:

- **Reporting:** After each run, Puppet agents send a report to the master detailing what changes were made and any errors that occurred. This aids in debugging and compliance monitoring.

3.2 Key Components or Key Terminologies used in Puppet

The key components of Puppet, a powerful configuration management tool, embody its functionality and are essential for understanding how it manages and automates systems. Here is an in-depth discussion of each major component:

1. Puppet Resources

The most basic building blocks of Puppet, **resources**, represent the elements of your system, like files, packages, services, or users. Each resource type has a set of attributes and a desired state defined by the

user. Puppet ensures that these resources are in the desired state on each managed node. Resource types include core types like file, package, service, and more custom types can be defined as needed.

2. Manifests

Manifests are the main configuration files for Puppet. They are written in Puppet's custom DSL (domain-specific language) and use a .pp file extension. Manifests describe the resources that need to be managed and their desired states using Puppet's declarative language. This means you specify what the system should look like, not how to get it there, which allows the code to be more manageable and understandable.

3. Modules

Modules are self-contained packages of Puppet code that include all the Puppet manifests, files, templates, and other components needed to manage a specific aspect of a system. Modules are designed to be reusable and shareable; they can be written by users or downloaded from Puppet Forge, a repository of pre-built modules from Puppet and the community. This modularity helps manage complex configurations and distribute Puppet code across multiple environments.

4. Classes

In Puppet, **classes** are named blocks of Puppet code that organize and group related resources. Classes can be declared within nodes, and they are a fundamental concept for achieving reusable, clean, and organized Puppet code. Classes can include other classes and thus can be nested to build complex configurations from simple components.

5. Node Definitions

Node definitions are blocks in Puppet manifests that associate classes with specific hosts. This is where you can differentiate configurations for different servers or devices in your infrastructure based on their hostname or a regex matching their hostname. Node definitions allow for specific tailoring of configurations to individual or groups of nodes.

6. Facter

Facter is another key component of Puppet that gathers basic facts about nodes (such as their operating system, IP addresses, and MAC addresses). These facts are then made available in your Puppet manifests as variables and are used to make decisions based on these system characteristics during the configuration management process.

7. Hiera

Hiera is Puppet's built-in key-value configuration data lookup system, which keeps site-specific data out of Puppet manifests. By using Hiera, administrators can separate data from their code and easily manage it without needing to alter the core Puppet codebase. This separation can simplify complex setups and make the Puppet code more portable and easier to maintain.

3.3 How Puppet works

The operation of Puppet can be divided into several steps that describe the lifecycle of a configuration management process:

Step 1: Node Checks In

- Periodically, each node with Puppet Agent installed sends a request to the Puppet Master. This request includes the node's facts gathered by Facter.

Step 2: Compiling Catalogs

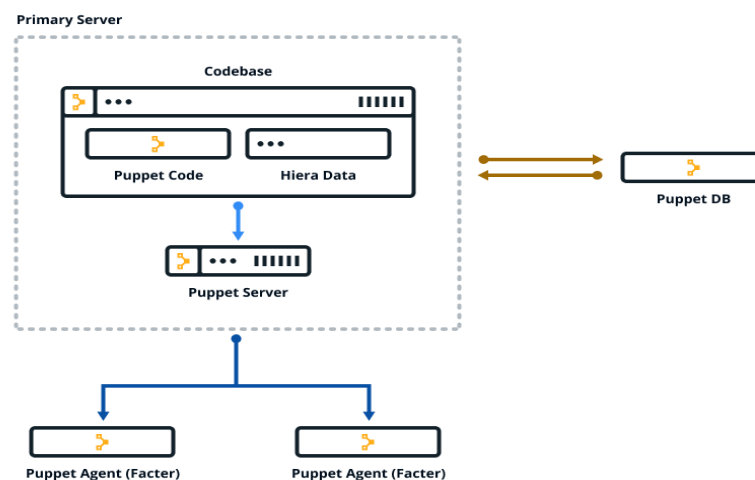
- The Puppet Master uses the facts provided by the node to compile a detailed configuration catalog. This catalog describes exactly how the node should be configured. Compilation is done according to the environments and node-specific details defined in the Puppet manifests.

Step 3: Applying Configurations

- Once the catalog is compiled, it is sent to the node. The Puppet Agent then applies this catalog to the node, making any changes necessary to bring the node into the desired state as described by the catalog.

Step 4: Reporting

- After the Puppet Agent finishes applying the catalog, a report will be sent back to the Puppet Master. This report contains details on what changes were made and any errors that were encountered during the process.



Puppet workflow.

3.4 Puppet module structure and environment

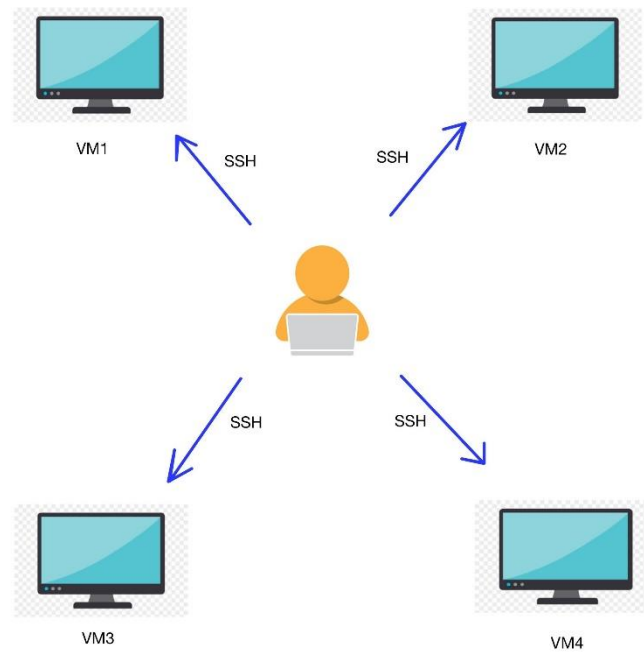
Puppet environments are isolated groups of Puppet agent nodes. A Puppet environment consists of a complete directory structure that includes its own modules, manifests, and data. Environments are used to manage different development stages (such as dev, staging, and production) within an infrastructure.

```
/etc/puppetlabs/code/
├── environments/
│   ├── production/
│   │   ├── manifests/
│   │   ├── modules/
│   │   └── hieradata/
│   ├── testing/
│   │   ├── manifests/
│   │   ├── modules/
│   │   └── hieradata/
│   └── development/
│       ├── manifests/
│       ├── modules/
│       └── hieradata/
└── modules/ # <-- Global modules directory (optional)
```

A Puppet module is essentially a collection of files and directories with a specific, predictable structure. Each module contains all the code and data necessary to manage a specific aspect of a system (like installing a piece of software or managing a service). The structure of a module is designed to keep Puppet code organized and reusable.

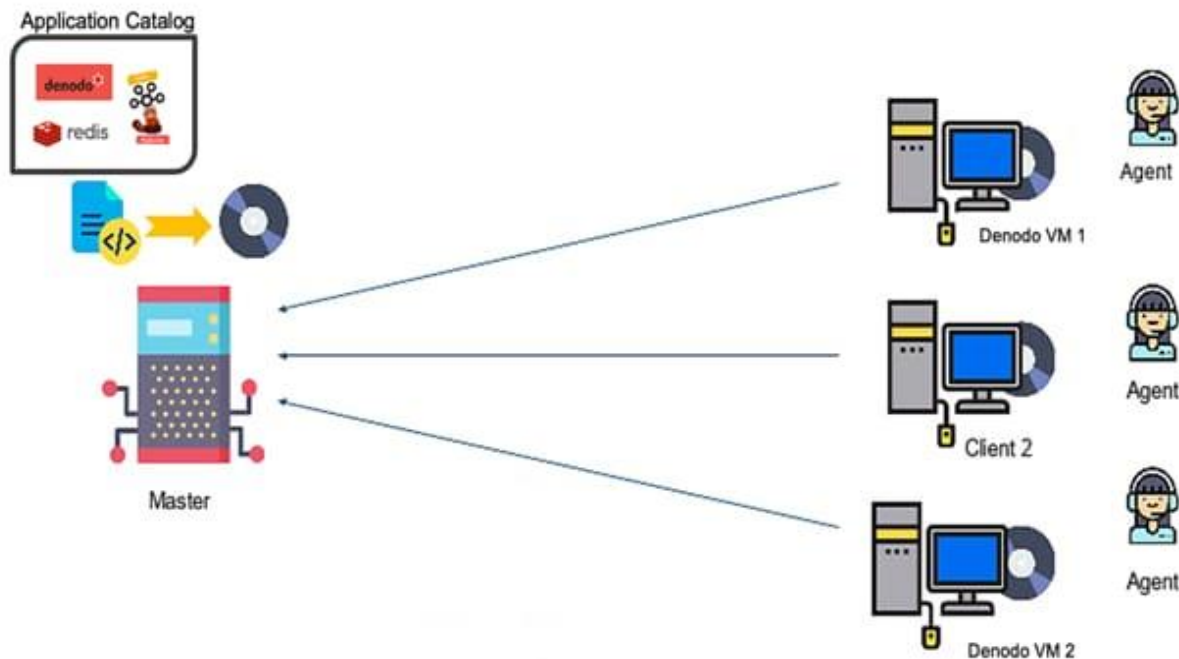
```
module_name/
├── manifests/
│   ├── init.pp # Main class definition
│   └── other_manifests.pp # Additional manifest files
├── templates/ # ERB templates for generating config files
├── files/ # Static files to transfer to managed nodes
├── lib/ # Custom Ruby modules or libraries
├── facts.d/ # Custom facts scripts
├── spec/ # RSpec tests for your module
└── README.md # Module documentation
```

4. Is the manual VS automated deployment process used in TI?



Manual Deployment workflow

1. **Accessing VMs:** The enterprise administrator logs in to each virtual machine (VM) using SSH or a similar remote access protocol.
2. **Installation:** The administrator manually installs the required software or application onto each VM by downloading the installer files or using package managers like apt or yum.
3. **Configuration:** After installation, the administrator configures the application according to the enterprise's specific requirements. This may involve editing configuration files, setting up database connections, or configuring network settings.
4. **Dependencies:** If the application relies on other software or libraries, the administrator ensures that all necessary dependencies are installed and configured correctly.
5. **Testing:** Once the application is installed and configured, the administrator performs testing to verify that it functions as expected and meets the enterprise's requirements.
6. **Monitoring and Maintenance:** After deployment, the administrator monitors the application for performance issues, security vulnerabilities, and updates.



Automated Denodo deployment using puppet workflow

1. Puppet master server has all the application manifests files defined in modules for each application. Application manifests are the steps written in Puppet (DSL Language) for installing and configuring the application.
2. Puppet agent or client servers are the targeted nodes where applications need to be deployed by the master.
3. As puppet works in pull configuration, the agent pulls configuration from the master.
4. In agent servers inside Puppet.conf (located in /etc/puppetlabs/puppet), we define the master server's FQDN from which it has to pull configurations.
5. In the puppet master node, we specify a particular environment for each agent to pull configuration.
6. In this case, Denodo VMs will pull configuration from a Denodo Specific environment set by the Puppet master node.
7. Puppet master will compile the denodo specific manifest files into catalogs and send them back to Denodo VM from which the request came.
8. Puppet agent applies those catalogs into VM and send report to the master about the updated state.

5. Problem Statement

5.1 Background

In the fast-evolving tech landscape, current manual processes for application deployment are proving inefficient. These methods are slow, error-prone, and poorly scalable, directly impacting our operational costs and market responsiveness.

5.2 Challenges

The primary issues with our existing system include:

- **Time-Consuming Operations:** Deployments are manually handled, leading to significant delays.
- **High Error Rate:** Manual setups increase the risk of deployment mistakes, affecting system reliability.
- **Scalability Problems:** Our current method cannot efficiently handle the growing number of applications.
- **Resource Inefficiency:** Excessive human resources are dedicated to routine tasks, detracting from innovation.

5.3 Project Objective

The goal is implementing an automated deployment system (using a configuration management tool) that ensures faster, more reliable, and uniformly efficient application rollouts across different environments. This system will minimize human errors and optimize operational expenditures.

5.4 Specific Goals

- **Enhance Speed:** Achieve at least a 50% reduction in deployment time.
- **Improve Accuracy:** Lower deployment-related errors by up to 90%.
- **Standardization:** Uniform deployment protocols across all operational teams.
- **Robust Monitoring:** Advanced systems for real-time problem-solving and continuous process improvement.

5.6 Expected Outcomes

- **Increased Efficiency:** Deployments will be quicker and require fewer manual interventions.
- **More excellent Reliability:** Standardized processes will minimize operational disruptions.
- **Enhanced Scalability:** The system will quickly adapt to increasing deployment demands without a drop in performance.
- **Resource Allocation:** More resources will be available for innovative and strategic projects.

6. Design of Solution

6.1 Installation and Configuration of Denodo Platform on a cluster

❖ List of Servers: -

- 1 VM for test VDP Server
- 3 VMs for stage VDP Server
- 3 VMs for Production VDP Server
- 1 VM For Denodo Solution Manager

❖ Hardware Requirements: -

- Data Directory file system: **XFS**
- Kernel: **4.18.0-477.15.1.el8_8.x86_64**
- Memory requirement: 12 GB
- CPU core- 4
- Hard disk: at least **50GB** for each server for a cluster. (* Storage space requirements depend on which group will be using that cluster and their requirements)
- Data directory: **/opt/denodo** should be mounted to **/dev/mapper/vgdata-lvoptdenodo** filesystem.

❖ OS Requirements: -

- CentOS / RHEL with version 8 or above

❖ Software requirements: -

- Denodo Platform version 8.0
- Oracle DB Setup for Backend Configuration
- Puppet agent installed

6.2 Installation and Configuration Module designed in puppet

Developed a complete puppet module for automating the installation and Configuration process of Denodo Platform on a server that runs on **CentOS/RHEL**. This module can be applied to any VM where Denodo Platform needs to be installed. Some manual tasks, such as getting access to specific accounts used in Denodo services and SSL certificates for a few Denodo services, are done beforehand before applying the puppet module.

Here is the Complete Puppet module for Automating Denodo Deployment: -

Module name – [*ti_denodo*]

- Manifest File for declaring parameter variables

File location –

[/opt/puppetlabs/code/environments/denodo/modules/ti_denodo/manifests/params.pp]

```

class ti_denodo::params{
  case $facts['os']['family']{
    'RedHat':{
      case $facts['os']['release']['major']{
        '8':{
          #if redhat version is 8 then do nothing
        }
        default:{
          fail("$facts['os']['family']['major'] version is not supported for OS
Family $facts['os']['family']")
        }
      }
    }
    default:{
      fail("$facts['os']['family'] is not supported by module ti_denodo")
    }
  }
}

#variables which will be used during installation are declared here
$denodo_opt_dir="/opt/denodo"
$denodo_backup_dir="${denodo_opt_dir}/backup"
$denodo_logs_dir="${denodo_opt_dir}/logs"
$denodo_certs_dir="${denodo_opt_dir}/certs"
$denodo_software_dir="${denodo_opt_dir}/software"
$denodo_platform_dir="${denodo_opt_dir}/denodo_platform_8.0"
$denodo_platform_installer="denodo-install-8.0-ga-linux64.zip" #main installer file
$denodo_platform_update="denodo-v80-update.jar"
$denodo_platform_hotfix="denodo-v80-hotfix.jar"
$denodo_platform_installer_dir="denodo-install-8.0" #after unzipping this folder is
created with this name by default
$denodo_platform_installer_script="installer_cli.sh" # this is the installer script
to run to install denodo platform
$denodo_platform_install_response="response_file_8_0.xml" # this is the response
file which will be used for silent installation
$denodo_platform_java="${denodo_platform_dir}/jre/bin/java"
$denodo_scripts="${denodo_platform_dir}/bin" # all services related scripts folder
$denodo_user="denodo"
$denodo_group="generic"
$denodo_server_hostname=$facts['networking']['hostname']
#variables related to SSL certs
$denodo_server_keytool_path="${denodo_platform_dir}/jre/bin/keytool"
$denodo_server_keystore_file="denodo_server_key_store.jks"
$denodo_server_keystore_password=hiera(keystore_password,"")
$denodo_server_truststore_password=hiera(truststore_password,"")
$denodo_server_key_alias=hiera(server_key_alias,"")
$denodo_server_csr_file=hiera(server_csr_file,"")
$denodo_server_signed_certificatep7b="${denodo_server_hostname}.p7b"
$denodo_server_signed_certificatecer="${denodo_server_hostname}.cer"
}

```

- Manifest File for defining the user by which module should be deployed

File location –

[/opt/puppetlabs/code/environments/denodo/modules/ti_denodo/manifests/user.pp]

```
class ti_denodo::user{
  $denodo_user=$ti_denodo::params::denodo_user
  $denodo_group=$ti_denodo::params::denodo_group
  exec{ 'change_user_to_denodo':
    command => "su - ${denodo_user}", # user must be denodo before proceeding
    with further command execution
    path => ['/bin','/usr/bin'],
    unless => "whoami | grep ${denodo_user}", # if the user is not denodo only
    then run this resource
  }
}
```

- Manifest File for installing the Denodo Platform

File location –

[/opt/puppetlabs/code/environments/denodo/modules/ti_denodo/manifests/install.pp]

```
class ti_denodo::install{
  #variables defined in ti_denodo params class are referred here
  $denodo_opt_dir=$ti_denodo::params::denodo_opt_dir
  $denodo_backup_dir=$ti_denodo::params::denodo_backup_dir
  $denodo_certs_dir=$ti_denodo::params::denodo_certs_dir
  $denodo_logs_dir=$ti_denodo::params::denodo_logs_dir
  $denodo_software_dir=$ti_denodo::params::denodo_software_dir
  $denodo_platform_dir=$ti_denodo::params::denodo_platform_dir
  $denodo_platform_installer=$ti_denodo::params::denodo_platform_installer

  $denodo_platform_installer_dir=$ti_denodo::params::denodo_platform_installer_dir

  $denodo_platform_installer_script=$ti_denodo::params::denodo_platform_installer_s
  cript

  $denodo_platform_install_response=$ti_denodo::params::denodo_platform_install_res
  ponse

  $denodo_platform_update=$ti_denodo::params::denodo_platform_update
  $denodo_platform_hotfix=$ti_denodo::params::denodo_platform_hotfix
  $denodo_platform_java=$ti_denodo::params::denodo_platform_java
  $denodo_user=$ti_denodo::params::denodo_user
  $denodo_group=$ti_denodo::params::denodo_group

  file{$denodo_opt_dir:
    ensure => directory,
    owner => $denodo_user,
    group => $denodo_group,
    mode => '0644',
  }
}
```

```

#check for /opt/denodo directory size
if ($::denodo_is_not_installed and $::denodo_dir_size < 49) { # available size
of the /opt/denodo is 49GB when checked at beginning, i.e to check lesser than
49GB and then give warning
  notify {'/opt/denodo directory size requirement not met':
    message => "/opt/denodo directory does not exist or is smaller than 50GB,
make sure that available is space greater than or equal to 50GB",
  }
}
else{
  #create /opt/denodo/software directory on the client's system only if
/opt/denodo directory is present
  file{$denodo_software_dir:
    ensure => directory,
    owner => $denodo_user,
    group => $denodo_group,
    mode => '0644',
    require => File[$denodo_opt_dir]
  }
  #create /opt/denodo/logs directory on the client's system only if /opt/denodo
directory is present
  file{$denodo_logs_dir:
    ensure => directory,
    owner => $denodo_user,
    group => $denodo_group,
    mode => '0644',
    require => File[$denodo_opt_dir]
  }
  #create /opt/denodo/certs directory on the client's system only if
/opt/denodo directory is present
  file{$denodo_certs_dir:
    ensure => directory,
    owner => $denodo_user,
    group => $denodo_group,
    mode => '0644',
    require => File[$denodo_opt_dir]
  }
  #create /opt/denodo/backup directory on the client's system only if
/opt/denodo directory is present
  file{$denodo_backup_dir:
    ensure => directory,
    owner => $denodo_user,
    group => $denodo_group,
    mode => '0644',
    require => File[$denodo_opt_dir]
  }
  #deploy the installer file into client's /opt/denodo/software dir
  file{"${denodo_software_dir}/${denodo_platform_installer}":
    ensure => present,
    source =>
"puppet:///modules/ti_denodo/${denodo_platform_installer}",#expecting to have
installer binary file into ti_denodo module file directory
    owner => $denodo_user,
    group => $denodo_group,
    mode => '0644',

```

```

        require => File[$denodo_software_dir],
    }
    #deploy the hotfix jar file into client's /opt/denodo/software dir
    file{"${denodo_software_dir}/${denodo_platform_hotfix}":
        ensure => present,
        source => "puppet:///modules/ti_denodo/${denodo_platform_hotfix}",
        owner => $denodo_user,
        group => $denodo_group,
        mode => '0644',
        require => File[$denodo_software_dir],
    }
    #deploy the update jar file into client's /opt/denodo/software dir
    file{"${denodo_software_dir}/${denodo_platform_update}":
        ensure => present,
        source => "puppet:///modules/ti_denodo/${denodo_platform_update}",
        owner => $denodo_user,
        group => $denodo_group,
        mode => '0644',
        require => File[$denodo_software_dir],
    }
    #put the ldap cert file into certs directory
    file{"${denodo_certs_dir}/ldap.cer":
        ensure => present,
        source => "puppet:///modules/ti_denodo/ldap.cer",
        owner => $denodo_user,
        group => $denodo_group,
        mode => '0644',
        require => File[$denodo_certs_dir],
    }
    #unzip the installer into client's /opt/denodo directory
    exec{'unzip_file_denodo_installer':
        command => "unzip -q ${denodo_software_dir}/${denodo_platform_installer}",
        path => ["/usr/bin", "/bin"],
        cwd => $denodo_software_dir,
        onlyif => "test ! -d
${denodo_software_dir}/${denodo_platform_installer_dir}", #run the unzip command
only if the folder(after unzipped) does not exist
        user => $denodo_user,
        group => $denodo_group,
        require => File["${denodo_software_dir}/${denodo_platform_installer}"],
    } ->
    # response.xml file for unattended installation
    file{
"${denodo_software_dir}/${denodo_platform_installer_dir}/${denodo_platform_install_response}": # place the response file into /opt/denodo/software/denodo-install-8.0
        ensure => present,
        source =>
        "puppet:///modules/ti_denodo/${denodo_platform_install_response}",
        owner => $denodo_user,
        group => $denodo_group,
        mode => '0644',
    }
    #change the access mode of the installer script file to executable
    exec{'change_mode_installer_script':

```

```

        command => "chmod +x
${denodo_software_dir}/${denodo_platform_installer_dir}/${denodo_platform_installer_script}",
        path => ["/usr/bin", "/bin"],
        onlyif => "test ! -x
${denodo_software_dir}/${denodo_platform_installer_dir}/${denodo_platform_installer_script}", # change the file operation mode to executable only if it is not in executable mode
        user => $denodo_user,
        group => $denodo_group,
        require => Exec['unzip_file_denodo_installer'],
    }

    #run the installer to complete installation
    exec{'install_denodo_platform':
        command =>
"${denodo_software_dir}/${denodo_platform_installer_dir}/${denodo_platform_installer_script} install --autoinstaller ${denodo_platform_install_response}",
        creates => $denodo_platform_dir,
        user => $denodo_user,
        group => $denodo_group,
        require => Exec['change_mode_installer_script'],
    }
    #steps for update
    exec{'update_denodo_platform':
        command => "${denodo_platform_java} -jar ${denodo_platform_update}
${denodo_platform_dir} -c > ${denodo_software_dir}/denodo_update_log.log",
        cwd => "${denodo_software_dir}",
        path => ["/usr/bin", "/bin"],
        onlyif => "test -f ${denodo_platform_dir}",
        user => $denodo_user,
        group => $denodo_group,
        subscribe => Exec['install_denodo_platform'],
        refreshonly => true, # only update at the time when denodo platform is installed
    }
    # steps for hotfix
    exec{'hotfix_denodo_platform':
        command => "${denodo_platform_java} -jar ${denodo_platform_hotfix}
${denodo_platform_dir} -c > ${denodo_software_dir}/denodo_hotfix_log.log",
        cwd => "${denodo_software_dir}",
        path => ["/usr/bin", "/bin"],
        onlyif => "test -f ${denodo_platform_dir}",
        user => $denodo_user,
        group => $denodo_group,
        subscribe => Exec['install_denodo_platform'],
        refreshonly => true, # only hotfix at the time when denodo platform is installed
    }
}
}
}

```

- Manifest file for declaring all the configurations related to Denod Platform

File location –

[/opt/puppetlabs/code/environments/denodo/modules/ti_denodo/manifests/config.pp]

```
class ti_denodo::config{

    #variables defined in ti_denodo_params is referenced here
    $denodo_platform_dir=$ti_denodo::params::denodo_platform_dir
    $denodo_user=$ti_denodo::params::denodo_user
    $denodo_group=$ti_denodo::params::denodo_group

    # variables related to SSL setup declared in params referenced here
    $denodo_certs_dir=$ti_denodo::params::denodo_certs_dir
    $denodo_server_keystore_file=$ti_denodo::params::denodo_server_keystore_file

    $denodo_server_keystore_password=$ti_denodo::params::denodo_server_keystore_passw
ord

    $denodo_server_truststore_password=$ti_denodo::params::denodo_server_truststore_p
assword
    $denodo_server_key_alias=$ti_denodo::params::denodo_server_key_alias
    $denodo_server_csr_file=$ti_denodo::params::denodo_server_csr_file
    $denodo_server_keytool_path=$ti_denodo::params::denodo_server_keytool_path

    $denodo_server_signed_certificatep7b=$ti_denodo::params::denodo_server_signed_cer
tificatep7b

    $denodo_server_signed_certificatecer=$ti_denodo::params::denodo_server_signed_cer
tificatecer

    #following variables which are related to configuration is defined here
    $denodo_platform_conf_dir="${denodo_platform_dir}/conf"
    $denodo_sm_properties="${denodo_platform_conf_dir}/SolutionManager.properties"

    $denodo_vdp_properties="${denodo_platform_conf_dir}/vdp/VDBConfiguration properti
es"

    $denodo_sch_properties="${denodo_platform_conf_dir}/scheduler/ConfigurationParame
ters.properties"
    $denodo_sm_hostname= hiera(ti_denodo::denodo_sm_hostname,'') # fetching sm
hostname from hiera DB
    $denodo_server_fqdn=$facts['fqdn']
    $denodo_scripts="${denodo_platform_dir}/bin"

    #variables related to SSL setup are directly defined here
```

```

$denodo_server_signed_certificatep7b_path="${denodo_certs_dir}/${denodo_server_signed_certificatep7b}"

$denodo_server_signed_certificatecer_path="${denodo_certs_dir}/${denodo_server_signed_certificatecer}"

$denodo_server_keystore_file_path="${denodo_certs_dir}/${denodo_server_keystore_file}"
$denodo_server_csr_file_path="${denodo_certs_dir}/${denodo_server_csr_file}"

$denodo_server_truststore_path="${denodo_platform_dir}/jre/lib/security/cacerts"

$denodo_server_encrypt_script_path="${denodo_platform_dir}/bin/encrypt_password.sh"
$denodo_server_keystore_encrypted_pass="${denodo_certs_dir}/keystore.txt"
$denodo_server_truststore_encrypted_pass="${denodo_certs_dir}/truststore.txt"

$denodo_mail_subject='Mail Subject'
# shut down all the services that are running before make any changes"
# exec { 'shutdown_script_run':
#   command => "${denodo_scripts}"
# }

# configuring solution manager properties file
augeas{'configure_denodo_sm_properties_1':
  incl => $denodo_sm_properties,
  lens => 'Properties.lns',
  changes => [
    "set com.denodo.license.host ${denodo_sm_hostname}",
  ],
}

# enable ssl authentication in the configuration file
file_line{'configure_denodo_sm_properties_2':
  path => $denodo_sm_properties,
  line => "com.denodo.license.security.ssl.enabled=true",
  match => "^#com.denodo.license.security.ssl.enabled=.*"
}

# configuring VDB server properties file
file_line {'configure_denodo_vdp_properties_1':
  path => $denodo_vdp_properties,
  line
=>"com.denodo.vdb.vdbinterface.server.VDBManagerImpl.registryURL=${denodo_server_key_alias}", # alias - dptmgsbx
  match => '^com.denodo.vdb.vdbinterface.server.VDBManagerImpl.registryURL=',
}

```

```

# fetching start_all_niv script for starting all services at a time
file{"${denodo_platform_dir}/bin/start_all_niv.sh":
  ensure => present,
  content => template("ti_denodo/start_script.erb"),
  mode   => "0755",
  owner  => $denodo_user,
  group  => $denodo_group,
}
# fetching shutdown_all_niv script for shutting down all services at a time
file{"${denodo_platform_dir}/bin/shutdown_all_niv.sh":
  ensure => present,
  content => template("ti_denodo/shutdown_script.erb"),
  mode   => "0755",
  owner  => $denodo_user,
  group  => $denodo_group,
}
# trigger regenerate.sh to apply all the configuration
exec { 'run_regenerate_script':
  command => "${denodo_platform_dir}/bin/regenerateFiles.sh",
  user    => $denodo_user,
  group   => $denodo_group,
  require =>
[Augeas['configure_denodo_sm_properties_1'],File_line['configure_denodo_sm_properties_2'],File_line['configure_denodo_vdp_properties_1'],File_line['configure_denodo_vdp_properties_1.1'],File_line['configure_denodo_vdp_properties_1.2'],File_line['configure_denodo_vdp_properties_1.3'],File_line['configure_denodo_vdp_properties_1.4'],File_line['configure_denodo_vdp_properties_1.5'],File_line['configure_denodo_vdp_properties_1.6'],File_line['configure_denodo_vdp_properties_2'],File_line['configure_denodo_vdp_properties_3'],File_line['configure_denodo_vdp_properties_2.1']]
}
->
# SSL setup and config
# can be put in separate manifest file

#generate private key
exec { 'generate_denodo_server_keystore_file':
  command => "${denodo_server_keytool_path} -genkeypair -dname
\"CN=${denodo_server_fqdn}, OU=IT Architecture, O=Texas Instruments\\, Inc.,
L=Dallas, S=Texas, C=US\" -alias ${denodo_server_key_alias} -keyalg RSA -keysize
2048 -storepass ${denodo_server_keystore_password} -keypass
${denodo_server_keystore_password} -keystore ${denodo_server_keystore_file_path}
-deststoretype JKS -v",
  user => $denodo_user,
  group => $denodo_group,
  cwd  => "${denodo_certs_dir}",
  path => ['/bin','/usr/bin'],

```

```

        creates => "${denodo_server_keystore_file_path}",
        require => Exec['run_regenerate_script'],
    }
-> #generate csr file with the generated keystore file
exec { 'generate_denodo_server_csr_file':
    command => "${denodo_server_keytool_path} -certreq -alias
${denodo_server_key_alias} -keystore ${denodo_server_keystore_file_path} -file
${denodo_server_csr_file_path} -storepass ${denodo_server_keystore_password} -
sigalg SHA1withRSA -v",
    user => $denodo_user,
    group => $denodo_group,
    cwd => "${denodo_certs_dir}",
    path => ['/bin', '/usr/bin'],
    creates => "${denodo_server_csr_file_path}",
}
->
# ++++++ Manually upload CSR to digicert and get the certificate ++++++ #
# change ownership to denodo user for all file of the current directory(as cert
file is copied as a root user)
exec { 'change_ownership_to_denodo_for_all_files_in_cert_dir':
    command => "chown ${denodo_user}:${denodo_group} *",
    path => ['/bin', '/usr/bin'],
    cwd => $denodo_certs_dir,
    user => 'root', # run the command as root user
    group => 'root',
    onlyif => "ls -l | cut -f3 -d ' ' | grep -v '^$' | grep -v 'denodo'", # run
this resource only if current ownership of files are not belong to denodo user
}
-> # import the p7b certificate into keystore file
exec { 'import_cert_to_keystore':
    command => "${denodo_server_keytool_path} -import -trustcacerts -alias
${denodo_server_key_alias} -file ${denodo_server_signed_certificatep7b_path} -
keystore ${denodo_server_keystore_file_path} -storepass
${denodo_server_keystore_password} -noprompt",
    user => $denodo_user,
    group => $denodo_group,
    path => ['/bin', '/usr/bin'],
    cwd => $denodo_certs_dir,
    onlyif => "test -f ${denodo_server_signed_certificatep7b_path} && test ! -f
${denodo_certs_dir}/tls.properties", # import the cert only if ssl is not
configured before
    require=> Exec['change_ownership_to_denodo_for_all_files_in_cert_dir'],
}
-> # convert p7b certificate into cer format
exec { 'convert_cert_from_p7b_to_cer_format':
    command => "openssl pkcs7 -print_certs -in
${denodo_server_signed_certificatep7b_path} -out
${denodo_server_signed_certificatecer_path}",

```

```

    user => $denodo_user,
    group => $denodo_group,
    path => ['/bin', '/usr/bin'],
    cwd => $denodo_certs_dir,
    onlyif => "test -f ${denodo_server_signed_certificatep7b_path} && ! test -f
${denodo_server_signed_certificatecer_path}",
    require => Exec['import_cert_to_keystore'],
  }
  # check if ssl certificate is present and tls.properties file does not exist or
empty
  if ($facts['ssl_cert_exists'] == 'true' and
$facts['tls_prop_file_not_exist_or_empty']) {
    # shutdown the services before installing ssl certificate
    exec {'shutting_down_denodo_services':
      command => "${denodo_platform_dir}/bin/shutdown_all_niv.sh",
      path => ['/bin', '/usr/bin'],
      onlyif => "lsof -i:9000 || lsof -i:8000", #shutdown when services are
running
    }
    # steps for creating tls.properties file with keystore and truststore
encrypted password
    # step 1 - create a txt file with raw password of keystore
    file{ "${denodo_certs_dir}/keystore_password.txt":
      ensure => file,
      owner => $denodo_user,
      group => $denodo_group,
      content => "password=${denodo_server_keystore_password}",
      require => Exec['import_cert_to_keystore'],
    }
    # use encryption script to encrypt and store the password into another txt
file
    exec { 'generate_encrypted_password_for_keystore':
      command => "${denodo_server_encrypt_script_path} -cf
${denodo_certs_dir}/keystore_password.txt -f
${denodo_certs_dir}/keystore_encpt_pass.txt",
      path => ['/bin', '/usr/bin'],
      cwd => $denodo_certs_dir,
      user => $denodo_user,
      group => $denodo_group,
      subscribe => File["${denodo_certs_dir}/keystore_password.txt"],
      refreshonly => true,
    }
    # step 2 - create a txt file with raw password of truststore
    file{ "${denodo_certs_dir}/truststore_password.txt":
      ensure => file,
      owner => $denodo_user,
      group => $denodo_group,
      content => "password=${denodo_server_truststore_password}",

```

```

        require => Exec['import_cert_to_keystore'],
    }
    # use encryption script to encrypt and store the password into another txt
    file
    exec { 'generate_encrypted_password_for_truststore':
        command => "${denodo_server_encrypt_script_path} -cf
        ${denodo_certs_dir}/truststore_password.txt -f
        ${denodo_certs_dir}/truststore_encpt_pass.txt",
        cwd => $denodo_certs_dir,
        path => ['/bin', '/usr/bin'],
        user => $denodo_user,
        group => $denodo_group,
        subscribe => File["${denodo_certs_dir}/truststore_password.txt"],
        refreshonly => true,
    }
    # create tls.properties with credentials
    file{ "${denodo_certs_dir}/tls.properties":
        ensure => file,
        owner => $denodo_user,
        group => $denodo_group,
        require =>
        [Exec['generate_encrypted_password_for_keystore'], Exec['generate_encrypted_passwo
        rd_for_truststore']],
    }
    -> # putting encrypted passwords into tls.properties file with the help of
    previously created txt files
    exec{"update_tls_properties_file":
        command => "bash -c 'echo keystore.password=$(cat
        /opt/denodo/certs/keystore_encpt_pass.txt) > /opt/denodo/certs/tls.properties &&
        echo truststore.password=$(cat /opt/denodo/certs/truststore_encpt_pass.txt) >>
        /opt/denodo/certs/tls.properties'",
        path => ['/bin', '/usr/bin'],
        user => $denodo_user,
        group => $denodo_group,
        require =>
        [Exec['generate_encrypted_password_for_keystore'], Exec['generate_encrypted_passwo
        rd_for_truststore']],
    }
    # import ldap cert into truststore
    exec { 'import_ldap_cer_file_into_truststore':
        command => "${denodo_server_keytool_path} -import -alias ldapSSL -file
        ldap.cer -keystore ${denodo_server_truststore_path} -storepass
        ${denodo_server_truststore_password} -noprompt",
        cwd => $denodo_certs_dir,
        user => $denodo_user,
        group => $denodo_group,

```

```

        unless => "${denodo_server_keytool_path} -list -keystore
${denodo_server_truststore_path} -storepass ${denodo_server_truststore_password}
-alias ldapssl",
    }
    #run tls_configurator.sh file to complete the setup of SSL certificate
    exec { 'run_tls_configurator':
/opt/denodo/denodo_platform_8.0/jre/lib/security/cacerts --license-manager-uses-
tls=true --components scheduler,vdp,tomcat,vdp-admin,scheduler-index --
credentials-file /opt/denodo/certs/tls.properties --denodo-home
/opt/denodo/denodo_platform_8.0"
        command => "${denodo_platform_dir}/bin/denodo_tls_configurator.sh --
keystore ${denodo_server_keystore_file_path} --cert-cer-file
${denodo_server_signed_certificatecer_path} --truststore
${denodo_server_truststore_path} --license-manager-uses-tls=true --components
scheduler,vdp,tomcat,vdp-admin,scheduler-index --credentials-file
${denodo_certs_dir}/tls.properties --denodo-home ${denodo_platform_dir}",
        user => $denodo_user,
        group => $denodo_group,
        subscribe => Exec['update_tls_properties_file'],
        notify    => Exec['change_max_threads'],
        refreshonly => true,
    }
    # delete all intermediate txt files which were used to create tls.properties
file(after tls configurator runs successfully)
    exec { 'delete_all_intermediate_files':
        command    => "find ${denodo_certs_dir} -type f -name \"*.txt\" -
delete",
        cwd        => $denodo_certs_dir,
        path       => ['/bin', '/usr/bin'],
        subscribe  => Exec['run_tls_configurator'],
        refreshonly => true,
    }
}
elseif(!$facts['tls_prop_file_not_exist_or_empty']) {
    # notify { '::: }
}
else{
    notify { 'Please obtain the SSL certificate into p7b format and place into
/opt/denodo/certs directory': }
}
# Configure Metadata DB
exec { 'copy_oracle_db_drivers':
    command => "cp ${denodo_platform_dir}/lib/extensions/jdbc-drivers/oracle-
19c/* ${denodo_platform_dir}/lib/data-catalog-extensions/",
    path    => ['/bin', '/usr/bin'],
    user    => $denodo_user,
    group   => $denodo_group,

```

```

    onlyif => "test ! -f ${denodo_platform_dir}/lib/data-catalog-
extensions/oracle-19c/ojdbc8.jar || test ! -f ${denodo_platform_dir}/lib/data-
catalog-extensions/oracle-19c/orai18n.jar", # copy only if jar files are not
present in the directory
}
file("${denodo_platform_dir}/bin/dv_meta_config_file.properties":
    ensure => present,
    owner  => $denodo_user,
    group  => $denodo_group,
    content => template('ti_denodo/meta_config.erb'),
    notify => Exec['run_regenerateMetadata'],
}
->
exec {'run_regenerateMetadata':
    command => "./regenerateMetadata.sh --file dv_meta_config_file.properties --
reset -y",
    cwd     => "${denodo_platform_dir}/bin",
    user    => $denodo_user,
    group   => $denodo_group,
    refreshonly => true,
}
# update maxThreads value in server.xml file
exec {'change_max_threads':
    command     => "sed -i 's/\\(maxThreads=\\\"\\\"\\)[0-9]*\\\"/\\1300\\\"/'
${denodo_platform_dir}/resources/apache-tomcat/conf/server.xml",
    path        => ['/bin', '/usr/bin'],
    unless      => "grep -q 'maxThreads=\\\"300\\\"'
${denodo_platform_dir}/resources/apache-tomcat/conf/server.xml",
    refreshonly => true,
}
->
#update tomcat properties file
file_line{ 'update_tomcat_properties_file':
    ensure => present,
    path   => "${denodo_platform_dir}/resources/apache-
tomcat/conf/tomcat.properties",
    match  => "java.env.DENODO_OPTS_START=-Xmx1024m -
Dorg.apache.tomcat.util.buf.UDecoder.ALLOW_ENCODED_SLASH=true -
Dorg.apache.catalina.connector.CoyoteAdapter.ALLOW_BACKSLASH=true -
Djava.locale.providers=COMPAT,SPI",
    line   => "java.env.DENODO_OPTS_START=-Xmx3072m -
Dorg.apache.tomcat.util.buf.UDecoder.ALLOW_ENCODED_SLASH=true -
Dorg.apache.catalina.connector.CoyoteAdapter.ALLOW_BACKSLASH=true -
Djava.locale.providers=COMPAT,SPI -XX:+HeapDumpOnOutOfMemoryError -
XX:HeapDumpPath=/opt/denodo/logs/tomcat_heap_dump.hprof",
    require => Exec['change_max_threads'],
}
# Update DesignStudioBackend.properties

```

```

file_line {'update_design_studio_prop1':
  ensure   => present,
  path     => "${denodo_platform_dir}/conf/design-
studio/DesignStudioBackend.properties",
  after    => "backend.web.retrieveRowsLimit=1000",
  line     => "backend.web.executeAndDownloadRowsLimit=100000",
}
->
file_line {'update_design_studio_prop2':
  ensure   => present,
  path     => "${denodo_platform_dir}/conf/design-
studio/DesignStudioBackend.properties",
  line     => "session.maxTimeToLive=8h",
}
file("${denodo_platform_conf_dir}/data-catalog/datasource.properties":
  ensure => present,
  owner  => $denodo_user,
  group  => $denodo_group,
  content => template('ti_denodo/dc_datasource.erb'),
}

# unzip denodo monitor installer
exec {'unzip_denodo_monitor':
  command   => "tar -xzf ${denodo_platform_dir}/tools/monitor/denodo-
monitor.tar.gz",
  path      => ['/bin', '/usr/bin'],
  user      => $denodo_user,
  group     => $denodo_group,
  cwd       => "${denodo_platform_dir}/tools/monitor",
  creates   => "${denodo_platform_dir}/tools/monitor/denodo-monitor/",
}
# update denodo monitor configuration properties
file("${denodo_platform_dir}/tools/monitor/denodo-
monitor/conf/ConfigurationParameters.properties":
  ensure   => present,
  content  => template('ti_denodo/denodo_monitor_conf_prop.erb'),
  mode     => "0644",
  owner    => $denodo_user,
  group    => $denodo_group,
  require  => Exec['unzip_denodo_monitor'],
}
}

```

- Manifest file for configuring Denodo Monitor Reports. This tool helps to monitor all the services operated by Denodo.

File location -

*[/opt/puppetlabs/code/environments/denodo/modules/ti_denodo/manifests/dmr_config.p
p]*

```
class ti_denodo::dmr_config {
  $denodo_user=$ti_denodo::params::denodo_user
  $denodo_group=$ti_denodo::params::denodo_group
  $denodo_platform_dir=$ti_denodo::params::denodo_platform_dir
  $denodo_software_dir=$ti_denodo::params::denodo_software_dir
  $denodo_server_key_alias=$ti_denodo::params::denodo_server_key_alias

  $denodo_monitor_reports_dir="${denodo_platform_dir}/tools/denodo_monitor_reports"
  $denodo_monitor_reports_zip="Denodo-Monitor-Reports.zip"
  $denodo_server_fqdn=$facts['fqdn']

  if ( $::denodo_platform_localhost_exist ){
    # fetching denodo monitor reports binary file into system
    file{"${denodo_software_dir}/denodo_monitor_reports":
      ensure      => directory,
      source      => "puppet:///modules/ti_denodo/denodo_monitor_reports",
      recurse     => true,
      owner       => $denodo_user,
      group       => $denodo_group,
      mode        => '0644',
    }
    # unzip denodo monitor reports binary
    exec{'unzip_denodo_monitor_reports':
      command     => "unzip
${denodo_software_dir}/denodo_monitor_reports/${denodo_monitor_reports_zip} -d
${denodo_monitor_reports_dir}",
      path        => ['/bin','/usr/bin'],
      user        => $denodo_user,
      group       => $denodo_group,
      onlyif      => "test ! -d ${denodo_monitor_reports_dir}",
      require     => File["${denodo_software_dir}/denodo_monitor_reports"],
    }
    # get dmr_startup script from templates
    file{"${denodo_monitor_reports_dir}/denodo-monitor-reports-jetty-8.0/files-
for-denodo-container/dmr_startup.sh":
      ensure      => present,
      content     => template('ti_denodo/dmr_startup.erb'),
      mode        => "0755",
      owner       => $denodo_user,
```

```

        group      => $denodo_group,
        require    => Exec['unzip_denodo_monitor_reports'],
    }
    # get dmr_shutdown script from template
    file{"${denodo_monitor_reports_dir}/denodo-monitor-reports-jetty-8.0/files-
for-denodo-container/dmr_shutdown.sh":
        ensure      => present,
        content     => template('ti_denodo/dmr_shutdown.erb'),
        mode        => "0755",
        owner       => $denodo_user,
        group       => $denodo_group,
        require     => Exec['unzip_denodo_monitor_reports'],
    }
    # copy DenodoMonitorReports.war file to $DENODO_HOME/resources/apache-
tomcat/webapps
    exec {'copy_war_file_to_apache_tomcat_webapps':
        command     => "cp ${denodo_monitor_reports_dir}/denodo-monitor-reports-
jetty-8.0/webapps/DenodoMonitorReports.war
${denodo_platform_dir}/resources/apache-tomcat/webapps/",
        path        => ['/bin','/usr/bin'],
        user        => $denodo_user,
        group       => $denodo_group,
        onlyif      => "test ! -f ${denodo_platform_dir}/resources/apache-
tomcat/webapps/DenodoMonitorReports.war",
        require     => Exec['unzip_denodo_monitor_reports'],
        notify      => Exec['change_ownership_to_denodo_in_webapps'],
    }
    # change ownership to denodo after copying
    exec { 'change_ownership_to_denodo_in_webapps':
        command     => "chown -R ${denodo_user}:${denodo_group} *",
        cwd         => "${denodo_platform_dir}/resources/apache-tomcat/webapps/",
        path        => ['/bin','/usr/bin'],
        user        => 'root',
        group       => 'root',
        refreshonly => true,
    }
    # copy DenodoMonitorReports.xml file to $DENODO_HOME/resources/apache-
tomcat/conf/DenodoPlatform-8.0/localhost
    exec {'copy_denodo_reports_xml_to_localhost':
        command     => "cp ${denodo_monitor_reports_dir}/denodo-monitor-reports-
jetty-8.0/files-for-denodo-container/DenodoMonitorReports.xml
${denodo_platform_dir}/resources/apache-tomcat/conf/DenodoPlatform-
8.0/localhost",
        path        => ['/bin','/usr/bin'],
        user        => $denodo_user,
        group       => $denodo_group,
        onlyif      => "test ! -f ${denodo_platform_dir}/resources/apache-
tomcat/conf/DenodoPlatform-8.0/localhost/DenodoMonitorReports.xml",

```

```

        require      => Exec['unzip_denodo_monitor_reports'],
        notify       => Exec['change_ownership_to_denodo_in_localhost'],
    }
    # change ownership to denodo after copying
    exec {'change_ownership_to_denodo_in_localhost':
        command      => "chown -R ${denodo_user}:${denodo_group} *",
        cwd           => "${denodo_platform_dir}/resources/apache-
tomcat/conf/DenodoPlatform-8.0/localhost",
        path          => ['/bin', '/usr/bin'],
        user          => 'root',
        group         => 'root',
        refreshonly   => true,
    }
}
}
}

```

- Manifests file for starting denodo services after installation and configuration all the services.

File location –

[/opt/puppetlabs/code/environments/denodo/modules/ti_denodo/manifests/service.pp]

```

class ti_denodo::service{
    #variables which are defined in ti_denodo params class are referred here
    $denodo_platform_dir=$ti_denodo::params::denodo_platform_dir
    $denodo_user=$ti_denodo::params::denodo_user
    $denodo_group=$ti_denodo::params::denodo_group
    $denodo_scripts=$ti_denodo::params::denodo_scripts

    exec{'run_denodo_services_start_script':
        command => "${denodo_scripts}/start_all_niv.sh", # script to start all
the services of denodo platform
        path    => ['/bin', '/usr/bin'],
        user    => $denodo_user,
        group   => $denodo_group,
        unless  => "lsof -i:9000 || lsof -i:8000", # run the start script only if
denodo services are not running
    }
}
}

```

- Manifest file for declaring order of execution for all other manifests files. This file is the entry point of execution.

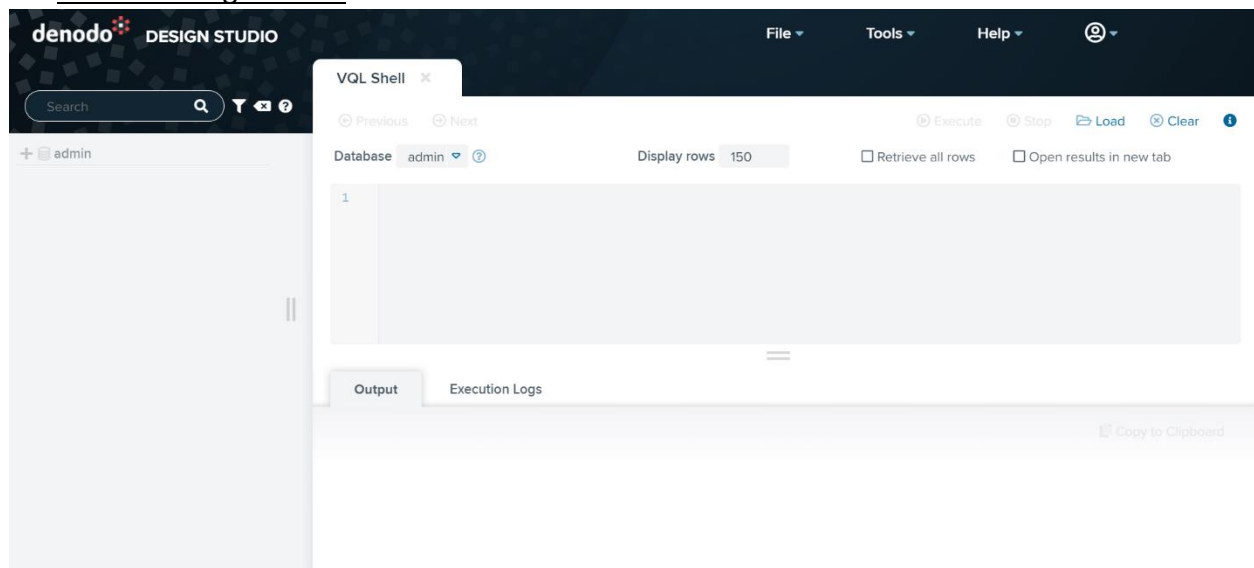
File location –

[/opt/puppetlabs/code/environments/denodo/modules/ti_denodo/manifests/init.pp]

```
class ti_denodo inherits ti_denodo::params{
  class { 'ti_denodo::user': }
  -> class { 'ti_denodo::install': }
  -> class { 'ti_denodo::config': }
  -> class { 'ti_denodo::dmr_config': }
  -> class { 'ti_denodo::service': }
  -> Class['ti_denodo']
}
```

Web Services that come up after automated deployment of Denodo using puppet module in a Denodo VM,

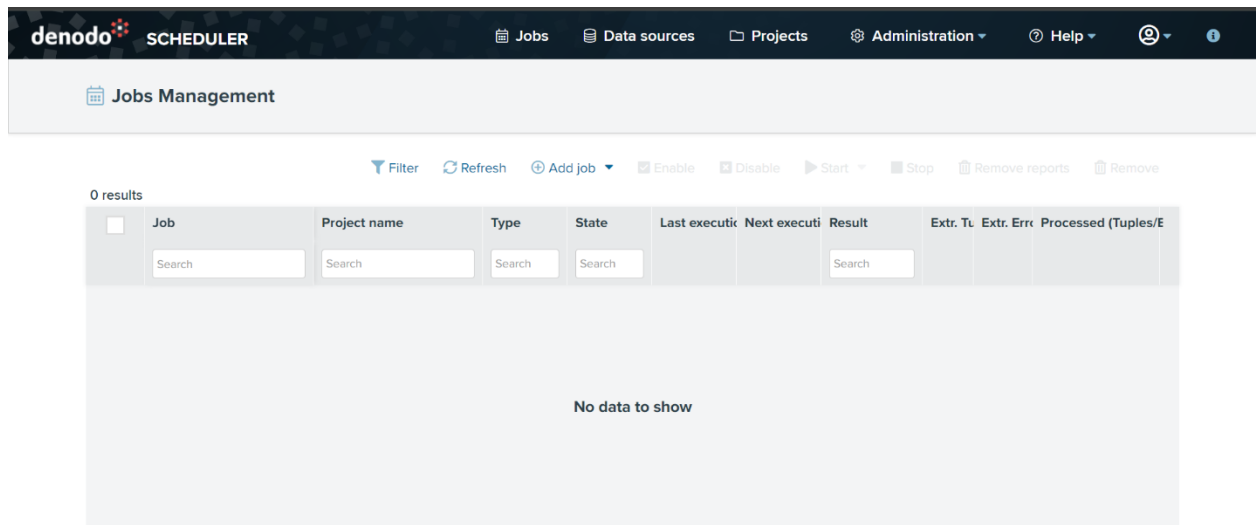
1. Denodo Design Studio



https://<denodo_vm_ip>:9443/denodo-design-studio

Denodo Design Studio is a powerful tool for developing and managing data virtualization solutions. With its intuitive graphical interface, users can easily design and configure data integration workflows, including views, base views, and data services.

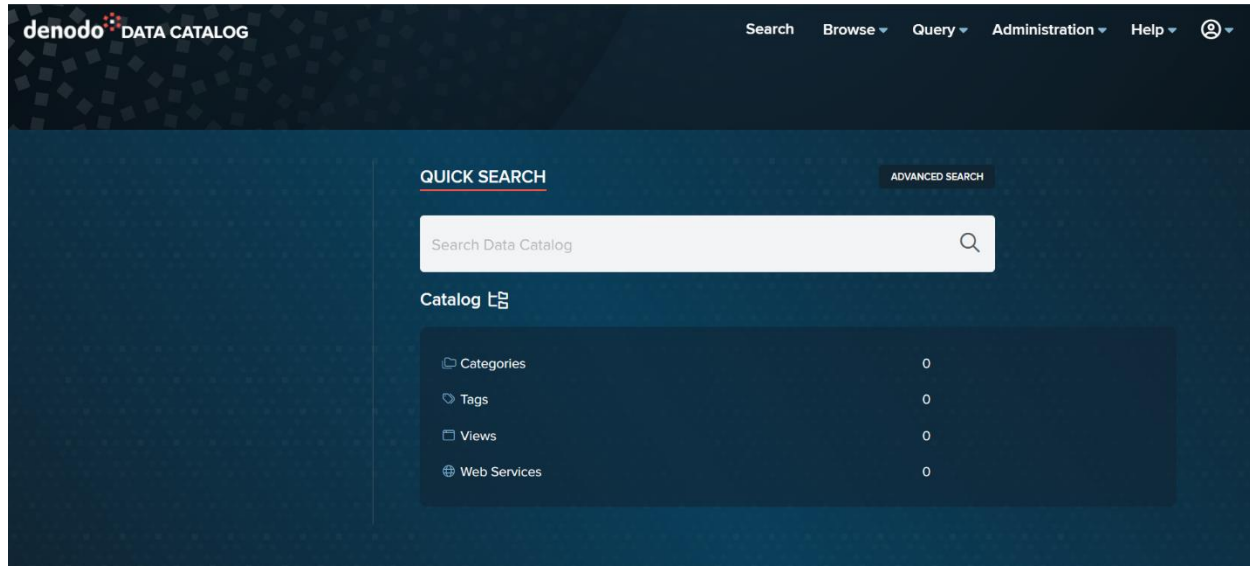
2. Denodo Scheduler Server



https://<denodo_vm_ip>:9443/webadmin/denodo-scheduler-server/

Denodo Scheduler Server is a centralized tool for orchestrating and automating tasks within Denodo data virtualization environments. It enables scheduling of data integration, ETL processes, and data refresh jobs, ensuring timely and efficient data delivery.

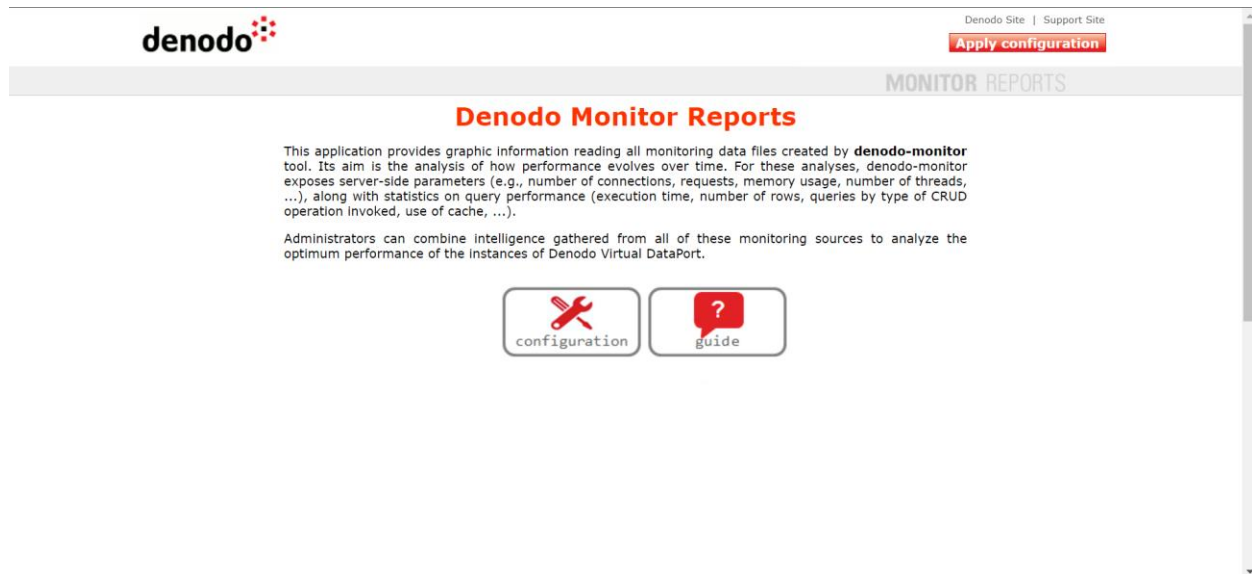
3. Denodo Data Catalog



https://<denodo_vm_ip>:9443/denodo-data-catalog/

Denodo Monitor Reports provide comprehensive insights into the performance and usage of Denodo data virtualization environments. They offer detailed analytics on query execution times, resource utilization, and data access patterns, facilitating optimization efforts.

4. Denodo Monitor Reports



https://<denodo_vm_ip>:9443/DenodoMonitorReports/

Denodo Monitor Reports provide comprehensive insights into the performance and usage of Denodo data virtualization environments. They offer detailed analytics on query execution times, resource utilization, and data access patterns, facilitating optimization efforts. With customizable dashboards and alerts, users can easily identify bottlenecks, optimize queries, and ensure optimal system performance.

7. Results and Analysis

Result:

- The deployment time for Denodo instances has been reduced by an estimated 80%, from an average of 8 hours per cluster to just 1.6 hours. With 10 new instances deployed each month, this translates to a monthly time savings of 64 hours, or approximately eight days.
- Automated deployment allows the organization to provision and configure multiple Denodo instances simultaneously. This has enabled a **300% increase in deployment scalability**, allowing the IT team to deploy new instances to meet growing demand without increasing manpower.
- Puppet's automation has dramatically reduced the incidence of deployment errors. Before automation, an average of 2 out of 10 deployments resulted in errors or misconfigurations. With Puppet, error rates have decreased to less than 1%, resulting in improved system stability and reduced downtime.

Analysis:

- The 150% increase in deployment capacity demonstrates the scalability of the automated deployment framework, positioning the organization for future growth and expansion.
- Before automation, deployment configurations were managed using manual processes, leading to communication gaps and inconsistencies between development and operations teams. With Puppet, deployment configurations are codified as infrastructure code and stored in version control, leading to a 50% improvement in team collaboration and alignment.
- Assuming an average hourly labor cost of \$50, the project results in monthly cost savings of \$5,000.

8. Scope of Future Work

Deploying the cluster is still in the early stages. But there is always scope for future work like: -

Discovery and Utilization of Backend Configuration Files

Future developments might include identifying and utilizing any backend configuration files that Denodo uses to store its settings. By automating the manipulation of these files directly, it would be possible to bypass the UI for many tasks, leading to faster and more reliable configurations.

Broader DevOps Integrations

Containerization of Denodo: Developing capabilities to containerize Denodo instances for better scalability and deployment efficiency using Docker or Kubernetes, with Puppet automating the deployment and management of these containers.

Improved Security and Compliance

Security Automation: Enhance Puppet scripts to include security checks and compliance validations specifically for Denodo setups, ensuring that all data access and integrations meet strict security standards.

Performance and Scalability Enhancements

Auto-Scaling Implementations: Implement auto-scaling strategies for Denodo platforms using Puppet to monitor performance metrics and automatically adjust resources to meet demand without human intervention.

References

Online resources used -

1. <https://community.denodo.com/docs/html/browse/8.0/en/platform/installation/index>
(date of access- 20th Jan,2024)
2. <https://community.denodo.com/docs/html/browse/6.0/vdp/developer/index>
(date of access- 20th Jan,2024)
3. <https://community.denodo.com/docs/html/browse/6.0/scheduler/administration/index>
(date of access- 15th Feb,2024)
4. https://community.denodo.com/docs/html/browse/7.0/vdp/data_catalog/index
(date of access- 5th March,2024)
5. <https://www.puppet.com/docs/puppet/5.5/architecture.html>
(date of access- 22th Jan,2024)
6. <https://www.puppet.com/docs/>
(date of access- 23rd Jan,2024)

Paper names -

1. DevOps Automation: Challenges and Best Practices" by Opsera.io
<https://www.opsera.io/blog/blog-devops-automation>
2. Automation of Production Servers using DevOps (Puppet) Tools
https://repository.stcloudstate.edu/cgi/viewcontent.cgi?article=1075&context=mme_etds

Book names -

1. Puppet 8 for DevOps Engineers
2. Modern DevOps Practices - Second Edition