 *DEPARTMENT OF INFORMATION TECHNOLOGY*

|  |  |
| --- | --- |
| Semester | S.E. Semester IV – Information Technology Engineering |
| Subject | Computer Networks and Network Design Lab |
| Subject Professor In-charge | Unnati Gohil |
| Assisting Teachers | - |
| Laboratory | MS Teams |

|  |  |  |
| --- | --- | --- |
| Student Name | Sanika Kate | |
| Roll Number | 22101A2005 | |
| Grade and Subject Teacher’s Signature |  |  |

|  |  |  |
| --- | --- | --- |
| Experiment Number | 8 | |
| Experiment Title | Case Study on Ansible. | |
| Resources / Apparatus Required | Hardware:  Basic Desktop with Windows or Linux. | Software:  Java/ Python/Wireshark/Cisco Packet Tracer |
| Objectives  (Skill Set / Knowledge Tested / Imparted) |  | | |
| Theory: | **Introduction to Ansible**  In short, Ansible is an open-source automation tool used by system administrators and DevOps professionals to automate tasks like server configuration, application deployment, and system maintenance. It uses simple, human-readable YAML files called playbooks to define automation tasks, connects to target machines via SSH, and ensures idempotent execution, making it easy to manage and maintain infrastructure efficiently. Ansible is agentless, versatile, and has a large community, making it a powerful choice for automating routine tasks in IT operations.  **Challenges Faced before Ansible**  **Manual Configuration:** Administrators had to manually set up and configure servers and software, leading to errors, inefficiencies, and inconsistencies across systems.  **Scalability Issues:** As organizations expanded, managing an increasing number of servers and services became complex and resource-intensive, often resulting in downtime and operational challenges.  **Lack of Documentation:** Inadequate documentation of infrastructure configurations made it difficult to troubleshoot issues, replicate setups, and ensure knowledge sharing within IT teams.  **Security Vulnerabilities**: Manual changes and updates introduced security risks, with human errors or oversights potentially compromising system security.  **Repetitive Tasks:** Routine tasks like patch management and software updates were performed manually, consuming significant time and resources, and increasing the likelihood of human errors.  **Challanges tackled by Ansible**  **Automation of Configuration**: Ansible automates server and software configurations, reducing manual effort, minimizing errors, and ensuring consistency across systems.  **Scalability and Orchestration**: Ansible simplifies the management of large-scale infrastructures by providing tools for efficient scaling and orchestration without downtime.  **Documentation and Reproducibility**: Ansible's infrastructure-as-code approach enhances documentation, making it easier to reproduce configurations and troubleshoot issues accurately.  **Security and Compliance**: Ansible enforces security policies consistently across systems, reducing the risk of vulnerabilities introduced by manual changes and updates.  **Time and Resource Efficiency**: By automating repetitive tasks like patch management, Ansible saves time and resources, increasing IT productivity and reducing the likelihood of human errors.  **Components of Ansible**  **Control Node:** This is where Ansible runs, containing the Ansible command-line tools, playbooks, and inventory files. It serves as the orchestrator of automation tasks.  **Managed Nodes**: These are the servers or devices that Ansible manages. Ansible connects to these nodes to execute tasks, which can be Linux or Windows servers, network devices, or other systems.  **Inventory:** The inventory file lists managed nodes and their attributes. It categorizes hosts and groups and may include variables. It is essential for organizing the infrastructure that Ansible automates.  **Playbooks:** Playbooks are YAML files that define the automation tasks to be performed on managed nodes. They encapsulate roles, variables, and tasks, providing a high-level description of automation.  **Roles:** Roles are reusable sets of automation content, including playbooks, variables, and templates. They help structure and share automation code for specific purposes.  **Modules:** Modules are lightweight units of code that Ansible uses to perform specific tasks on managed nodes. Ansible includes numerous built-in modules for various actions.  **Tasks:** Tasks are individual steps within playbooks, specifying the actions to be taken on managed nodes. Tasks often correspond to module invocations with specific parameters.  **Handlers**: Handlers are tasks that are triggered when specific conditions are met, usually to perform actions like restarting services in response to configuration changes.  **Playbook Execution**: Ansible executes playbooks on the control node, connecting to managed nodes to apply defined tasks. Ansible ensures idempotence, only making necessary changes to reach the desired state.  **Ansible Vault**: Ansible Vault is a tool for encrypting sensitive data, such as passwords and keys, within playbooks and variables, enhancing security by protecting sensitive information.  **Working On Ansible**  **Control Node:**  Ansible is typically installed on a control node, which is the machine from which you manage and execute Ansible tasks. The control node can be a developer's laptop or a dedicated server.  **Inventory:**  The inventory file is a list of target hosts that Ansible will manage. You define the hosts by their IP addresses or hostnames and organize them into groups. Additionally, you can assign variables to hosts or groups in the inventory file.  **SSH Communication:**  Ansible uses SSH (Secure Shell) to connect to the target hosts. It requires SSH access to be set up and configured correctly on the target machines. Ansible connects to the hosts using SSH keys or usernames/passwords, depending on your configuration.  **Modules:**  Ansible provides a wide range of built-in modules that perform specific tasks on remote hosts. Modules are executed on the target machines and return results to the control node. Modules can be used for tasks like installing packages, managing files, starting services, and more.  **Playbooks:**  Playbooks are written in YAML and define a series of tasks to be executed on remote hosts. A playbook can consist of multiple plays, each of which targets a specific group of hosts and executes tasks using modules. Playbooks can also use variables, conditionals, and loops to make automation flexible and dynamic.  **Ad-Hoc Commands:**  Ansible allows you to run ad-hoc commands directly from the command line. These commands are useful for quick, one-off tasks without the need for creating playbooks. Ad-hoc commands use Ansible modules and can target specific hosts or groups.  **Execution Flow:**  When you run an Ansible playbook or ad-hoc command, Ansible connects to the target hosts via SSH, uploads any necessary modules, and executes the tasks defined in the playbook. It collects results and reports back to the control node.  **similar tools like ansible:**  **Chef**:  Chef is an automation platform that uses a domain-specific language (DSL) called "Chef Infra" to define system configurations. It offers a more code-centric approach, where you write recipes and cookbooks to specify how systems should be configured.  **Puppet**:  Puppet is another configuration management tool that uses a declarative language to define desired system states. Puppet uses a client-server architecture, with agents installed on managed nodes that periodically check in with a Puppet master to enforce configurations.  **SaltStack**:  SaltStack, often referred to as Salt, is an infrastructure automation and management tool that uses a unique "event-driven" approach. It can be used for remote execution, configuration management, and orchestration of tasks across a large number of systems.  **CFEngine**:  CFEngine is one of the earliest configuration management tools and focuses on automating system configurations, ensuring compliance, and maintaining desired states across a diverse set of infrastructure.  **Terraform**:  Terraform is an infrastructure-as-code (IaC) tool that specializes in provisioning and managing cloud resources and infrastructure. It uses HashiCorp Configuration Language (HCL) to define infrastructure configurations in a declarative manner.  **Docker and Kubernetes:**  Docker is a containerization platform, and Kubernetes is an orchestration tool for managing containerized applications. While they are not strictly configuration management tools, they are used extensively for application deployment and scaling.  **Comparison of Ansible with Other Tools**  **Ansible vs. Puppet:**  - Ansible is agentless, while Puppet uses agents.  - Ansible&#39;s learning curve is generally considered to be lower.  - Puppet is known for its rich reporting and compliance features.  **Ansible vs. Chef:**  - Similar to Puppet, Chef uses agents, while Ansible is agentless.  - Chef&#39;s recipes and cookbooks provide fine-grained control over  configurations.  - Ansible&#39;s simplicity and YAML-based playbooks make it easier to get started.  **Ansible vs. SaltStack:**  - SaltStack is event-driven and excels in real-time automation.  - Ansible&#39;s playbooks are typically more straightforward and easier to read.  - SaltStack&#39;s use cases often involve complex event-based automation  scenarios.  **Ansible vs. Terraform:**  - Terraform focuses on infrastructure provisioning and is not a configuration  management tool.  - Ansible can handle both configuration management and application  deployment.  - The two tools can be complementary, with Ansible configuring the  infrastructure provisioned by Terraform. | | |
| Output |  | | |
| Conclusion | We have studied about ansible software | | |