DOI: https://doi.org/10.26565/2304-6201-2024-64-10

УДК (UDC) 004.415.2

Vladyslav Samoilenko *Student Kharkiv National University named V. N. Karazin 61000*
e-mail: vladyslav.samoilenko@student.karazin.ua;
ORCID: 0009-0001-3217-4473

Dmitry Bulavin *PhD, associate professor Kharkiv National University named V. N. Karazin 61000. e-mail:* d.bulavin@karazin.ua
ORCID: 0000-0002-4840-4763

Self-Configuring Virtual Machine Environments Using Ansible Pull: A Review of Approaches and Challenges

The automation of virtual machine (VM) configuration has become an integral part of modern IT infrastructure, ensuring compliance with the high demands of scalability, reliability, and security. With the increasing adoption of cloud technologies, there is a growing need for standardized approaches to infrastructure management that eliminate the limitations of traditional methods, such as manual configuration or individual scripts. The goal of this research is to design an architecture for automated VM self-configuration based on Ansible Pull in combination with tools like Terraform, Hashicorp Vault, and Git to ensure flexibility, autonomy, and security.
To build the system, an analytical approach was used to evaluate existing configuration tools, model the architecture using Git repositories, Terraform, and Vault, and experimentally implement cloud-init to initialize Ansible Pull at the VM level. The proposed architecture automates the VM configuration process, ensuring scalability and reducing dependence on centralized management servers. The implementation of Ansible Pull with a tagging system provides idempotent task execution, regular updates, and maintenance of the desired system state.
It has been proven that combining Ansible Pull with Terraform, Git, and Vault ensures simplicity of implementation, flexibility in scaling, and secure configuration management. The proposed approach is effective for dynamic environments requiring frequent updates and aligns with modern DevOps practices.
Keywords: *Ansible Pull, automation of the configuration, Terraform, Vault, self-configuring, DevOps, Ansible*

Як цитувати: Samoilenko V. V., Bulavin D. O. «Self-Configuring Virtual Machine Environments Using Ansible Pull: A Review of Approaches and Challenges». *Вісник Харківського національного університету імені В. Н. Каразіна, серія Математичне моделювання. Інформаційні технології. Автоматизовані системи управління*. 2024. вип. 64. С.104-111. https://doi.org/10.26565/2304-6201-2024-64-10

How to quote: V. Samoilenko, D. Bulavin, “Self-Configuring Virtual Machine Environments Using Ansible Pull: A Review of Approaches and Challenges” *Bulletin of V.N. Karazin Kharkiv National University, series Mathematical modelling. Information technology. Automated control systems*, vol. 64, pp. 104-111, 2024. https://doi.org/10.26565/2304-6201-2024-64-10

1 Introduction

In the modern world of information technology, process automation is a key factor in enhancing efficiency and reducing operational costs. One of the primary tasks in this context is the automation of virtual machine (VM) configuration, which constitutes a vital part of the infrastructure for most organizations. Traditional methods of manual configuration are not only labor-intensive but are also prone to human error, potentially leading to severe system failures.

Automation tools such as Ansible, Puppet, and Chef provide a means to simplify this process by employing a declarative approach. Among these, Ansible stands out for its simplicity, idempotency (ensuring consistent execution results), support for YAML files, and agentless architecture. These features make Ansible an effective tool for managing large-scale server infrastructures in cloud environments, such as AWS, Google Cloud, and Azure.

This study focuses on exploring the capabilities of Ansible Pull as a method for self-organizing virtual machine environments. The emphasis is placed on configuration management, performance optimization, and cost reduction when employing this approach. In addition to analyzing the advantages of Ansible Pull, the paper examines key aspects of deployment process management, including comparisons with similar solutions. The primary challenges associated with integrating Ansible into scalable environments are identified, and strategies for overcoming these challenges are proposed.

Thus, this work aims to establish a foundation for further research and the implementation of automated solutions for virtual machine configuration in modern IT environments.

2 Analysis of the Problem of Self-Configuring Virtual Environments

2.1 Analysis of Modern Methods and Their Limitations

Traditionally, self-configuration is an automated process through which a system configures itself to operate by following predefined instructions without requiring manual intervention. This process includes:

- Installation and management of software dependencies.
- Creation and configuration of user accounts with appropriate access rights.
- Configuration of network settings and ensuring their security.
- Monitoring and ensuring compliance with predefined standards.

The goal of self-configuration is to minimize the risk of human error, accelerate deployment cycles, and ensure the scalability and repeatability of the infrastructure configuration process. This is particularly important for environments that require frequent updates or have complex distributed configurations.

Today, there are several tools and approaches to configuration automation, each with its own advantages and disadvantages:

1. **Custom Scripts:**

- These are easy to create but poorly scalable and challenging to maintain in dynamic environments. Updates often require manual intervention, increasing the risk of errors.

2. **Configuration Management Tools (e.g., Puppet, Chef):**

- These tools are powerful but often require the installation of specialized agents on each node, complicating their use and increasing resource consumption.

3. **Ansible Push:**

- Ansible's push model eliminates the need for agents, simplifying configuration management. However, this approach can create bottlenecks when managing a large number of nodes due to the centralized execution of commands.

4. **Ansible Pull:**

- Ansible Pull offers a decentralized approach where managed nodes retrieve configurations from a Git repository. This approach ensures scalability and reduces dependency on a central control node but comes with its own challenges:
- **Dependency on Git:** Any issues with the repository can halt configuration updates.
- **Network Stability:** Requires a stable SSH connection for initial setup and updates.
- **Version Management:** Ensuring consistent configurations across all nodes can be challenging without proper control.

Thus, Ansible is not the only tool in the configuration automation space. Several other leading tools predate Ansible's release in 2012: SaltStack in 2011, Puppet in 2005, and Chef in 2008. A comparison of the popularity of search queries among these tools is shown in Figure 2.1.

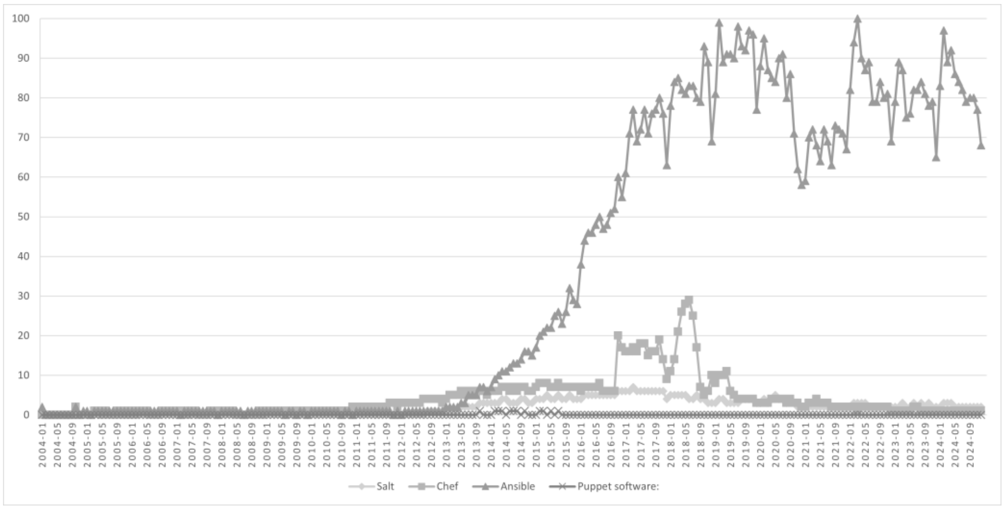


Fig.2.1 Comparing of the search requests Salt, Chef, Ansible ma Puppet popularity in Google.

The choice of tools and methods for self-configuration depends on specific requirements such as scalability, ease of use, and integration capabilities. With Ansible Pull, self-configuration becomes a robust, scalable, and repeatable process that ensures the consistent provisioning of systems according to desired specifications described in code. Storing configurations as code provides significant flexibility and simplifies maintenance.

2.2 Ansible Pull Analysis: Advantages and Disadvantages

Ansible is a popular tool in the category of Infrastructure-as-Code (IaC) that enables automation of deployment and configuration of IT infrastructure. Its agentless architecture, intuitive YAML syntax, and scalability have made Ansible a top choice for organizations seeking to improve process efficiency. Ansible developers create playbooks containing a series of tasks that can be automatically executed on a set of hosts to achieve the desired infrastructure state.

As an automation technology, Ansible is based on the following principles:

- **Agentless Architecture.** Ansible does not require installing software on managed nodes (except in the pull model).
- **Simplicity.** YAML-based playbooks allow easy creation and comprehension of automation scripts. Ansible is also decentralized, using SSH with existing OS credentials to access remote machines.
- **Scalability and Flexibility.** Its modular design allows for rapid scaling, supporting a wide range of platforms and environments.
- **Error-Free, Predictable, and Idempotent.** Ansible files describe infrastructure as code, ensuring the same outcome regardless of how many times the playbooks are executed. [1]

Ansible Pull offers a unique model for self-configuring environments, where managed nodes independently fetch and apply configurations from a repository. This approach enables:

- Avoiding dependence on a central control node.
- Reducing risks associated with single points of failure.
- Efficient operation in distributed infrastructure environments.

Ansible Pull is ideal for dynamic environments that require regular updates, meeting the demands of modern agile systems. [2]

Ansible Pull utilizes a decentralized configuration management method where each node independently fetches and applies updates from a central Git repository. This approach eliminates the need for a centralized control server, enhancing system fault tolerance. Nodes periodically execute the ansible-pull command to download and apply the latest playbooks. Additionally, playbooks can include tasks to update Ansible Pull itself, increasing the autonomy of the process.

Red Hat, the developer of Ansible, provides tools to implement this model but not out-of-the-box solutions. Automating the execution of ansible-pull requires the use of task schedulers such as cron jobs, systemd timers, or similar tools [3]. This modular approach is characteristic of Ansible, offering flexible tools rather than rigid solutions.

Advantages:

1. Autonomy: Each node fetches and applies its configuration independently, minimizing reliance on a central control server and ensuring resilience against network disruptions or server downtime.
2. Scalability: Adding new nodes is simplified since the self-configuration process is integrated into deployment.
3. Flexibility: Changes in the repository are automatically applied during the next execution cycle of ansible-pull, ensuring prompt and consistent updates across all nodes.
4. Version Control: Git integration provides robust version management, allowing effective collaboration, tracking changes, and reverting to previous configurations when needed.
5. No Single Point of Failure: Eliminating reliance on a central server reduces risks associated with potential unavailability.

Disadvantages:

1. Network Dependency: Nodes require connectivity to the Git repository for updates. This can be mitigated by setting up local repository mirrors or using caching mechanisms.
2. Lack of Centralized Management: Unlike the push model, task execution control is more complex, requiring additional monitoring tools such as Ansible callback modules or third-party systems like Prometheus or ELK Stack.

3. Error Debugging: Error logs are stored locally on each node, complicating centralized troubleshooting, especially in large-scale environments. Integration with centralized log collection systems can address this issue.
4. Initial Node Configuration: For ansible-pull to work, each node must be pre-configured with access to the Git repository and necessary dependencies. While this adds steps to configuration, it is a common practice in decentralized systems.
5. Security Challenges: Granting nodes access to the Git repository can pose risks related to data leakage or unauthorized access. Recommendations include restricting access rights, using SSH keys, and secure communication channels (e.g., VPN).

Ansible Pull is a powerful solution for decentralized configuration management, offering autonomy and flexibility. However, its implementation requires additional setup for monitoring, security, and debugging. When properly configured, the system demonstrates high scalability and reliability.

2.3 Comparative Analysis of Existing Solutions

We compare popular configuration management systems such as Chef, Puppet, SaltStack, and Ansible Pull based on key criteria: autonomy, flexibility, security, ease of updates, and scalability to highlight their strengths and weaknesses.

Chef: Uses a pull model where nodes fetch configurations (cookbooks) from a central server. While effective for managing complex configurations, its reliance on the Ruby programming language and the need for a dedicated central server complicates setup and usage.

Puppet: Operates similarly, with nodes fetching manifests from a Puppet master. It supports scalability and consistency but requires installing the Puppet agent and master, increasing complexity. Its domain-specific language (Puppet DSL) reduces accessibility for teams unfamiliar with it.

SaltStack: Primarily uses a push model with a master-minion architecture. While SaltStack offers significant scalability, its setup and usage are complex, though it is compatible with YAML.

Ansible Pull: Unlike the others, Ansible Pull eliminates the need for centralized servers or agents. Nodes independently fetch configurations from a Git repository, making it lightweight and straightforward. Its YAML-based approach and Git integration align with modern DevOps practices, ensuring an intuitive process.

Key Comparative Insights:

- Autonomy: Ansible Pull ensures node autonomy, as each node independently manages updates. Chef and Puppet also support autonomous operation but rely on central servers, creating a potential single point of failure.
- Flexibility: Ansible Pull offers high flexibility with its Git-based workflow, facilitating seamless integration with existing tools and processes. Salt is versatile but more complex to configure, whereas Chef and Puppet require specialized languages or infrastructure, limiting flexibility in some environments.
- Security: Chef and Puppet provide robust authentication and access control through master-agent models. Ansible Pull leverages Git for security, requiring careful management of credentials and repository access control. Salt also provides high security but is harder to configure in pull mode.
- Ease of Updates: Periodic Ansible Pull synchronization with Git ensures quick and consistent updates. Puppet and Chef support automatic updates but with potential delays. Salt’s push model provides real-time updates but loses this advantage in pull mode.
- Scalability: Puppet and Chef scale well due to their master-agent architecture. Ansible Pull is suitable for distributed systems as nodes operate independently. However, large-scale deployments can strain Git traffic during updates, which can be mitigated through update scheduling. Salt is efficient in push scenarios but less performant in pull mode.

Ansible Pull is an excellent choice for environments favoring lightweight, decentralized, and straightforward self-configuration processes. It is especially effective where teams use Git, which was adopted by approximately 93% of developers in 2023 [4]. However, environments requiring highly centralized control or low-latency updates may benefit from solutions like Puppet, Chef, or Salt. Ultimately, Ansible Pull is ideal for organizations seeking scalability, flexibility, efficiency, and seamless integration with modern DevOps workflows.

3 Description of the Self-Configuring Virtual Machine Environments solution

3.1 Architecture

After analyzing all possible approaches to architecture design, we propose a solution distinguished by enhanced flexibility and efficiency.

The proposed architecture utilizes several Git repositories to organize and manage different aspects of the system:

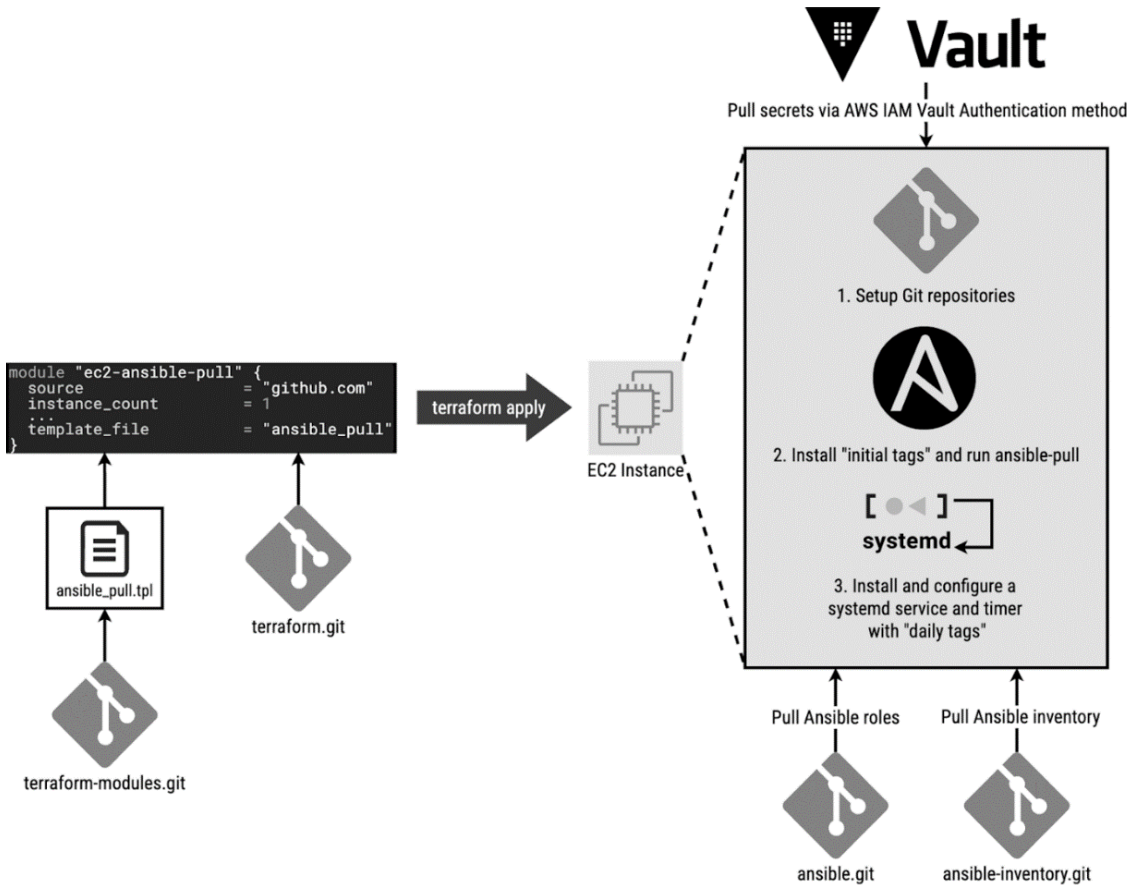
- Ansible Repository: Contains roles and corresponding configurations for automation.
- Ansible-Inventory Repository: Includes inventory files necessary for accurate system configuration.
- Terraform Repository: Stores infrastructure-as-code (IaC) definitions, enabling automated infrastructure creation and management.
- Terraform-Modules Repository: Used for reusable modules to enhance scalability and code reusability.

To manage secrets, Hashicorp Vault is integrated, providing a high level of security. EC2 instances are configured to authenticate with Vault using AWS IAM roles, ensuring secure retrieval of secrets and facts required for configuration processes.

This architecture ensures:

- Separation of responsibilities through specialized repositories.
- Secure handling of sensitive data.
- Scalability and flexibility through the use of Terraform modules.

An example of the proposed architecture is shown in Fig. 3.1.



```
module "ec2-ansible-pull" {
  source              = "github.com"
  instance_count      = 1
  ...
  template_file       = "ansible_pull"
}
```

Vault
Pull secrets via AWS IAM Vault Authentication method
1. Setup Git repositories
2. Install "initial tags" and run ansible-pull
systemd
3. Install and configure a systemd service and timer with "daily tags"
terraform apply
EC2 Instance
ansible_pull.tpl
terraform.git
terraform-modules.git
Pull Ansible roles
Pull Ansible inventory
ansible.git
ansible-inventory.git

Fig.3.1. The overall architecture of the Ansible Pull solution using Terraform, Ansible, Git, and Vault

When an EC2 virtual machine instance is created using Terraform, a cloud-init script embedded in the configuration is executed during its launch. Cloud-init facilitates the connection between VM deployment and configuration without requiring additional user intervention [5]. This script plays a pivotal role in initializing the Ansible Pull model, retrieving secrets, and preparing the VM environment for automated configuration. The architecture seamlessly integrates with AWS services, delivering a scalable and secure self-configuring solution.

Environment Configuration:

- **Git Repository Setup:** The ansible repository contains playbooks and roles, while the ansible-inventory repository holds inventory configurations. The terraform and terraform-modules repositories organize infrastructure code.
- **Vault Setup:** Vault is configured to securely store secrets, ensuring EC2 instances can authenticate using AWS IAM roles [6].

Provisioning and Initialization:

- **Terraform Deployment:** Terraform creates EC2 instances and configures them using a cloud-init script.
- **Dependency Installation:** Dependencies such as Git, Python, Ansible, Ansible Collections, and Pip modules are installed.
- **Repository Cloning:** The ansible and ansible-inventory repositories are cloned onto the VM.
- **Environment Preparation:** The ansible-pull role prepares the environment by fetching the latest updates and applying the necessary configurations.

Self-Configuration:

- The ansible-pull command is executed with specific tags to configure the VM based on its purpose (e.g., database server).
- Following initial configuration, the Ansible Pull role sets up a systemd service to execute the ansible-pull command daily with “daily” tags, ensuring configurations remain up to date.

3.2 Use Cases

Database Server Configuration: using database-specific tags, Ansible Pull installs and configures database software, such as MySQL. Credentials and access permissions are securely retrieved from Vault, and monitoring and logging systems are initialized. Daily pulls ensure configurations remain intact, and security settings are consistently reapplied.

Application Server Configuration: application servers are configured to install runtime dependencies, deploy application code, and set environment variables using secrets from Vault. Daily execution of Ansible Pull ensures the application environment remains consistent and dependencies stay updated.

General Maintenance: a daily system service verifies critical configurations such as SSH settings and dependency versions, applies updates from the Ansible repository, and ensures the system maintains its desired state without manual intervention.

This architecture allows organizations to establish a fault-tolerant and automated environment for managing virtual machines, ensuring consistency, scalability, and security across their infrastructure.

3.3 Key metrics

Ansible Pull can be evaluated using several key metrics, including setup time, failure rate, and maintenance costs. The setup time for Ansible Pull is minimal compared to traditional push-based configuration management systems, as it enables nodes to autonomously fetch configurations without requiring centralized orchestration. The failure rate is typically low, provided proper error-handling mechanisms are included in the playbooks.

Ansible Pull avoids the network bottlenecks and scalability issues inherent in push models. Maintenance costs are reduced because nodes are self-sufficient, decreasing the need for continuous monitoring or administrator intervention. In contrast, Ansible Push requires a control node and is less efficient in large-scale environments, where connectivity issues can disrupt processes. Such a control node may lack centralization, complicating dependency unification and management.

Comparatively, tools like Puppet or Chef offer similar performance but often require more complex setup and incur ongoing licensing or support costs. Research comparing Ansible, Chef, SaltStack, and

Puppet in configuration management highlights metrics such as deployment speed, reliability, and ease of use.

Roman Kostromin conducted a detailed analysis of these tools in heterogeneous distributed computing environments, emphasizing that Ansible outperforms others in simplicity and agentless architecture, reducing setup complexity and human errors. While Chef and Puppet excel in state management and scalability, they demand higher configuration and maintenance effort [7]. Moreover, a 2022 study noted that Ansible's reliance on SSH infrastructure and its agentless nature make it a secure and efficient choice for configuration management, particularly for small to medium-sized deployments, while Chef and Puppet are more suitable for complex infrastructures with frequent configuration changes [8].

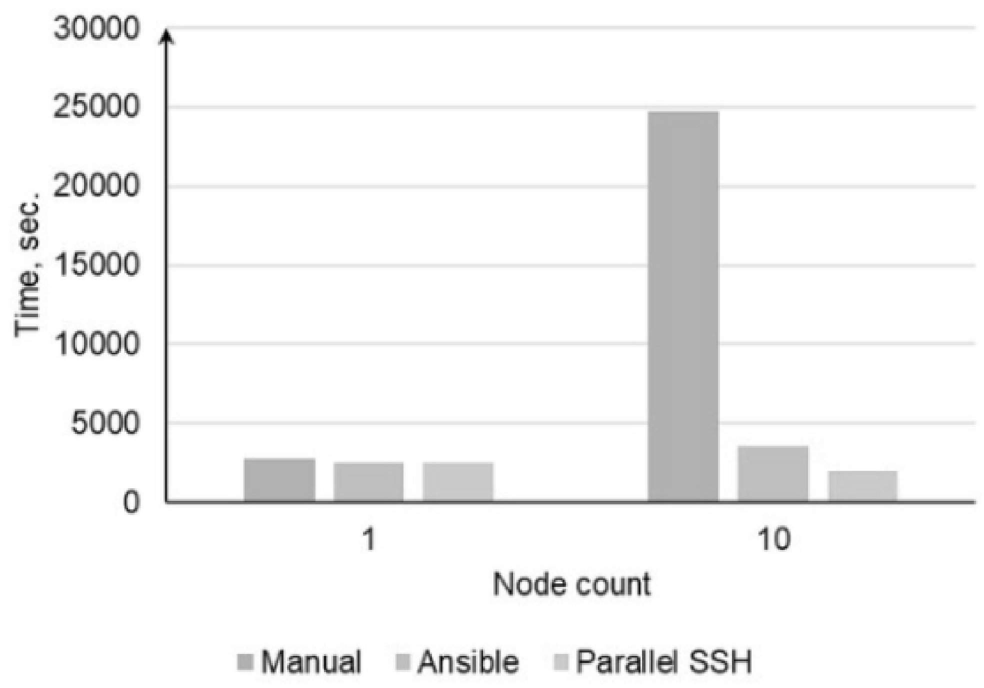


Fig.3.2. Comparison of Node Configuration Time: Manual Setup, Ansible, and Parallel SSH Connections [8]

Ansible Pull is a streamlined, autonomous solution for self-configuring virtual environments, particularly in scenarios where simplicity, scalability, and cost-efficiency are key priorities. Its decentralized model reduces dependence on a central control node, providing enhanced resilience and deployment flexibility.

REFERENCES

1. Introduction to ansible. Introduction to Ansible - Ansible Community Documentation. (2024, November 20). https://docs.ansible.com/ansible/latest/getting_started/introduction.html
2. Ansible concepts. Ansible concepts - Ansible Community Documentation. (2024, November 20). https://docs.ansible.com/ansible/latest/getting_started/basic_concepts.html#id1
3. ansible-pull – Ansible Community Documentation. Ansible Documentation. URL: https://docs.ansible.com/ansible/latest/cli/ansible-pull.html (date of access: 08.12.2024)
4. Beyond Git: The other version control systems developers use - Stack Overflow. The Stack Overflow Blog - Stack Overflow. URL: https://stackoverflow.blog/2023/01/09/beyond-git-the-other-version-control-systems-developers-use/ (date of access: 15.12.2024).
5. cloud-init 24.4 documentation. cloud-init 24.4 documentation. URL: https://cloudinit.readthedocs.io/en/latest/ (date of access: 18.12.2024).
6. AWS - Auth Methods | Vault | HashiCorp Developer. AWS - Auth Methods | Vault | HashiCorp Developer. URL: https://developer.hashicorp.com/vault/docs/auth/aws (date of access: 18.12.2024).

7. Kostromin, R. O. (2020). Survey of software configuration management tools of nodes in heterogeneous distributed computing environment. The International Workshop on Information, Computation, and Control Systems for Distributed Environments. https://doi.org/10.47350/iccs-de.2020.15
8. S, L. (2022). Automation of server configuration using Ansible. International Journal for Research in Applied Science and Engineering Technology, 10(6), 4109–4113. https://doi.org/10.22214/ijraset.2022.44840

Dmitry Bulavin *PhD, associate professor*
Kharkiv National University named V. N. Karazin 61000
e-mail: d.bulavin@karazin.ua
ORCID: 0000-0002-4840-4763

Vladyslav Samoilenko *Student*
Kharkiv National University named V. N. Karazin 61000
e-mail: *vladyslav.samoilenko@student.karazin.ua;*
ORCID: 0009-0001-3217-4473

Self-Configuring Virtual Machine Environments Using Ansible Pull: A Review of Approaches and Challenges

Relevance. Automation of virtual machine (VM) configuration is a key component of modern IT infrastructure, ensuring scalability, reliability, and security. With the increasing adoption of cloud technologies, there is a growing demand for standardized approaches to infrastructure management that overcome the limitations of traditional methods such as manual configuration or standalone scripts.
Goal. To design an architecture for automating VM self-configuration based on Ansible Pull, integrated with tools like Terraform, Hashicorp Vault, and Git, to ensure flexibility, autonomy, and security.
Research methods. The proposed system utilizes an analytical approach to evaluate existing configuration tools, models architecture using Git repositories, Terraform, and Vault, and implements cloud-init to initialize Ansible Pull on VMs.
Results. The proposed architecture automates VM configuration processes, ensuring scalability and reducing dependency on centralized management servers. Ansible Pull implementation with tagging supports idempotent task execution, regular updates, and maintaining the desired state of the system.
Conclusions. Combining Ansible Pull with Terraform, Git, and Vault provides ease of implementation, scalability, and secure configuration handling. The approach is effective for dynamic environments requiring regular updates and aligns with modern DevOps practices.

Keywords: *Ansible Pull, configuration automation, Terraform, Vault, self-configuration, DevOps, Ansible*

Надійшла у першій редакції 22.08.2024, в останній - 05.10.2024.
The first version has been received on 22.08.2024, the final version - on 05.10.2024.