Title:

HashiCorp Vault Best Practices for Namespaces - when to configure root authentication or child namespaces

Author:

Amir A. Vetry

March 1, 2024

HashiCorp

Here are 8 steps to create a new branch in a version control system like Git.

1- (Using Git) Open Terminal or Command Prompt:

Navigate to the local repository on your machine using the terminal.

2- Ensure You're on the Main/Branch you Want to Branch From:

Run: **'git checkout main'**

3- Pull the Latest Changes:

Run: **'git pull origin main'** to ensure you have the latest changes from the main branch.

4- Create a New Branch:

Run: **'git checkout -b your-new-branch-name'**

This command creates and switches to a new branch named "your-new-branch-name."

5- Make Changes to your code or project within this new branch.

6- Add and Commit Changes:

Run:

git add .

git commit -m "Your commit message"

This stages and commits your changes to the new branch.

7- Push the New Branch to the Remote Repository:

Run: **‘git push origin your-new-branch-name’**

This pushes your new branch and its changes to the remote repository.

8- Switching Between Branches:

To switch back to the main branch: **‘git checkout main’**

Example (in bash):

```
# Create and switch to a new branch
git checkout -b feature/new-feature-branch

# Make changes, add, and commit
git add .
git commit -m "Adding a new feature"

# Push the new branch to the remote repository
git push origin feature/new-feature-branch
```


Remember to replace "your-new-branch-name" with a meaningful and descriptive name for your branch. This helps in maintaining a clear and organized version control history.

HashiCorp

Vault: Best Practices for Namespaces - when to configure root authentication or child namespaces

HashiCorp Vault is a tool for managing secrets and protecting sensitive data in IT environments.

Vault provides a feature called Namespaces.

Namespaces allows you to create isolated environments within Vault.

When deciding whether to configure root authentication or use child namespaces, it depends on the specific use case and security requirements of your organization.

Note: Next 3 slides will provide more details

When to configure root authentication

Root Authentication:

1- Root authentication is typically configured when we have a single Vault instance that serves as the central authority for managing secrets.

2- It is suitable for smaller deployments or situations where a single Vault instance can adequately handle the workload and security requirements.

3-Root authentication provides access to the entire Vault environment and all secrets stored within it.

4-This is a simpler configuration and may be more suitable for organizations with straightforward requirements.

When to configure use child namespaces

Child Namespaces:

1- Child namespaces are useful in larger and more complex deployments where multiple teams or departments require their own isolated environments within the same Vault cluster.

2- Each child namespace operates as a distinct Vault instance within the overall cluster, providing logical separation of secrets and policies.

3- Child namespaces enable different teams to manage their secrets independently while sharing the same underlying Vault infrastructure.

4-This approach is useful when there are distinct security boundaries or compliance requirements for different parts of the organization.

Considerations:

(root authentication vs child namespaces)

1- Scale and Complexity:

For smaller deployments or simple use cases, root authentication might be sufficient. As the deployment grows or becomes more complex, child namespaces offer better isolation and organization.

2- Security and Compliance:

Consider the security and compliance requirements of your organization. Child namespaces may help enforce separation of duties and meet specific compliance standards.

3- Collaboration and Isolation:

If different teams need to collaborate while maintaining
separate control over their secrets, child namespaces provide a way to achieve this balance.

root authentication vs child namespaces ?

In summary:

Choose **root authentication** for simpler deployments or situations where a single Vault instance is sufficient.

Choose **child namespaces** when you need logical separation, isolation, and more complex access control for different parts of your organization within the same Vault cluster.

The decision should align with the scale, complexity, and security requirements of your specific use case.

Two HashiCorp products working together.

HashiCorp provides a suite of infrastructure automation tools.

Two of their popular products that often work together are **Terraform** and **Vault**.

1- Terraform:

Terraform is an open-source Infrastructure as Code (IaC) tool that allows you to define and provision infrastructure in a declarative configuration language.

With Terraform, we can describe your infrastructure using HashiCorp Configuration Language (HCL) or JSON, and then Terraform automates the process of creating, updating, and destroying infrastructure components.

2- Vault:

Vault is a secrets management and data protection tool. It helps you secure, store, and tightly control access to tokens, passwords, certificates, API keys, and other secrets in modern computing.

Integration:

- Terraform and Vault can be seamlessly integrated to enhance the security and management of infrastructure provisioning.

- **Dynamic Secrets:** Vault can generate dynamic, short-lived credentials (such as database credentials) on-demand. Terraform can leverage these dynamic secrets to provision resources securely without hardcoding sensitive information.

- **Secrets Injection:** Vault can be used to inject secrets into infrastructure deployments orchestrated by Terraform. This ensures that sensitive information is not exposed in the Terraform configurations or stored in version control.

- **Key Management:** Vault can manage encryption keys, and Terraform can be configured to use Vault for key retrieval or encryption purposes, ensuring secure handling of cryptographic material.

Example Workflow:

1- Define Infrastructure in Terraform: Use Terraform to define your infrastructure, specifying resources and configurations.

2- Secrets Management with Vault: Store sensitive information such as API keys, passwords, or certificates securely in Vault.

3- Integration in Terraform: Use Terraform providers to integrate with Vault. Utilize Vault-generated dynamic secrets or inject secrets securely during the Terraform execution.

4- Secure Communication: Ensure secure communication between Terraform and Vault, typically using tokens and secure communication protocols.

5- Policy Management: Leverage Vault policies to control access to secrets and infrastructure resources based on roles and permissions.

By combining Terraform and Vault,
organizations can achieve a secure and automated infrastructure provisioning process while maintaining strong control over sensitive information.

This integration aligns with HashiCorp's goal of providing tools for a consistent workflow in infrastructure management and security.

Assignment 2 - Script

HashiCorp

Create a bash script setup.sh:

Sets an environment variable with your github username in /root/.bash_profil

```
#!/bin/bash

# Check if the script is running as root
if [ "$EUID" -ne 0 ]; then
  echo "Please run this script as root."
  exit 1
fi

# Your GitHub username
github_username="your_github_username"

# Set the environment variable in /root/.bash_profile
echo "export GITHUB_USERNAME=$github_username" >> /root/.bash_profile

# Source the .bash_profile to apply the changes
source /root/.bash_profile

echo "GitHub username environment variable set successfully."
```

Assignment 2 – Script continue…

Create a bash script setup.sh that:

- Sets an environment variable with your github username in /root/.bash_profile
- Writes the following code to a file called config.hcl

```
#!/bin/bash

# Set GitHub username as an environment variable in /root/.bash_profile
echo 'export GITHUB_USERNAME="your_username_here"' >> /root/.bash_profile
source /root/.bash_profile

# Write code to config.hcl
cat <<EOL > config.hcl
# Sample configuration
github_username = "$GITHUB_USERNAME"
# Add more configurations as needed
EOL

echo "Setup complete! Environment variable and configuration file created."
```

Assignment 2 – Script continue…

Make the script executable by running:

bash

chmod +x setup.sh

Then, you can execute the script using:

bash

./setup.sh

Assignment 2 – Script continue…

Create a bash script setup.sh that:

. Sets an environment variable with your github username in /root/.bash_profile

. Writes the following code to a file called config.hcl

```
cluster_addr  = "https://<HOSTNAME>:8201"
api_addr      = "https://<HOSTNAME>:8200"
disable_mlock = true
```

Replaces <HOSTNAME> value vault-server.hashicorp.com in config.hcl

```
#!/bin/bash

# Set environment variable in /root/.bash_profile
echo 'export GITHUB_USERNAME="your-github-username"' >> /root/.bash_profile

# Write the configuration code to config.hcl
cat <<EOF > config.hcl
cluster_addr  = "https://<HOSTNAME>:8201"
api_addr      = "https://<HOSTNAME>:8200"
disable_mlock = true
EOF

# Replace <HOSTNAME> value in config.hcl
sed -i 's/<HOSTNAME>/vault-server.hashicorp.com/g' config.hcl

echo "Setup completed successfully."
```

Assignment 2 – Script continue…

After creating the script, you can make it executable and run it using the following commands:

bash

chmod +x setup.sh
./setup.sh

Thank you