Vault

Vault can be thought of as a Swiss Army knife for security. Vault securely stores and tightly controls access to secrets such as tokens, passwords, certificates, API keys, connection strings, etc. It can be used for secrets management, privilege access management, and as encryption as a service. Vault manages leasing, key revocation, key rolling, and provides granular auditing. It supports a number of secret backends including AWS, PostgreSQL, SSH, PKI, and generic secrets. Vault provides a break-glass procedure to effectively respond if security has been compromised. Vault is used at Direct Supply to manage secrets and sensitive information needed by users and applications running in the cloud and to control access to these applications and their resources. Active Directory is used to authenticate users with Vault

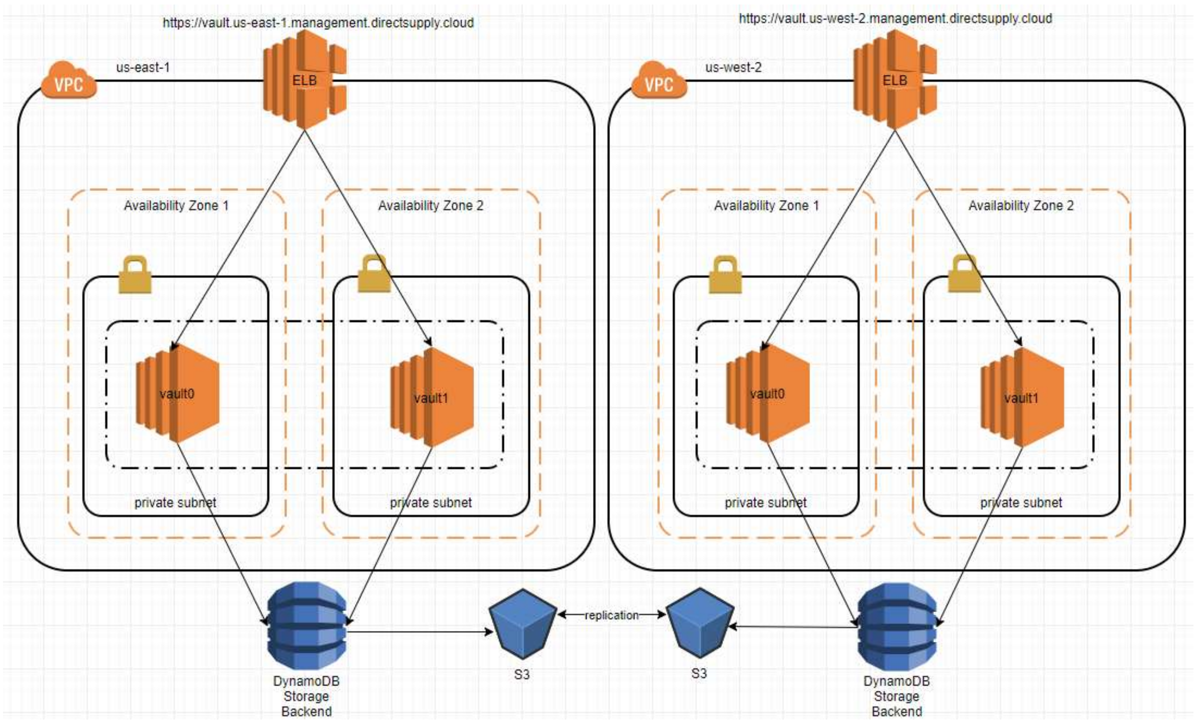
Server Architecture

Highly Available Deployment

A separate cluster of Vault servers is deployed into the Management VPC of each account (e.g., Production, Sandbox, Testing) in each the `us-east-1` and `us-west-2` regions. Servers in the cluster are deployed to multiple availability zones for HA purposes. A single server in a Vault cluster is identified as the active server. All traffic is sent to it by the elastic load balancer. If the active server becomes unresponsive, one of the inactive servers becomes active and traffic is sent to it.

Each cluster is backed by a DynamoDB table. For DR purposes, the DynamoDB table is replicated to a second AWS region.

The `us-east-1` and `us-west-2` deployments are to be considered *separate databases*. If your application requires multi-region DR then you will have to ensure you deploy your secrets to both environments.



Auto Scaling

The Vault clusters are auto-scaling groups that are configured with a desired node count of 2. This ensures that there will always be a vault server available in case one becomes unresponsive. When a Vault server is started, it is sealed. Secrets cannot be obtained from the Vault server until it is unsealed. During normal operation all running Vault servers will be unsealed. If the active Vault server becomes unresponsive, an inactive Vault server will become active. Since it is already unsealed, no manual intervention is required to ensure continuous operation of the Vault cluster.

Master Keys

Vault clusters are configured to require 3 of 10 keys to unseal. All keys are stored in PMP which allows PMP administrators to view all keys and unseal in cases where all other keyholders are unavailable to provide their unseal keys. PMP admins who use other keyholders' keys to unseal will need approval from senior leadership. Key distribution will be maintained by the information security team.

Access Policies

Vault Policies are the basis for access control inside of Vault. Access to any resource in Vault is granted by the policies that are assigned to the user or resource. They can be used to declaritively grant or deny access to secrets. They consist of the path the policy applies to and the capabilities they grant (or deny) to that path. Please email the "Vault Engineering Review" group for review when creating or editing a policy.

Example Vault policy:

```
Key     Value
---     -----
name    engineer-all
rules   path "v1/*" {
        capabilities = ["create", "read", "update", "list", "delete"]
}
```

Auth Backends

Please email the "Vault Engineering Review" group for review if you are creating or editing any auth types.

Vault Login: OIDC

Vault OIDC Auth Backend allows Direct Supply partners to authenticate to Vault using their MTOLYMPUS domain credentials through Azure AD with 2FA. Vault allows mapping of Active Directory domain groups to its access policies allowing us to control access to secrets in Vault via AD group membership.

Additional information on the OIDC auth method can be had here.

Default Active Directory Groups

Custom Vault policies can be created as needed for granular access control. However, we have configured the following default groups.

- `Vault_Admin` - Grants admin access to Vault on all tiers: Production, Testing, and Sandbox
- `Vault_Engineer_Production_All` - Grants SRE release access to Vault secrets in Production account
- `Vault_Engineer_Production_Read` - Read access to Vault secrets in Production account
- `Vault_Engineer_Testing_All` - Engineer access to Vault secrets in Testing account
- `Vault_Engineer_Testing_Read` - Read access to Vault secrets in Testing account
- `Vault_Engineer_Sandbox_All` - Engineer access to Vault secrets in Sandbox account
- `Vault_Engineer_Sandbox_Read` - Read access to Vault secrets in Sandbox account

In Vault, these are the specific capabilities assigned to the default AD groups:

	Sandbox	**Sandbox**	**Sandbox**	**Testing**	**Testing**	**Testing**	**Production**	**Production**	**Pro**
AD Groups	"read", "list"	"create", "read", "update", "list", "delete"	"create", "read", "update", "list", "delete", "sudo"	"read", "list"	"create", "read", "update", "list", "delete"	"create", "read", "update", "list", "delete", "sudo"	"read", "list"	"create", "read", "update", "list", "delete"	"cre "rea "up "list "de "su
Vault_Admin			x			x			x
Vault_Engineer_Sandbox_Read	x								
Vault_Engineer_Sandbox_All		x							
Vault_Engineer_Testing_Read				x					
Vault_Engineer_Testing_All					x				
Vault_Engineer_Production_Read							x		
Vault_Engineer_Production_All								x	

AWS

Vault AWS Auth Backend allows access to be granted to resources running in AWS based on their IAM role. The IAM role of a specific resource or group of resources is attached to a Vault policy by configuring a Vault AWS role. This allows applications runing in AWS to read/create secrets in Vault.

This is the authentication method used by the deployment bastion for accessing secrets and managing Vault configurations.

Example Vault AWS role:

```
Key                             Value
---                             -----
allow_instance_migration        false
auth_type                       iam
bound_account_id
bound_ami_id
bound_iam_instance_profile_arn
bound_iam_principal_arn         arn:aws:iam::689557391783:role/devops-production-deployment-bastion
bound_iam_principal_id          AROAIRSMPK6TQFZ6RUHFM
bound_iam_role_arn
bound_region
bound_subnet_id
bound_vpc_id
disallow_reauthentication       false
inferred_aws_region
inferred_entity_type
max_ttl                         129600
period                          0
policies                        [default deployment-bastion-root]
resolve_aws_unique_ids          true
role_tag
ttl                             0
```

UserPass

Vault UserPass Auth Backend is simple username and password authentication to Vault. Only one username and password is configured and intended to be used for emergency administrative access if Vault is not able to authenticate users via Active Directory. This admin username and password are stored in Password Manager Pro.

Secret Backends

Vault Path Structure

Vault paths are user defined locations within Vault that secrets are stored. Vault paths should be alpha-numeric with no spaces or hyphens. Camel-case should be used throughout Vault.

There are two top-level paths that should be utilized for storing application configuration and secrets. `/config` and `/secrets` utilize Vault's version 2 KV secrets engine, which provides versioned storage.

The base path for resource specific configuration and secrets should abide by the following format:
Secrets: `secrets/[application]/[environment]/[role]`
Configuration: `config/[application]/[environment]/[role]`

Generic Key/Value Secrets

Vault Generic Secrets Backend is Vault's primary backend and is used for storing secrets in key/value pairs. Generic secrets shall be stored in Vault using the path structure in the previous section.

New secrets can be added to Vault via the UI:

- https://vault.us-east-1.management.directsupply-sandbox.cloud
- https://vault.us-east-1.management.directsupply-testing.cloud
- https://vault.us-east-1.management.directsupply.cloud

Reading V1 Key/Values

Vault generic secrets can be accessed by running `vault read {secretPath}`. See the example below for expected input and output.

```
[jakeg@ip-10-144-82-44 ~]$ vault read v1/test
Key                        Value
---                        -----
refresh_interval           768h0m0s
value                      goPackgo
```

Reading V2 Key/Values

Key/value stores utilizing versioned v2 storage can be accessed by running `vault kv get {secretPath}`. See the example below for expected input and output

```
vault kv get config/test
====== Metadata ======
Key                 Value
---                 -----
created_time        2019-07-08T20:52:57.181813883Z
deletion_time       n/a
destroyed           false
version             1

=== Data ===
Key     Value
---     -----
foo     bar
```

Creating and Storing SSH Key Pairs

SSH key pairs can be used for many things including storing in AWS to secure SSH to Linux or for retrieving Windows Administrator passwords for AWS EC2 instances. In this case, you can create your own key pairs for sandbox but you must request SRE to generate the versions for the testing and production accounts. From a Windows Git Bash prompt or a Linux/macOS terminal:

```
# Set this variable for whichever AWS account you are generating the key
account=sandbox
# The following line will generate a new public and private key pair named sandbox and sandbox.pub
ssh-keygen -t rsa -C "7400@directs.com" -b 4096 -N "" -f $account
# The following line will encode the public and private keys in JSON formatting and save to sandbox.jsor
# NOTE: This requires jq to be installed ('choco install jq' for Windows)
echo "{\"private_key\":$(cat $account | jq -sR .),\"public_key\":$(cat $account.pub | jq -sR .)}" > $acc
# On Windows Git Bash, this will copy the contents of sandbox.json to the clipboard
cat $account.json | clip
```

When having SRE generate the keys have them send the `*.pub` (public key) file back to you. You will then use the contents of that file as the public_key input to this terraform module:

https://gitlab.directsupply.cloud/terraform-modules/ssh-keypair

Use the Vault web UI to insert the private keys into Vault. Access it directly at:

- sandbox - https://vault.us-east-1.management.directsupply-sandbox.cloud
- testing - https://vault.us-east-1.management.directsupply-testing.cloud
- production - https://vault.us-east-1.management.directsupply.cloud

To log in and add your secret:

1. Select "OIDC" from the "Method" dropdown and log in.
2. Follow the UI to find your secrets path
3. Create a new version (if a v2 secrets engine) or edit the existing secret (if a v1 secrets engine)
4. Paste in the contents of the private key file. Newlines should be preserved.

Warning

Once the Vault secrets have been entered and you (the developer) have the public key, make sure to delete the generated keys and json from the disk, as shown below.

```
# Delete generated files once private key is in vault and public key is in terraform.
rm $account*
```

SSH One-Time Passwords

Vault SSH One-Time Password backend allows users to request ssh passwords to login to EC2 instances. These passwords can one be used once and have a specified TTL. Vault allows you to configure different SSH roles for different linux users on the target machines. The roles are scoped to a specific CIDR range, port, and TTL.

Example vault ssh role configuration:

```
Key                        Value
---                        -----
allowed_users              read-user
cidr_list                  10.144.0.0/15
default_user               read-user
exclude_cidr_list
key_type                   otp
port                       22
```

Custom Vault SSH roles can be created as needed for granular access control. However, we have configured the following default roles.

	Sandbox	Sandbox	Testing	Testing	Production	Production
AD Groups	read-user	ec2-user	read-user	ec2-user	read-user	ec2-user
Vault_Admin	x	x	x	x	x	x
Vault_Engineer_Sandbox_Read	x					
Vault_Engineer_Sandbox_All		x				
Vault_Engineer_Testing_Read			x			
Vault_Engineer_Testing_All				x		
Vault_Engineer_Production_Read					x	
Vault_Engineer_Production_All						x

Database Credentials

The Vault Database secret backend allows users to request temporary credentials to databases based on configured Vault database backends and roles. A database backend is configuration data that Vault uses to connect to and manage users on a given database instance. It consist of the allowed database roles and connection string. A database role is configured for creating database users and specifying the permissions those users will have on the given database. Roles consist of the SQL creation statements that control which schema and what access the accounts will have, the database itself, and a TTL for the temporary credentials.

Example Vault database backend configuration:

```
Key                     Value
---                     -----
allowed_roles           [*]
connection_details      map[connection_url:postgresql://dbup_dev:{password}@ogm3-sandbox-database.czrcxz
plugin_name             postgresql-database-plugin
```

Example Vault database role configuration:

```
Key                     Value
---                     -----
creation_statements     CREATE ROLE "{{name}}" WITH LOGIN PASSWORD '{{password}}' VALID UNTIL '{{expirat
db_name                 order_guide_management
default_ttl             3600
max_ttl                 86400
renew_statements
revocation_statements
rollback_statements
```

Security

Auditing

Vault Audit Backend

The Vault audit backed is configured through Terraform as part of the vault-config project. The audit backend is configured to log Vault activity to local file, `/var/log/vault/audit.log`. This file will only exist on the active Vault server.

CloudWatch Configuration

Vault audit logs are pushed to CloudWatch from the Vault servers using AWS CloudWatch Logs Agents. The log streams are configured locally in `/etc/awslogs/templates/vault_logs.conf` which is included by default in the Vault server AMI. The log streams are configured to publish to the 'vault-sandbox-app-vault' log group which is created with Terraform as part of the vault-server project. The retention policy on that log group is 10 years.

Local Log Rotation

The Vault servers use logrotate to rotate audit logs daily. Seven days worth of logs are retained locally on the Vault servers. Logrotate creates a new, empty logfile and sends a SIGHUP command to Vault which initiates a new connection to the log file. This config can be found in `/etc/logrotate.d/vault` and is included by default in the vault-server ami.

Secure Deployment Pipeline

The Vault infrastructure and configuration code is deployed securely using GitLab Enterprise CI/CD to store, build, and package the code and Octopus Deploy to deploy it.

Break Glass Procedure

The break glass procedure is followed if the integrity of the infrastructure is determined to be compromised. In order to prevent future access to the Vault cluster, all Vault servers in the Vault cluster are sealed via the Vault CLI by a user that has elevated access to the vault servers such as SRE.
Once the integrity of the infrastructure has been restored, the Vault servers can be unsealed following the standard Vault unseal procedure.

If it is known that the compromise is isolated to a specific application, environment, or role, a less drastic approach of revoking leases on secrets may be employed. For example, if the compromise is limited to a particular environment within an application, leases on all secrets for the `[version]/[environment]/[application]` can be revoked:

```
vault revoke -prefix v1/dev/ogm
```

> Note
>
> This operation requires `sudo` capability, and it does not affect generic secrets since they do not have leases. In the event of an isolated compromise, the procedure to shutdown all Vault servers may be the first step. Once the response team is in place, the Vault servers can be unsealed and the specific leases can then be revoked.

FMEA

Vault's failure modes and effects analysis can be found here: Vault FMEA

Guides

- Best Practices

- SSH to a Server Using a Vault OTP
- Request Temporary Database Credentials From Vault
- Grant AWS Resources Access to Vault
- Vault Runbook
- Upgrade Procedure