SecureVault Getting Started Guide

Document ID: PRD-SV-001 **Last Updated:** 2024-02-20 **Owner:** SecureVault Product Team **Classification:** Public

---

Welcome to SecureVault

SecureVault is NovaTech's secrets management platform. Securely store, access, and manage sensitive data like API keys, passwords, certificates, and encryption keys with enterprise-grade security.

Quick Start Checklist

- ☐ Create your SecureVault account
- ☐ Set up your first vault
- ☐ Store your first secret
- ☐ Access secrets programmatically
- ☐ Set up access policies

Why SecureVault?

The Problem

- Secrets in code repositories
- Passwords in configuration files
- Keys shared via insecure channels
- No audit trail of secret access
- Difficult secret rotation

The Solution

SecureVault provides: - **Centralized storage:** One place for all secrets - **Access control:** Fine-grained permissions - **Audit logging:** Complete access history - **Dynamic secrets:** Auto-rotating credentials - **Encryption:** AES-256-GCM at rest

---

Step 1: Create Your Account

Sign Up

1. Visit securevault.novatech.com
2. Click **Start Free Trial**
3. Enter email and create password
4. Verify email address
5. Enable two-factor authentication (required)
6. Complete organization setup

Plan Selection

Plan	Secrets	Users	Price
Free	25	3	$0
Team	1,000	25	$99/mo
Business	10,000	100	$499/mo
Enterprise	Unlimited	Unlimited	Custom

Step 2: Set Up Your First Vault

What is a Vault?

A vault is a secure container for secrets. Organize vaults by: - Application (e.g., “web-app”, “mobile-api”) - Environment (e.g., “production”, “staging”) - Team (e.g., “engineering”, “devops”)

Create a Vault

1. From dashboard, click **New Vault**
2. Enter vault name (e.g., “production-api”)
3. Select region for data residency
4. Configure initial permissions
5. Click **Create**

Vault Settings

- **Name:** Descriptive identifier
- **Description:** Purpose and contents

- **Region:** Data storage location (us-west-2, eu-west-1, etc.)
- **Retention:** How long to keep deleted secrets (7-90 days)

Step 3: Store Your First Secret

Via Web UI

1. Open your vault
2. Click **New Secret**
3. Enter secret details:
   - **Path:** `myapp/database/password`
   - **Value:** Your secret value
   - **Metadata:** Optional key-value pairs
4. Click **Save**

Via CLI

Install the SecureVault CLI:

```
# macOS
brew install novatech/tap/securevault

# Linux
curl -fsSL https://securevault.novatech.com/install.sh | bash

# Windows
choco install securevault
```

Authenticate:

```
securevault login
```

Store a secret:

```
securevault secret put myapp/database/password value="supersecret123"
```

Via API

```
curl -X POST https://api.securevault.novatech.com/v1/secrets/myapp/database/password \
  -H "Authorization: Bearer $SECUREVAULT_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"value": "supersecret123"}'
```

Step 4: Access Secrets Programmatically

Using the SDK

Node.js:

```
const SecureVault = require('@novatech/securevault');

const client = new SecureVault({
  token: process.env.SECUREVAULT_TOKEN,
});

async function getSecret() {
  const secret = await client.secrets.get('myapp/database/password');
  console.log(secret.value);
}
```

Python:

```
from securevault import SecureVaultClient

client = SecureVaultClient(token=os.environ['SECUREVAULT_TOKEN'])

secret = client.secrets.get('myapp/database/password')
print(secret.value)
```

Go:

```
package main

import (
    "fmt"
    "github.com/novatech/securevault-go"
)
```

```go
func main() {
    client := securevault.NewClient(os.Getenv("SECUREVAULT_TOKEN"))

    secret, _ := client.Secrets.Get("myapp/database/password")
    fmt.Println(secret.Value)
}
```

Using Environment Variables

SecureVault can inject secrets as environment variables:

```
securevault exec --secrets myapp/database -- node app.js
```

Your app receives:

```
DATABASE_PASSWORD=supersecret123
```

Step 5: Set Up Access Policies

Understanding Policies

Policies control who can access what secrets: - **Path-based:** Define paths users can access - **Capabilities:** read, write, delete, list - **Conditions:** Time-based, IP-based restrictions

Create a Policy

1. Go to **Policies → New Policy**
2. Enter policy name (e.g., "developer-read")
3. Define rules:

```
# Allow reading secrets under myapp/
path "myapp/*" {
  capabilities = ["read", "list"]
}

# Deny access to production secrets
path "myapp/production/*" {
  capabilities = ["deny"]
}
```

4. Click **Save**

Assign Policy to User

1. Go to **Users**
2. Select user
3. Click **Assign Policies**
4. Select policies
5. Click **Save**

Common Policy Patterns

Developer (read non-production):

```
path "myapp/development/*" {
  capabilities = ["read", "list", "write"]
}

path "myapp/staging/*" {
  capabilities = ["read", "list"]
}
```

DevOps (full access):

```
path "myapp/*" {
  capabilities = ["create", "read", "update", "delete", "list"]
}
```

Application (read-only):

```
path "myapp/production/{{identity.entity.name}}/*" {
  capabilities = ["read"]
}
```

Secret Organization

Recommended Structure

```
myapp/
   development/
      database/
         host
```

```
        username
        password
    api/
        key
staging/
    ...
production/
    ...
```

Naming Conventions

- Use lowercase
- Separate with forward slashes
- Be descriptive but concise
- Include environment in path

Good: - `myapp/production/database/password` - `payment-service/stripe/api-key`

Avoid: - `prod_db_pw` (not descriptive) - `my-super-secret-key` (no context)

Authentication Methods

Token Authentication

Best for: Development, scripts, CI/CD

```
export SECUREVAULT_TOKEN="svt_abc123..."
securevault secret get myapp/database/password
```

AppRole Authentication

Best for: Applications, automated systems

1. Create AppRole:

```
securevault approle create myapp-backend
```

2. Get Role ID and Secret ID:

```
securevault approle get-role-id myapp-backend
securevault approle get-secret-id myapp-backend
```

3. Authenticate:

```
securevault login -method=approle \
  role_id=$ROLE_ID \
  secret_id=$SECRET_ID
```

OIDC Authentication

Best for: User access, SSO integration

1. Configure OIDC provider (Okta, Auth0, etc.)
2. Create OIDC auth method in SecureVault
3. Map groups to policies
4. Users authenticate via SSO

Dynamic Secrets

What Are Dynamic Secrets?

Dynamic secrets are generated on-demand and automatically expire. Benefits: - No shared credentials - Automatic rotation - Audit trail per access - Reduced blast radius

Database Credentials

Configure dynamic database credentials:

1. Go to **Dynamic Secrets → Databases**
2. Click **Add Database**
3. Configure connection:
   - Host, port, database
   - Admin credentials (for SecureVault to create roles)
4. Create role:

   ```
   CREATE ROLE "{{name}}" WITH LOGIN PASSWORD '{{password}}' VALID UNTIL '{{expiration}}';
   GRANT SELECT ON ALL TABLES IN SCHEMA public TO "{{name}}";
   ```

5. Request credentials:

```
securevault dynamic database/myapp/creds/readonly
```

Output:

```
{
  "username": "v-approle-readonly-abc123",
  "password": "generated-password-xyz",
  "ttl": "1h"
}
```

Best Practices

Security

1. **Enable MFA** for all user accounts
2. **Use AppRole** for applications (not tokens)
3. **Apply least privilege** - minimal permissions needed
4. **Rotate secrets** regularly (automate with dynamic secrets)
5. **Audit access** - review logs monthly

Organization

1. **Structure paths** by app/environment
2. **Use policies** consistently across teams
3. **Document** secret ownership and purpose
4. **Tag secrets** with metadata (owner, rotation schedule)

Operations

1. **Monitor** secret access patterns
2. **Alert** on unusual access
3. **Backup** vault configuration
4. **Test** disaster recovery procedures

Integrations

CI/CD Integration

DevPipeline:

```
stages:
  - name: deploy
    steps:
      - uses: securevault/get-secrets
        with:
          secrets: |
            myapp/production/database/password -> DATABASE_PASSWORD
            myapp/production/api/key -> API_KEY
      - run: ./deploy.sh
```

GitHub Actions:

```
- uses: novatech/securevault-action@v1
  with:
    token: ${{ secrets.SECUREVAULT_TOKEN }}
    secrets: |
      myapp/production/database/password -> DATABASE_PASSWORD
```

Kubernetes Integration

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    securevault.novatech.com/inject: "true"
    securevault.novatech.com/secrets: "myapp/production/database"
spec:
  containers:
    - name: myapp
      image: myapp:latest
```

Troubleshooting

Common Issues

"Permission denied" - Check user has policy with required capabilities - Verify path matches policy exactly - Check token hasn't expired

"Secret not found" - Verify secret path is correct - Check secret hasn't been deleted - Ensure vault exists

"Authentication failed" - Verify token is valid - Check AppRole credentials - Ensure MFA is completed

---

Getting Help

- **Documentation:** docs.securevault.novatech.com
- **API Reference:** api.securevault.novatech.com
- **Support:** support@novatech.com
- **Status:** status.novatech.com

Related Documents: Access Control Guide (PRD-SV-010), CLI Reference (PRD-SV-020), Encryption Overview (PRD-SV-005)