



Access Model Workshop

CyberArk Conjur

LESSON OBJECTIVES

This lesson provides an introduction to the CyberArk Conjur solution.

Upon completion of this lesson the participant will be able to:

- Policy Administration Overview
- Users & Groups Management
- Hosts & Layers Management
- Secrets Management
- Best practices

KEY CONCEPTS

TERMINOLOGY

| | |
|--------------------------|---|
| USERS: | Human user account either native to Conjur or transparently integrated through Directory Services |
| GROUPS: | Collection of Users, Groups, and Roles |
| HOSTS: | Non-human user account (“robots” or “machines”) used for applications, processes, services, jobs, and automation |
| MACHINE IDENTITY: | Non-human actor (“Hosts”) requiring authentication and authorization |
| LAYERS: | Non-human Infrastructure permissions layer to organize your system into broad permission groups |
| RESOURCES: | Records on which permissions are defined. Partitioned by “kind” such as “group”, “host”, “file”, “environment”, or “variable” |
| ROLES: | An actor in a system (human or non-human) which receive permission grants |
| PERMISSIONS: | Defines privilege for “Roles” and associated permissions on given “Resource” |
| POLICIES: | Collection of “Objects”, “Annotations”, “Permissions”, and “Roles” used define security rules |
| SECRETS: | Passwords, SSH keys, certificates, API tokens, cloud identities stored into Conjur |
| VARIABLES: | Containers to store secrets. Each secret is encrypted (AES-256) and versioned |
| HOST FACTORY: | Conjur service to manage machine identity using limited-time tokens for authentication |

CONJUR SERVICES OVERVIEW



AUTHENTICATION

- Human & Non-Human Users
- “authn” (login, logout, whoami, authenticate)
- API Keys & Authentication Tokens
- 3rd Party Authentication



AUTHORIZATION

- Role-Based Access Control (RBAC)
- Engine to enforce access control
- Role, Resource, Permission, Global Permissions
- Import External Sources Support



DIRECTORY

- Engine to manage users, groups, hosts, and layers
- Enable AD/LDAPS integration
- LDAP Sync for exporting directory services data
- LDAP Authenticator to authenticate directory services



PUBLIC KEYS

- Manage, store, and distribute public keys
- Added and fetched in OpenSSH format
- Methods: add, delete, names, and show



HOST FACTORY

- Service to host limited-time tokens to provide authentication to dynamic VMs/Containers
- Manage policies and maintain consistency
- High performance and security authorization engine



AUDIT

- Manage centralized audit database
- Immutable permission events and records across entire infrastructure
- Easy-to-read format or JSON stream

HUMAN IDENTITY: USERS & GROUPS

Users

- **Human user account** used for authentication and authorization
- Unique “login” (username/ID), API Key, and optional “password” per user
- API Key = randomly generated secret assigned by Conjur
- API Keys can be redeemed using “login” and “password”
- **Common Workflow:** Often created and provisioned by syncing with LDAP



Groups

- Collection of “Users”, “Groups”, and/or “Roles”
- Each Group is assigned a unique ID
- Adding a Role to a Group gives the Role all the permissions held by the Group
- Records can be assigned to Groups for ownership and survivorship



MACHINE IDENTITY: HOSTS & LAYERS

Hosts

- **Non-human user account** used for authentication and authorization
- Unique Host ID and API Key assigned for authentication
- API Key = randomly generated secret assigned by Conjur
- **Common Workflow:** Often created and provisioned with automation tools
- **Examples:** server, virtual machine, application container, serverless Lambda (AWS)



Layers

- Logical container for Hosts which perform similar jobs or functions
- Each Layer is assigned a unique ID
- Adding a Role to a Layer gives the Role all the permissions held by the Layer
- **Common Workflow:** A Layer owns a Variable granting member Hosts access
- **Examples:** Database Passwords, Application API Keys, SSH Keys



MACHINE IDENTITY STORAGE

FILE STORAGE

- Information includes unique identifier, secret key, and configuration information
- NOTE:** “/etc/conjur.identity” contains the machine’s API key and should be kept secret. Good practice is to symlink this file from /dev/shm.
- Remaining files do not contain sensitive information and can be freely shared

| file | type | contains | security concerns |
|----------------------|-------------|---------------------------|-------------------------|
| /etc/conjur.identity | netrc | identifier and secret key | only accessible by root |
| /etc/conjur.conf | yaml | configuration | only writable by root |
| /etc/conjur.pem | certificate | info from conjur server | only writable by root |

VARIABLE STORAGE

- Apply machine identity via environment variables
- USE CASE:** Applications or tools that accept configuration through the environment (Docker containers, Jenkins jobs, Heroku apps, etc.)

| variable | description | example |
|------------------------|--|--------------------------------------|
| CONJUR_ACCOUNT | account specified during Conjur setup | myorg |
| CONJUR_APPLIANCE_URL | Conjur HTTPS endpoint | https://conjur.myorg.com/api |
| CONJUR_AUTHN_API_KEY | host API key | sb0ncv1yj9c4w2e9pb1a2s... |
| CONJUR_AUTHN_LOGIN | host identity | host/production/redis001 |
| CONJUR_AUTHN_TOKEN | Conjur JSON access token, Base64 encoded | eyJkYXRhbjNTdjOWJkNWQ1ZDgyYTU0In0... |
| CONJUR_POLICY_ID | policy namespace (for variables) | redis-v1 |
| CONJUR_SSL_CERTIFICATE | public Conjur SSL cert | -----BEGIN CERTIFICATE-----... |

CONJUR_AUTHN_TOKEN requires CLI version 5.2.0 or greater.

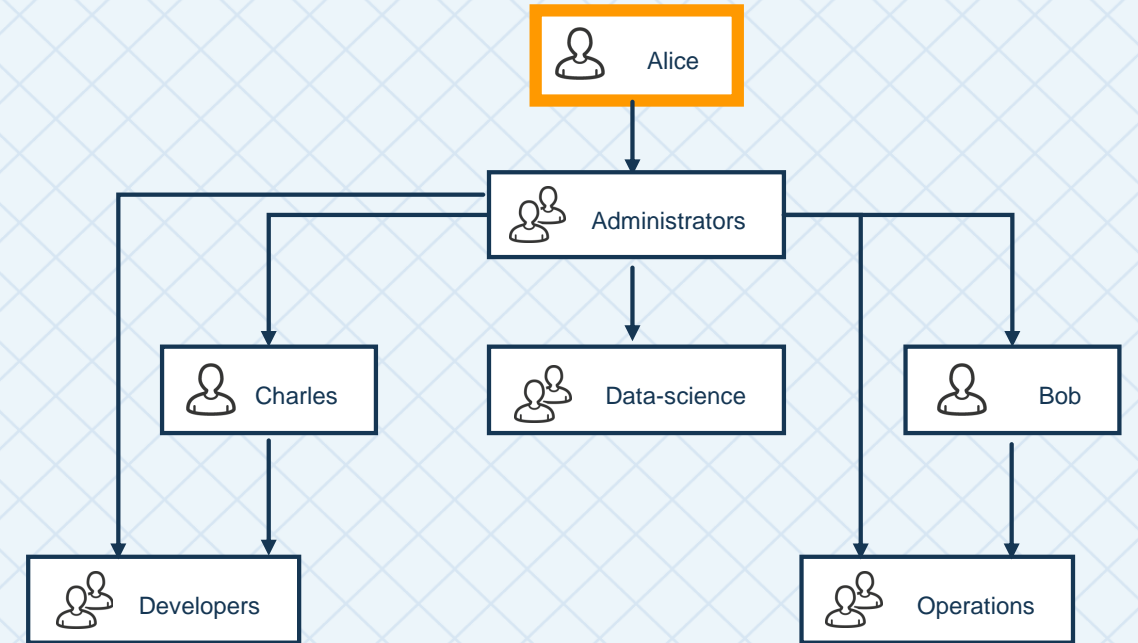
ROLES

Assign & Receive Privileges

- Roles primary purpose is to assign and receive privileges
- Represent human user, group, or non-human user (virtual machine, process, application, etc.)
- Roles can be granted to other roles
- Roles granted are fully inherited

Admin Flag

- Administration function assigned to each role
- Allows ability to “grant” or “revoke” roles to other roles



| | |
|-----------------|--|
| Example: | group 'developers' is granted to user 'alice' |
| Result: | User “alice” gains all privileges of the granted role (group ‘developers’) |

ROLE-BASED ACCESS CONTROL

- Policy Model for Authorization and Access to Secrets
- Administration, Creation, and Maintenance of Policy Rules
- Easy, Scalable Way to Scope, Map, and Organize Access Structure
- Does Role “X” have Privilege “Y” to Access Resource “Z”?
- Default deny access to resources (requires explicit permission provisioning)
- **Security Policy as Code** using standard declarative descriptions for infrastructure

The screenshot displays the CyberArk console interface for a specific secret resource. The breadcrumb navigation at the top reads "Secrets > dev/db/mysql/cars/secret".

Details

Owner: dev/db/mysql/cars/secret
Created by: dev/db/mysql
MIME type:
Kind: variable
Versions:
Annotations: Descriptions credential
Instance cars
conjur/kind password

Resource Permissions

| Role | Privileges | Resource |
|--------------|-----------------------|--------------------------|
| dev/db/mysql | owner | dev/db/mysql/cars/secret |
| dba | execute, read, update | dev/db/mysql/cars/secret |
| dev | execute, read | dev/db/mysql/cars/secret |
| dev/srv/nix | execute, read | dev/db/mysql/cars/secret |

RESOURCES

Protected Assets

- Resources define the entities on which permissions are defined
- Unique arbitrary string to identity protected assets (secrets)
- Privileges assigned once resource is created
- Supports custom association between resource and privilege assignment

Grant Option

- Special privilege to enable role to assign privilege to other roles

Ownership

- Each resource has designated “owner” granting full privileges
- Owners can grant privileges on resources to other roles
- Owners can transfer ownership to another role

EXAMPLE:

Encrypted Database Password

read – obtain properties and metadata

execute – retrieve value of secret

update – change value of secret



WARNING:

Transferring ownership to another role might result in loss of privileges to original ownership

SECRETS

- What is a secret?
 - Passwords, SSH Keys, API Keys, SSL Digital Certificates, Cloud Credentials
 - Stored into Conjur Database as a **variable**
 - Unique AES-256 bit encryption stored at-rest per variable
 - Each variable stored is versioned with a unique encrypted value
 - Updates/Changes to variable increments the version schema
 - Current version fetched by default (All versions available)
- Secrets Management
 - Creation & Retrieval performed via Conjur CLI and/or REST API
 - Each secret variable possesses role-based access-control permissions
 - **Read** – list variable exists
 - **Write** – modify variable value
 - **Execute** – fetch variable value
 - Full accounting / audit trail logging recorded for each variable transaction



CONJUR POLICY

- Define security rules into declarative format to easily check into source control
- **Security Policy as Code:** define users and hosts, organize into groups and layers, provide logical roles to control access to resources (secrets)
- Define structure of authorization
- Policy Markup format (YAML-like) to provide easy to generate and manipulate methods
- Supports any programming language and automation friendly
- Safely re-apply policy any number of times
- Policy is loaded using the Conjur CLI

```
- !user
  id: bob
  annotations:
    name: Bob Green
    email: bgreen@example.com
    dept: Org-2
```

```
$ conjur policy load root root-policy-file-name.yml
```



Note:

Policy files do not contain secret or password values. They contain only the declaration of the variable that will store the values and rules defining which entities can access the values.

CONJUR CLI

Conjur CLI

- Powerful, robust command-line tools pre-packaged to support several platforms
- Open Source and hosted via GitHub (<https://github.com/cyberark/conjur-cli>)
- Ruby Gem & Docker Container image installation available if platform not supported
- Easy installation onto any system requiring administration access to Conjur
- Administration support for users, hosts, roles, resources, policies, and secrets
- Use --help option to display command help

Common Usage

- Create & manage roles, resources, and policies
- Users & Groups administration
- Listing & searching object values
- Secrets management

```
$ conjur --help
NAME
  conjur - Command-line toolkit for managing roles, resources and privileges

SYNOPSIS
  conjur [global options] command [command options] [arguments...]

VERSION
  6.2.0

GLOBAL OPTIONS
  --help      - Show this message
  --version   - Display the program version

COMMANDS
  authn      - Login and logout
  check      - Check for a privilege on a resource
  env        - Use values of Conjur variables in local context
  help       - Shows a list of commands or help for one command
  host       - Manage hosts
  hostfactory - Manage host factories
  init       - Initialize the Conjur configuration
  ldap-sync  - LDAP sync management commands
  list       - List objects
  plugin     - Manage plugins
  policy     - Manage policies
  pubkeys    - Public keys service operations
  resource   - Manage resources
  role       - Manage roles
  show       - Show an object
  user       - Manage users
  variable   - Manage variables
```

CLI BASIC COMMANDS

| | |
|------------------------------------|--|
| <code>conjur init</code> | Initialize the CLI with the Conjur server |
| <code>conjur authn</code> | Authentication methods for CLI command-line access |
| <code>login / logout</code> | authenticate with a specific username / password & log off |
| <code>whoami</code> | verify username actively authenticated to the CLI |
| <code>conjur user / host</code> | Manage users and hosts |
| <code>rotate_api_key</code> | change/rotate a user or host API Key |
| <code>update_password</code> | change/set a user password (Users only) |
| <code>conjur list</code> | List resource object information |
| <code>--kind=arg</code> | display the summary information for a resource type (i.e. user, host, variable, etc) |
| <code>conjur policy</code> | Manage policy configuration |
| <code>load</code> | load a policy configuration via YAML file |
| <code>conjur role</code> | Manage roles |
| <code>members / memberships</code> | list members of a role & list a roles memberships |
| <code>conjur variable</code> | Manage secrets ("variables") |
| <code>value</code> | get a secret value |
| <code>values</code> | set/change a secret value or access variable values |

MODELING ROLE-BASED ACCESS CONTROL (RBAC)

Define Users & Groups

user.yml

```
---
- !user
  id: dba01
  annotations:
    name: Database Admin 01
    email: dba01@cyberark.com
  ...
```

group.yml

```
---
- !group
  id: dba
  annotations:
    description: Database Admins
  members: [ !user dba01 ]
  ...
```

Define Hosts & Layers

host.yml

```
---
- !host
  id: db-server01
  annotations:
    platform: Database Servers
  ...
```

layer.yml

```
---
- !layer db-servers
  annotations:
    description: Database Servers
  - !grant
    role: !layer db-servers
    member : !host db-server01
  ...
```

Define Variables

variable.yml

```
---
- !variable
  id: app01/secret
  annotations:
    description: DB Instance Secret
- !permit
  resource: !variable app01/secret
  privileges: [ read, write, execute ]
  role: !group dba
- !permit
  resource: !variable app01/secret
  privileges: [ read, execute ]
  role: !layer db-servers
  ...
```

Define Policy

policy.yml

```
---
- !policy
  id: db
  annotations:
    description: DB Servers Model
  body:
    - !policy
      id: mysql
      annotations:
        engine: MySQL Instances
  ...
```



USE POLICY TO TIE IT TOGETHER:

- Declarative and Descriptive
- *Security Policy as Code*

POLICY ELEMENTS

Annotations – free form name:value pair to custom add descriptors & metadata

Permissions – role-based access control (RBAC) defining 3 parts (role, privilege, resource)









Roles – defines the objects that receive permissions (users, groups, hosts, and layers)

- Declare role in policy using “tags” (**!user**, **!grant**, **!deny**, etc.)
- Roles can be assigned permissions using **!permit**
- Roles can be granted to each other using **!grant**

Policies – organize objects, annotations, permissions, and roles

- Isolate permissions of different parts within Conjur
- Declare using **!policy**

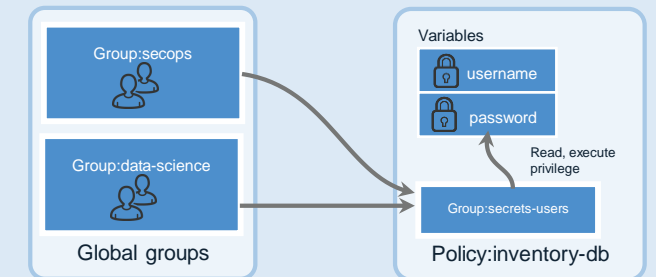
PERMISSIONS

| Role | Privileges | Resource |
|---|---------------|---|
|  dev/srv/nix | execute, read |  dev/db/mysql/cars/secret |
|  dev/srv/nix | execute, read |  dev/db/mysql/cars/username |
|  dev/srv/nix | execute, read |  Vault01A/ConjurSync/Test-DB-MySQL/cardbapp/password |
|  dev/srv/nix | execute, read |  Vault01A/ConjurSync/Test-DB-MySQL/cardbapp/username |

ANNOTATIONS

```
# policy.yml
---
- !variable
  id: password
  annotations:
    mime_type: text/plain
    my_annotation: a value
```

ROLES



POLICY: OWNERSHIP & ENTITLEMENT

Who Can Load Policy?

Policy statements are loaded into either the Conjur root policy or a branch policy. The loading role must have appropriate privileges on the policy. Appropriate privileges are create for creating new objects and update for changing existing policy. Privileges are assigned in permit statements for specific roles. A role can be a policy, a user, a group, a host, or a layer

- Default admin user account
- root policy admin role
- Policy branch owner
- Other roles with create / update privileges

Create an Admin Group

Recommend creating a GROUP role as the root policy owner

- ✓ Members can load policy under root.
- ✓ Owns all policy unless an explicit owner role is named

```
- !group conjur_admin
  owner: !user admin
  annotations:
    Department: Conjur Security Admins

- !permit
  role: !group conjur_admin
  privileges: [ read, create, update ]
  resource: !policy root
```



The default admin account is the owner of the root policy by default

POLICY: LOAD POLICY

- Policies are loaded via Conjur command-line (CLI)
- Create policy in YAML file format and load using the **conjur policy load** command
- Source control is recommended for ease of management, audit, and control
- Using the **owner: <role>** declaration inside the YAML policy file allows specific ownership designation
- Choose the branch policy to load into by declaring as an argument before the YAML file is declared:

conjur policy load <branch> <YAML_file>

Example: Root Policy

```
---
# Create Conjur Security Admin Group
- !group
  id: conjur_admin
  owner: !user admin
  annotations:
    Department: Conjur Security Admins
- !permit
  role: !group conjur_admin
  privileges: [ read, create, update ]
  resources:
    - !policy root

# Define Sub-Level Branches (Dev, Test, Prod)
- !policy
  id: dev
  owner: !group conjur_admin
- !policy
  id: test
  owner: !group conjur_admin
- !policy
  id: prod
  owner: !group conjur_admin
...
```

POLICY: LOAD POLICY

- Command-line help can be viewed by issuing a **--help** optional flag
- Policy can be loaded and reloaded multiple times. Only updates will be applied if loading the same YAML file into the same branch
- Use the **--delete** or **--replace** optional flags to completely remove policy or remove and replace policy
- Use the declarative **!delete** when completing removing a resource from policy
- A numeric counter is tracked for each time a policy is loaded into each branch

```
$ conjur policy load --help
NAME
  load - Load a policy

SYNOPSIS
  conjur [global options] policy load [command options] POLICY FILENAME

COMMAND OPTIONS
  --[no-]delete - Allow explicit deletion statements in the policy.
  --[no-]replace - Fully replace the existing policy, deleting any data that is not declared in the new policy.
```

```
$ conjur policy load root root-policy.yml
Loaded policy 'root'
{
  "created_roles": {
  },
  "version": 1
}

$ conjur policy load dev dev-policy.yml
Loaded policy 'dev'
{
  "created_roles": {
  },
  "version": 1
}
```

```
---

# Delete Users
- !delete
  record: !user testuser01

...
```

USERS & GROUPS MANAGEMENT

USER MANAGEMENT: CONJUR UI CONSOLE

- Conjur UI provides visible access to all of the user accounts
- Use the Dashboard for top-down viewpoints
- Click **Users** menu option or graphical display panel to drill into more detailed information



NOTE:

Conjur UI currently supports view only

The screenshot displays the CyberArk Conjur UI console. The top navigation bar includes the CyberArk logo, a search bar, and user information (admin, Sign Out). The left sidebar contains a menu with options: Dashboard, Policies, Hosts, Layers, Users (highlighted with an orange box), and Groups. The main content area shows a dashboard with six tiles: Groups (8), Users (33, highlighted with an orange box), Layers (1), Hosts (3), Services (2), and Secrets (7). Below the tiles are sections for 'Warnings' (no warnings available) and 'Recent Permission Model Changes' (a table with columns Time, User, and Action). The bottom section is titled 'Users' and contains a search bar and a table with columns ID, Owner, and Integrations.

| ID | Owner | Integrations |
|-----------|------------------|--------------|
| auditor01 | conjur/ldap-sync | |
| auditor02 | conjur/ldap-sync | |

GROUP MANAGEMENT: CONJUR UI CONSOLE

- Conjur UI provides visible access to all of the groups and associated memberships
- Use the Dashboard for top-down viewpoints
- Click **Groups** menu option or graphical display panel to drill into more detailed information



NOTE:

Conjur UI currently supports view only

The screenshot displays the CyberArk Conjur UI console. The top navigation bar includes the CyberArk logo, a search bar, and user information (admin, Sign Out). The left sidebar contains a menu with options: Dashboard, Policies, Hosts, Layers, Users, and Groups. The Groups option is highlighted with an orange box. The main content area shows a dashboard with six summary cards: Groups (8), Users (33), Layers (1), Hosts (3), Services (2), and Secrets (7). The Groups card is also highlighted with an orange box. Below the summary cards, there are sections for 'Warnings' (no warnings available) and 'Recent Permission Model Changes' (a table with columns Time, User, and Action). The bottom section is titled 'Groups' and contains a search bar and a table listing groups.

| ID | Owner | Integrations |
|----------------|------------------|--------------|
| conjur_admin | admin | |
| ConjurAdmins | conjur/ldap-sync | |
| ConjurAuditors | conjur/ldap-sync | |

USER & GROUP ADMINISTRATION OVERVIEW

Creating, Updating, Replacing, and Removing Users & Groups

Methods

- Use **conjur user** CLI command to update password or rotate API Key
 - Creating and updating via CLI have been **deprecated**
 - **conjur user update_password** is supported to change a user's password
 - **conjur user rotate_api_key** is supported to rotate a user's API Key
- Use **conjur policy load** CLI commands to create and update users and groups
 - Use of YAML file format is expected
- Conjur REST API
 - Use of the REST API offers the same functionality for user controls as the CLI
 - Change password and rotate API key

CONJUR CLI: CREATE USERS & GROUPS

- Creating and updating users and groups is performed via Conjur CLI using **conjur policy load** functions
- YAML file format is used to describe the actions to instruct Conjur CLI
- Bulk creation and updates supported for scripting and automation
- Use **conjur list -k <resource_type>** to verify resources created



NOTE:

CLI stdout returns the API Key. Record for accounts not using password credentials.

```
---  
  
# Create User  
- !user dba01  
  
# Create Groups  
- !group dba  
  
# Assign Users to Group  
- !grant  
  role: !group dba  
  member: !user dba01  
  
...  
$ conjur policy load root dba-user_group-create.yml  
Loaded policy 'root'  
{  
  "created_roles": {  
    "CAU:user:dba01": {  
      "id": "CAU:user:dba01",  
      "api_key": "20bva6h1g8enr9qvfg19tw2t6113wh88t364050c1mfqbgr3kmtyh5"  
    }  
  },  
  "version": 2  
}
```

```
$ # Verify User & Group are created  
$ conjur list -k user -s dba01  
[  
  "CAU:user:dba01"  
]  
$ conjur list -k group -s dba  
[  
  "CAU:group:dba"  
]
```

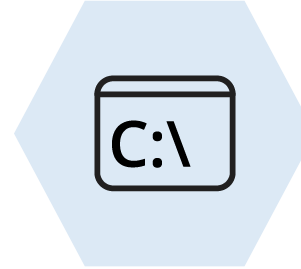
SECRETS RETRIEVAL OVERVIEW

Flexible / Easy-to-Use Interfaces



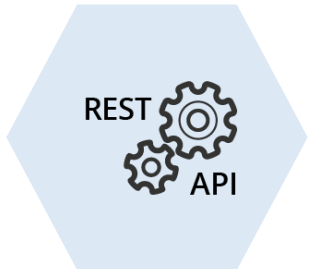
Conjur UI

- Enterprise Web UI
- View/Hide Secrets



Conjur CLI

- Command-Line Tools
- Retrieve Secrets Using Conjur CLI
- Simple / Easy to Use



REST API

- Powerful RESTful API
- Ruby, Java, Python, Node.JS, and .NET C#



Conjur Summon

- Secrets Fetch & Injection
- Declarative Methods
- Invoke w/ Authenticated Identity

CONJUR REST API: AUTHENTICATION

- Conjur REST API requires a temporary authentication access token
- Default expiration is 8 minutes
- Expiration and renewal is automatically handled by CLI and client libraries

Authentication Process:

1. Username and password to obtain user's API key (refresh token)
2. User's API key to obtain temporary authentication access token

Encode User Credential

Linux/Unix Shell

```
echo -n username:password | base64
```

Retrieve API Key

Request

Method: GET

URI: `https://<conjur_server>/authn/<account>/login`

Headers: Authorization: Basic <User_Base64_Credential>

Retrieve Access Token

Request

Method: POST

URI: `https://<conjur_server>/authn/<account>/<login>/authenticate`

Headers: Content-Type: plain/text

Body: <API_Key_String>

CONJUR REST API: CHANGE USER PASSWORD

- Provide User ID and password or API Key using HTTP Basic Authentication in REST header prior for authorization
- Password are ONLY for Users and NOT Hosts; Users have both API Keys and Passwords (if set). Hosts only use API Keys
- Only the User themselves can change their password. The API Key can be rotated/changed by any role with the proper permissions
- Rotate another role's API Key providing proper authentication and permission to control the role being updated; **<kind>** is typically either User or Host

Change User Password

Request

Method: PUT
URI: https://<conjur_server>/authn/<account>/password
Headers: Content-Type: text/plain
Authorization: Basic <User_ID>:<User_Password>
Body: <New_Password>

Rotate Role API Key

Request

Method: PUT
URI: https://<conjur_server>/authn/<account>/api_key?role=<kind>
:<role_id>
Headers: Authorization: Basic <User_ID>:<User_Password>
Response: <New_API_Key>

CONJUR REST API: LIST GROUP MEMBERS

- Conjur REST API requires a temporary authentication access token
- Retrieve members of any Conjur role such as Group members.
- **account** is the organization account name for the Conjur Server
- **kind** is the type of role (user, group, host, layer, or policy)
- **role_id** is the unique Conjur identifier for the specified role to search
- **role_id** must be escaped in REST API URL
- **“/”** becomes **%2F** and **“:”** becomes **%3A**

List Group Members

Request

Method: GET

URI:

https://<conjur_server>/roles/<account>/<kind>/<role_id>?members

Headers: Authentication: Token token="<access_token>"

REST API Escape Syntax

role_id: dev/db/dbadmins

Escaped: dev%2Fdb%2Fdbadmins

SECRETS MANAGEMENT: CONJUR UI CONSOLE

- Secrets are managed using Conjur CLI and/or REST API
- Conjur UI offers role-based access control and visibility to stored secrets
- User credentials and permissions define visibility to secrets
- Secret/Variable values can be updated using the Conjur UI, Conjur CLI, and/or REST API

The screenshot displays the CyberArk Conjur UI console. The top navigation bar includes the CyberArk logo, a search bar, and user information (admin, Sign Out). The left sidebar contains a menu with options: Dashboard, Policies, Hosts, Layers, Users, Groups, Secrets (highlighted with an orange box), and Webservices. The main content area shows a dashboard with six cards: Groups (10), Users (35), Layers (1), Hosts (4), Services (2), and Secrets (13, highlighted with an orange box). Below the cards are sections for Warnings (no warnings available) and Recent Permission Model Changes (a table of authentication events). The bottom section is titled 'Secrets' and contains a table of stored secrets.

| ID | Owner | Integrations |
|---|----------------------------------|--------------|
| conjur/ldap-sync/blind-password/default | conjur/ldap-sync | |
| dev/app/web/app01/secret | dev/app/web | |
| dev/app/web/app01/username | dev/app/web | |
| dev/db/mysql/cars/secret | dev/db/mysql | |
| dev/db/mysql/cars/username | dev/db/mysql | |

CONJUR CLI: CREATE SECRETS

- Use YAML policy statements combined with **conjur policy load** to create secrets
- Creating variables can be very simple or complex depending on policy statements
- Minimum requirements are using **!variable** declaration and defining a unique identifier for each variable
- Load policy into the designated branch to inherit policy settings and use **!permit** to declare and define RBAC controls and permissions



NOTE: Secret variable values **MUST** be set and initialized after creating via policy

```
---  
  
### Create Variables  
- !variable  
  id: cars/secret  
  kind: password  
  annotations:  
    Instance: cars  
    Descriptions: credential  
  
### Assign Variable Permissions  
- !permit  
  resource: !variable cars/secret  
  privileges: [read, execute, update]  
  roles:  
    - !group /dba  
  
...  
$ conjur policy load dev/db/mysql secret-db.yml  
Loaded policy 'dev/db/mysql'  
{  
  "created_roles": {  
  },  
  "version": 1  
}
```


CONJUR CLI: ADD, UPDATE, SHOW SECRETS

- Conjur CLI offers tools to set, update, list, show, retrieve values, and remove secret values
- **conjur variable value** is used to retrieve a secret value
- **conjur variable values** is used to set or update a secret value
- **conjur list** is used to list resources available to the User or Host authenticated to Conjur
- Each secret value update will increment the version and become the new default
- Use command substitution to improve visibility: **echo \$(command)** or **echo `command`**

```
$ conjur variable --help
NAME
    variable - Manage variables

SYNOPSIS
    conjur [global options] variable value [-v arg|--version arg] VARIABLE
    conjur [global options] variable values

COMMANDS
    value - Get a value
    values - Access variable values
```

```
$ # Retrieve Secret Value
$ conjur variable value dev/app/web/app01/secret
Cyberark3$
$ # Change Secret Value
$ conjur variable values add dev/app/web/app01/secret "newpass01"
Value added
$ # Retrieve Secret Value (using command substitution)
$ echo $(conjur variable value dev/app/web/app01/secret)
newpass01
```

```
$ # List a Variable Resource
$ conjur list -k variable -s dev/app/web/app01/secret
[
    "CAU:variable:dev/app/web/app01/secret"
]
```


CONJUR CLI: SHOW VARIABLE DETAILS

- Show the details of a resource such as a variable using the **conjur show** command
- Determine the roles within Conjur that are permitted access to a resource like a variable based on a permission (read, execute, update) using the **conjur resource permitted_roles** command
- **read** permission allows to view the activity for a given resource
- **execute** permission allows to retrieve the resource value
- **update** permission allows to change or update the resource value

```
$ conjur show variable:dev/app/web/app01/secret
{
  "created_at": "2018-10-22T19:26:34.073+00:00",
  "id": "CAU:variable:dev/app/web/app01/secret",
  "owner": "CAU:policy:dev/app/web",
  "policy": "CAU:policy:dev/app/web",
  "permissions": [
    {
      "privilege": "read",
      "role": "CAU:group:dev",
      "policy": "CAU:policy:dev/app/web"
    }
  ],
}
```

... stdout truncated ...

```
$ conjur resource permitted_roles variable:dev/app/web/app01/secret update
[
  "CAU:user:conjuradmin02",
  "CAU:policy:dev/app",
  "CAU:group:dev",
  "CAU:user:conjuradmin01",
  "CAU:user:developer01",
  "CAU:policy:dev",
  "CAU:policy:dev/app/web",
  "CAU:user:admin",
  "CAU:group:conjur_admin"
]
```

CONJUR UI: DISPLAY SECRETS

- Authenticate to Conjur UI using a credential with proper permissions to access secrets
- Click **Secrets** on sidebar navigation or Dashboard
- View secrets details by clicking on a specific secret

The screenshot displays the Conjur UI interface. The top section, titled 'Secrets', contains a search bar and a table of secrets. The table has three columns: ID, Owner, and Integrations. One row is highlighted with an orange border. Below the table, the details for the selected secret are shown.

| ID | Owner | Integrations |
|--|----------------------------------|--------------|
| P conjur/ldap-sync/bind-password/default | conjur/ldap-sync | |
| P dev/app/web/app01/secret | dev/app/web | |
| P dev/app/web/app01/username | dev/app/web | |
| P dev/db/mysql/cars/secret | dev/db/mysql | |
| P dev/db/mysql/cars/username | dev/db/mysql | |

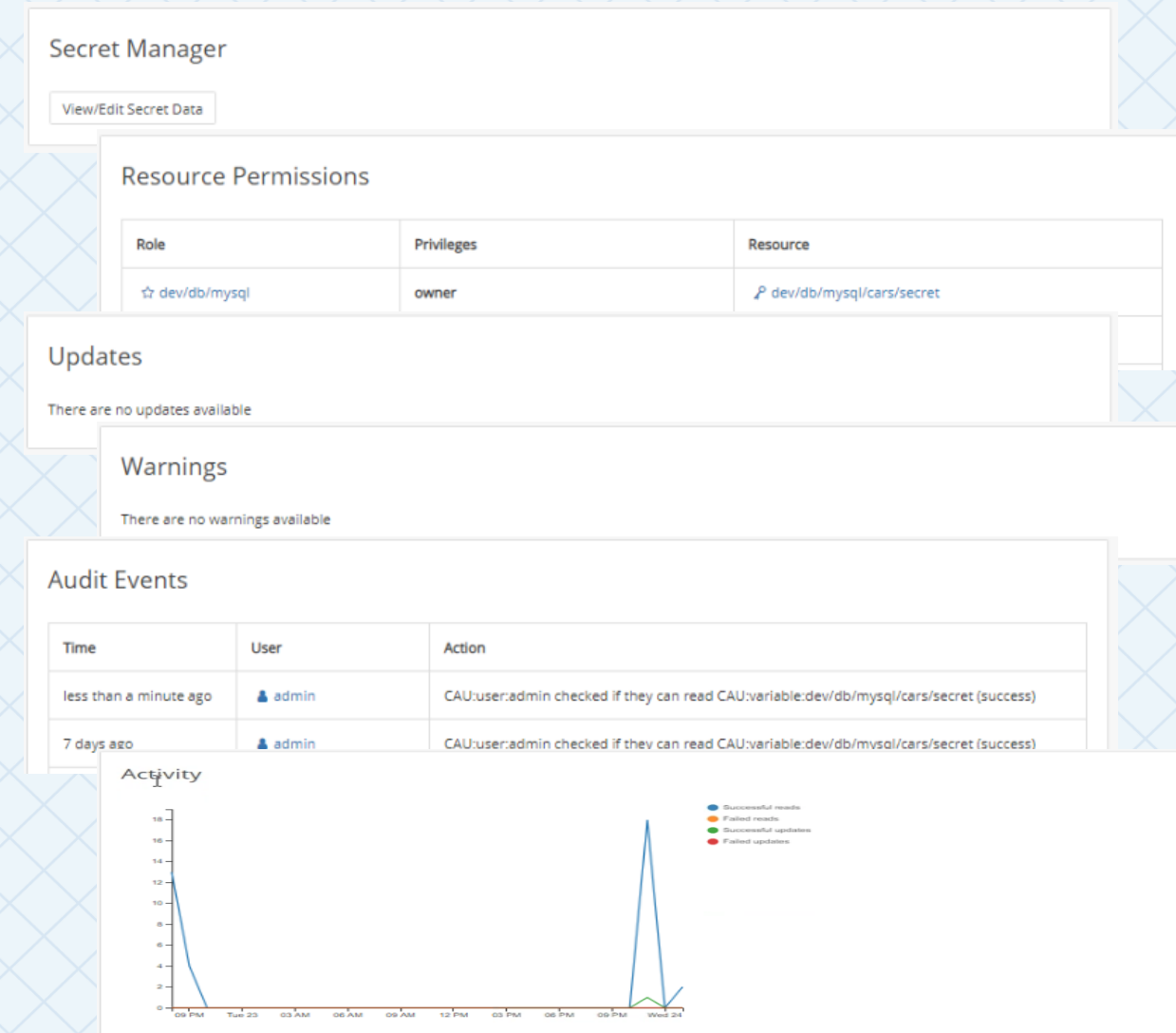
dev/db/mysql/cars/secret Secrets > dev/db/mysql/cars/secret

Details

Owner: [P dev/db/mysql/cars/secret](#)
Created by: [dev/db/mysql](#)
MIME type:
Kind: variable
Versions
Annotations: Descriptions credential
Instance cars
conjur/kind password

CONJUR UI: SECRETS DETAILS

- Gain critical insight for each secret stored in Conjur via the UI Console
- **Secrets Manager** allows access to view/edit secret values
- **Resource Permissions** shows each secrets privileges
- **Updates** shows a granular audit trail of each change
- **Warnings** will provide a health status
- **Audit Events** show a granular audit trail of each transaction
- **Activity** provides a graphical display of usage and activity



CONJUR UI: VIEW & EDIT SECRET VALUE

- Click **Secrets** on sidebar navigation or Dashboard
- View secrets details by clicking on a specific secret
- Click **View/Edit Secret Data** under Secret Manager section to view current secret value
- Change current secret value by editing the value inside the box, then click **Save**
- Click Refresh or use the **conjur variable value** command from the CLI to show updated secret value

Secret Manager

View/Edit Secret Data

Secret Manager

Hide Secret Data

Refresh

Save

Cyberark1

Secret Manager

Hide Secret Data

Refresh

Save

ANewPassword01

```
$echo -e "`conjur variable value prod/linsrv/dns/ext/root`\n"
ANewPassword01
```

CONJUR REST API: CREATE SECRETS

- Requires proper temporary access token authentication prior to processing Conjur API call
- Use `secrets/<account>/<kind>/<var_ID>` REST API to add/update secret values
- **<account>** is the organization account name
- **<kind>** is the type of resource (i.e. variable)
- Secret value to add or update must be **binary** data submitted using the POST method.

Add/Update Secret Value

Request

Method: POST

URI: `https://<conjur_server>/secrets/<account>/<kind>/<var_ID>`

Headers: Authentication: Token token="<access_token>"

Body: Type Binary: <variable_value>

CONJUR REST API: VIEW SECRETS VALUE

- Requires proper temporary access token authentication prior to processing Conjur API call
- Retrieve the value of a secret variable using REST API
- Secret ID must be escaped in REST API URL
- Supports retrieving active/current value and archived versions using the **version** reference option
- Secret current value will be returned if **version** is omitted



NOTE: Conjur retains the last 20 versions of a secret by default. Access in numeric order.

View Secret Value

Request

Method: GET

URI:

https://<conjur_server>/secrets/<account>/<kind>/<id>?<version>

Headers:

Authentication: Token token="<access_token>"

Parameters:

id = name of variable (Example: dev/app/web/app01/secret)

version = version number of secret value to return (default is current)

View Multiple Secret Values

Request

Method: GET

URI:

https://<conjur_server>/secrets?<var_IDs>

Headers:

Authentication: Token token="<access_token>"

Parameters:

vars = List of variables to retrieve

Example:

dev%2Fwebapp%2Fdba01%2Cdev%2Fwebapp%2Fdba02

HOSTS & LAYERS MANAGEMENT

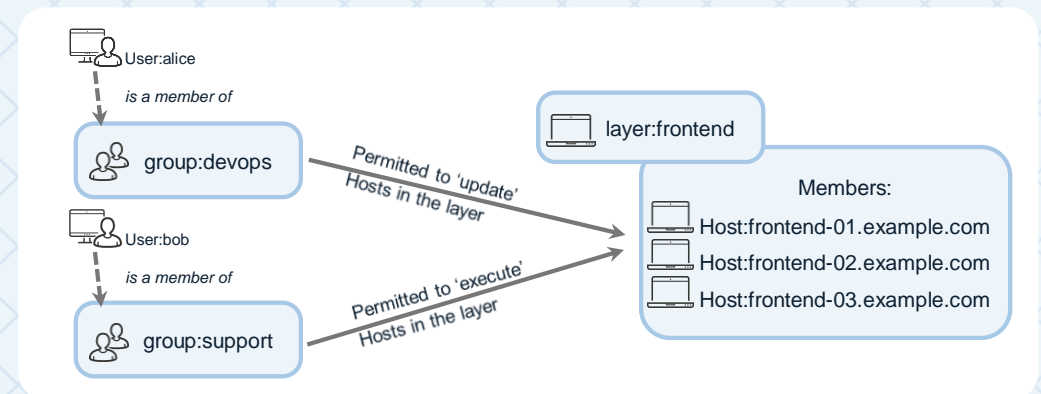
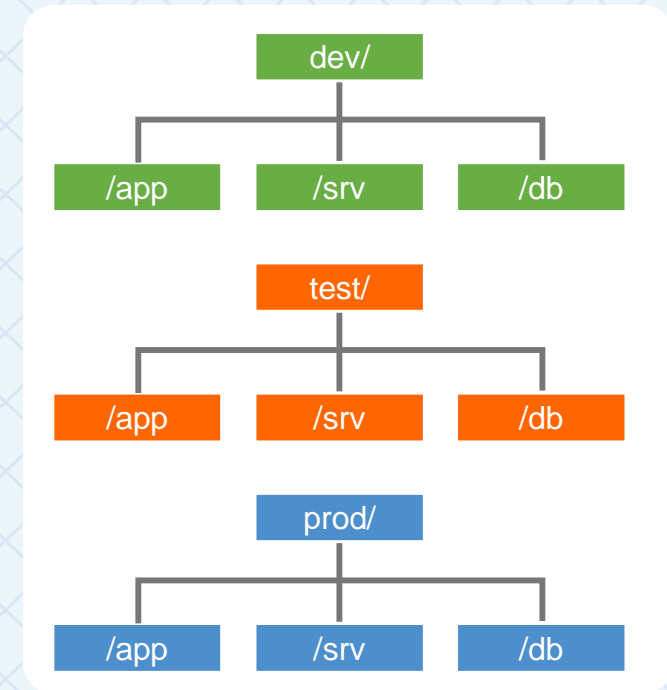
LAYERS OVERVIEW

Conjur Layers

- Manage broad and/or granular privileges and permissions within your enterprise while providing an easy method of organizing systems and infrastructure
- Custom and flexible convention to declare infrastructure hierarchy and logical / functional architecture/design
- Define privileges and permissions for each stage of your enterprise IT environment

Layers Use Cases:

1. Assign Hosts to layers to define privilege to resources (variables)
2. Add Users/Groups to layers to assign membership and automatically grant privileges to hosts



HOST IDENTITY OVERVIEW

Authenticating & Authorizing Non-Human Actors (Hosts)

- Hosts must be authenticated (authn) and authorized (authz) just like Users
- Host Identity is created via YAML policy
- Unique API Keys for each Host Identity
- Permissions & Resource Access is granted through Layers
- Create Layers → Create Host → Grant Host Membership to Layer
- Layers Control Access to Resources

Apply & Store Host Identity Using Files

| File Name | Type | Purpose | Security |
|----------------------|-------------|-------------------------------|------------------|
| /etc/conjur.identity | netrc | Host Identifier & API Key | root only (600) |
| /etc/conjur.conf | yaml | Conjur Server Configuration | root write (644) |
| /etc/conjur.pem | certificate | Conjur Server SSL Certificate | root write (644) |

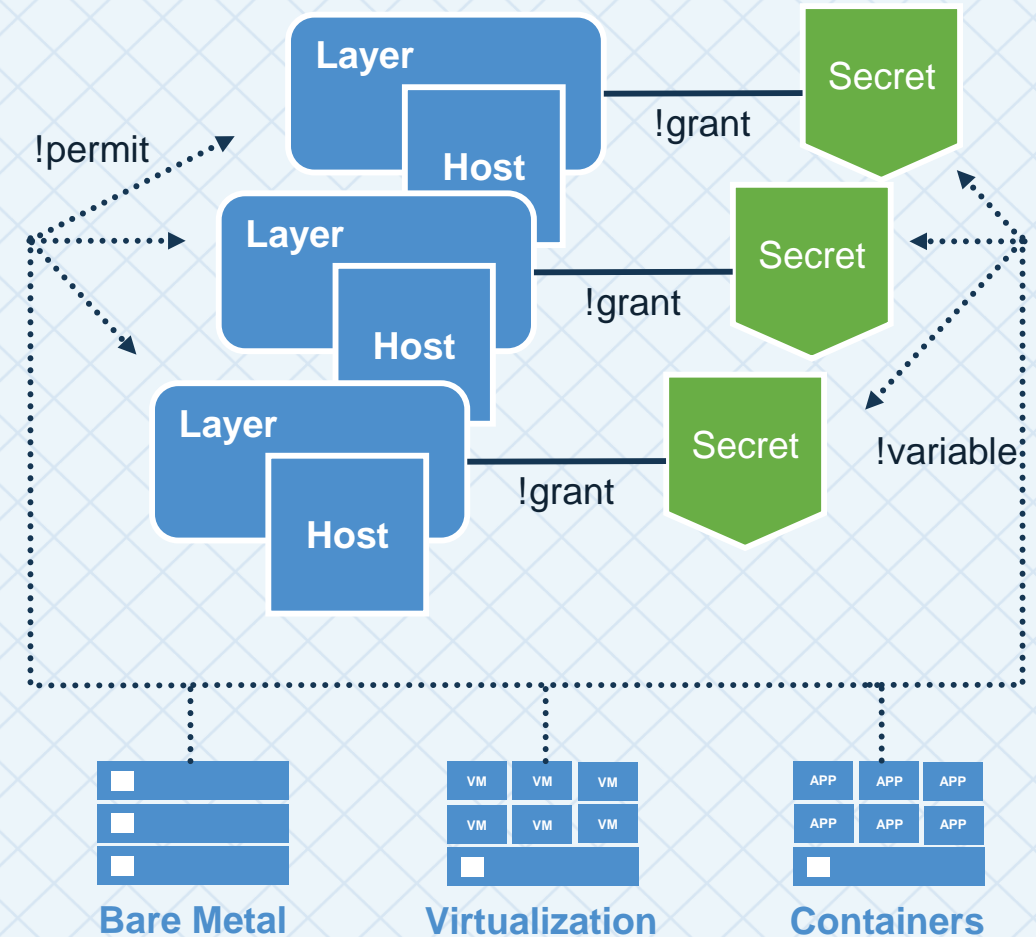


NOTE: Host Identity can also be applied using environment variables. (Useful for applications and tooling that accepts configuration through environment, such as Docker containers or Jenkins jobs)

HOST IDENTITY MANAGEMENT WORKFLOW

Configure Host Identity Workflow

1. Create Layer
2. Create Host
3. Add Host to Layer
4. Create/Update Secret
5. Grant Layer Access
6. Install Host Identity on Server
7. Identify Host Identity Automation



CONJUR CLI: CREATE LAYER & REGISTER HOSTS

- Use YAML policy statements to create layers and register hosts
- Use **!policy** tags to define more complex layer + host associations
- Note and record API keys generated for newly created hosts
- Use Conjur UI, CLI, or REST API to view configuration details
- API Keys can be rotated using Conjur CLI or REST API. Rotate in real-time or automate.
- Use **conjur host rotate_api_key -h <host_ID>** command

```
---
# Create Host
- !host
  id: linsrv01
  annotations:
    Hostname: linsrv01.dev.cyber-ark-demo.local
# Create Layer
- !layer
  annotations:
    Platform: Linux Servers
# Define Layer Membership
- !grant
  role: !layer
  member: !host linsrv01
...
```

```
$ conjur policy load dev/srv/nix host_layer-create.yml
Loaded policy 'dev/srv/nix'
{
  "created_roles": {
    "CAU:host:dev/srv/nix/linsrv01": {
      "id": "CAU:host:dev/srv/nix/linsrv01",
      "api_key": "20y1y72vbczf839h1se52w57tpg2pya6w72j4yby237wxghrep824t"
    }
  },
  "version": 1
}
```

```
$ # Rotate Host API Key
$ conjur host rotate_api_key -h dev/srv/nix/linsrv01
1sy5szb27da3j7226xkwd38djpq9cs3bd2tp0d6e22971cj2p68bse
$ # Rotate Host API Key --> Variable Assignment
$ linsrv01APIKey=$(conjur host rotate_api_key -h dev/srv/nix/linsrv01)
$ echo $linsrv01APIKey
3rbrjtm9eds148k84v3q39c7b1s2kpn02h3mkty5mm59gfkhrvy
```

CONJUR CLI: CREATE SECRET + GRANT LAYER ACCESS

- Establish secret role-based access control with existing secrets and/or newly created
- Use **!permit** to provision the layer associated with the host(s) to obtain secret access privileges
- Define **fetchers (read)** and/or **updaters (execute)** permissions
- Verify host permissions using **conjur resource permitted_roles** CLI command

```
---  
  
# Create Secret  
- !variable  
  id: app01/secret  
  kind: password  
  annotations:  
    Application: Web App01  
  
# Assign Secret Permissions  
- !permit  
  resource: !variable app01/secret  
  privileges: [read, execute, update]  
  roles:  
    - !group /dev  
- !permit  
  resource: !variable app01/secret  
  privileges: [read, execute]  
  roles:  
    - !layer /dev/srv/nix  
  
...
```

```
$ # Load Secret Policy  
$ conjur policy load dev/app/web secret-layer-grant.yml  
Loaded policy 'dev/app/web'  
{  
  "created_roles": {  
  },  
  "version": 1  
}  
$ # Initialize Secret Value  
$ conjur variable values add dev/app/web/app01/secret "Cyberark1"  
Value added
```

```
$ # Verify Host & Layer Permissions  
$ conjur resource permitted_roles variable:dev/app/web/app01/secret execute | egrep "host|layer"  
"CAU: host:dev/srv/nix/linsrv01",  
"CAU: layer:dev/srv/nix",
```

HOST IDENTITY: MANUALLY

Apply Host Identity Manually

1. Create Layer, Create Host, Grant Host Membership, Create Variable, Grant Layer Execute
2. Create, Build, Install Conjur CLI, & Start Host Machine
3. Create & Configure Host Identity Files (conjur.identity, conjur.config, conjur.pem)
4. Set Host Identity File Permissions
`chmod 600 /etc/conjur.identity; chmod 644 /etc/conjur.config /etc/conjur.pem`
5. Edit `/etc/hosts` File → Add Conjur Server IP and FQDN
6. Test & Verify Functionality → Connect to Host using SSH → `conjur authn whoami`

Sample `/etc/conjur.identity`

```
machine https://<conjur_server>/authn
login host/<host_ID>
password <host_api_key>
```

Sample `/etc/conjur.conf`

```
account: <org_name>
appliance_url: https://<conjur_server>/
cert_file: /etc/conjur-<org_name>.pem
netrc_path: /etc/conjur.identity
plugins: []
```

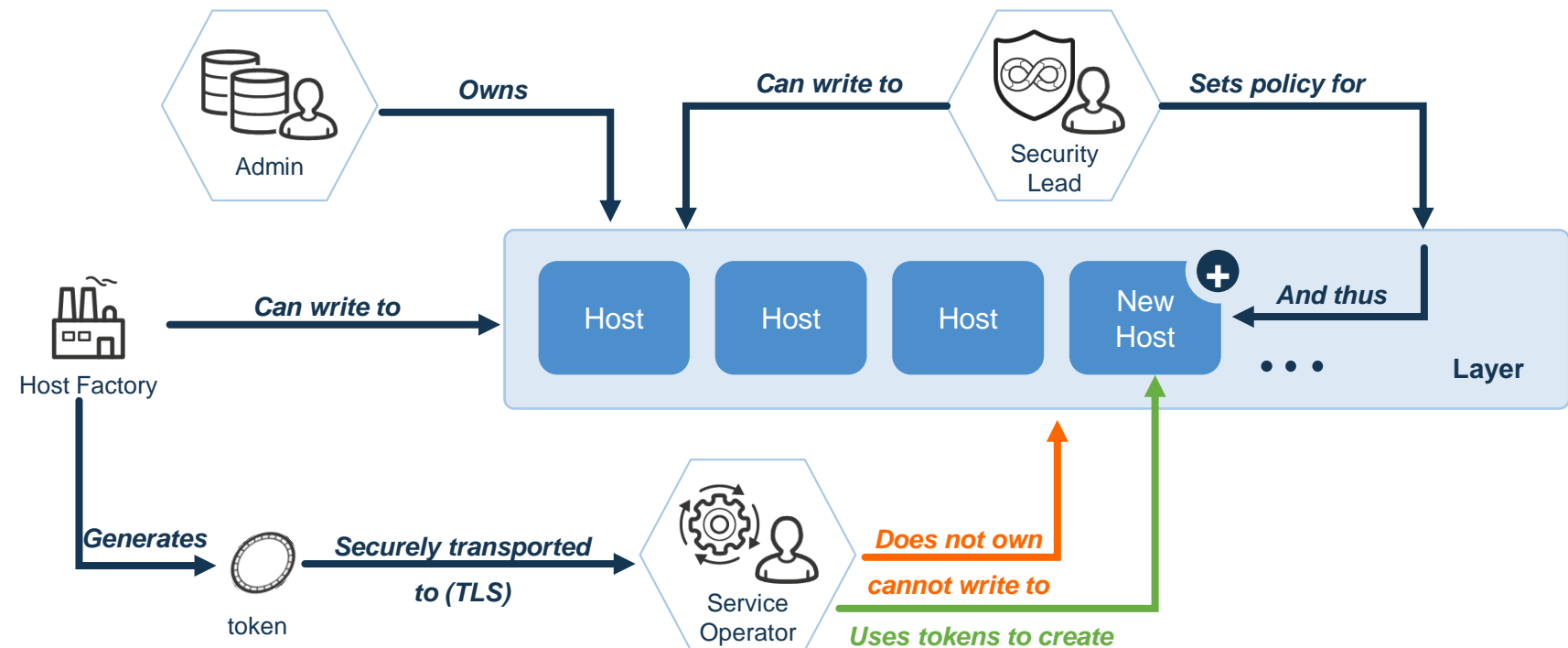
HOST IDENTITY: HOST FACTORY

Easy & Secure Method to Apply Host Identity

- host-factory runs as Conjur Service
- Method to Easily control policy & maintain consistency for Host Identity
- Inherit policy/permissions of Layer membership (Layer includes Host Factory)
- Secure Access using temporary tokens and CIDR address restrictions (optional)
- Supports use of automation tools (Puppet, Chef, AWS CloudFormation)

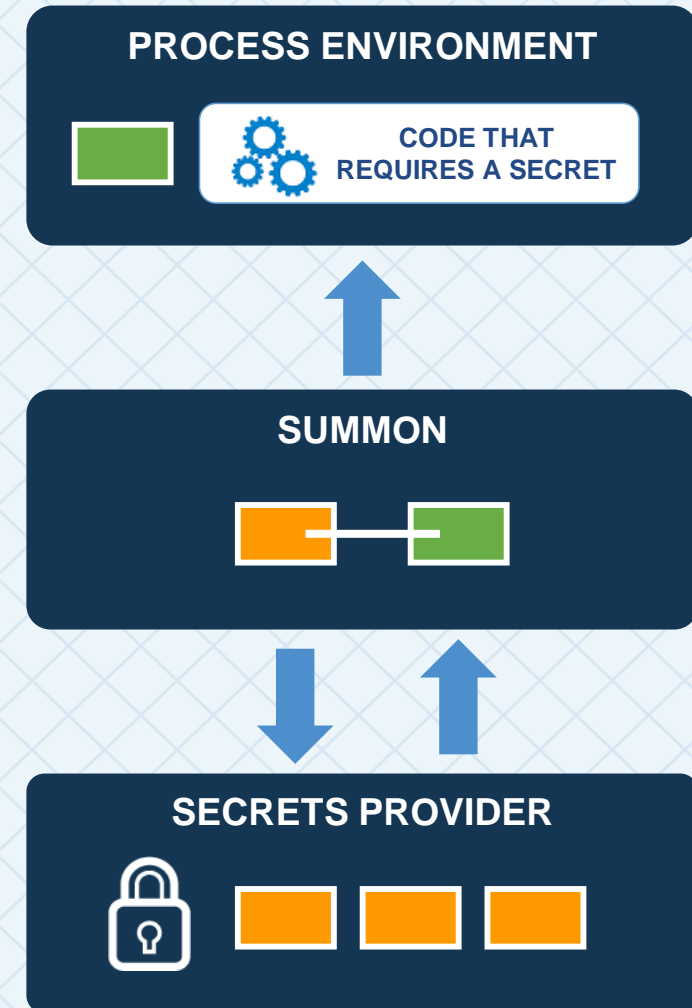
Host Factory Tokens

- Temporary Access Key to uniquely identify new Host
- Authenticate & Validate to Conjur
- Supports Short or Long-lived Tokens
- Supports Encrypted Tokens (example: AWS KMS)



SUMMON

- Summon Overview
 - Open Source tool to help developers get control of secrets while using source control
 - Pluggable provider to retrieve secrets
 - Map environment variable to secrets identifiers using YAML (secrets.yml)
 - No secrets remain after process exits
- Use Cases:
 - Web Applications, Deployment Scripts, Puppet/Chef Runs
- <https://github.com/cyberark/summon>



POLICY BEST PRACTICES

ROOT POLICY BEST PRACTICES

- admin group. The admin group should be the owner of the system policies and environments which are defined later in the root policy.
- Type of Policies:
 - System policies
 - Environments.
 - Application policies.
- Entitlements For example, an entitlement may allow:
 - A user group to manage a policy
 - An application layer to use database secrets
 - A cluster (e.g. k8s) host to join an application layer

ROOT POLICY EXAMPLE

```
# root.yml
- !group admin
- !permit
  role: !group admin
  privileges: [ read, create, update ]
  resource: !policy root
# Users and groups will be managed in this policy by ldap-sync.
# Some of those users and groups may be added to the "admin" group.
- !policy
  id: ldap
# System policies. Used to configure Conjur features such as authenticators.
- !policy
  id: conjur
  owner: !group admin
  body:
    - !policy authn-ldap
    - !policy authn-k8s/gke
    - !policy authn-iam/2884274272
# Environment owner groups
- !group
  id: ci-admin
  owner: !group admin
- !group
  id: us-prod-admin
  owner: !group admin
- !group
  id: us-pci-admin
  owner: !group admin
- !group
  id: eu-prod-admin
  owner: !group admin
- !policy
  id: ci
  owner: !group ci-admin
# Environments
- !policy
  id: us
  owner: !group admin
```

Alternatively, the sub-policies of a region can be defined in us.yml and loaded with: "conjur policy load us us.yml"
However it may be simpler to just create all the environment structure here.

```
body:
  - !policy
    id: prod
    owner: !group us-prod-admin
  - !policy
    id: pci
    owner: !group us-pci-admin
  - !policy
    id: eu
    owner: !group admin
    body:
      - !policy
        id: prod
        owner: !group eu-prod-admin
# etc, other flavors of production policies
```

POLICY BEST PRACTICE 1/2

Policy planning

- Take time to plan your policy approach.
- Enterprise-wide identity policy, such as all users and groups, might be appropriate to maintain under root.
- For most other policy, we recommend creating a branching plan. By isolating policy in branches, you lessen the impact of any change to only that branch. Least privilege is more easily enforced. Policy records contain the namespace name, so troubleshooting and auditing are easier.
- You can organize policy to fit your organization's business needs, development needs, and security operational requirements.

Development versus production policy

- Development and production policy may be organized quite differently because they have different purposes.
- Development teams might own and write their own policy that provides open permits to all hosts (applications, servers, databases, and so on) used in development and testing functions.
- For enterprise production operations, a security team traditionally owns and strictly controls access to policy.

POLICY BEST PRACTICE 2/2

Policy files in source control

- We recommend storing policy files in a source control system. We recommend establishing controlled review and update procedures for changing policy.

Recommended practice is to maintain the following types of policy files:

- A root policy in its own file
- Application policy files that define resources specific to each application. Variables are usually defined in application policies.
- Identity/Entitlement files that define users, groups, hosts, and layers and their associated permissions.

Policy branches

We recommend creating *most* policy in branches off of root. Branches have the following advantages:

- Convenient for organizing policy rules.
- Helps to isolate policy for least privilege assignments.
- Enforces role-based access control with ownership and specific permits.
- Provide DevOps advantages by allowing relevant users to manage their own policy.

ANY QUESTIONS?