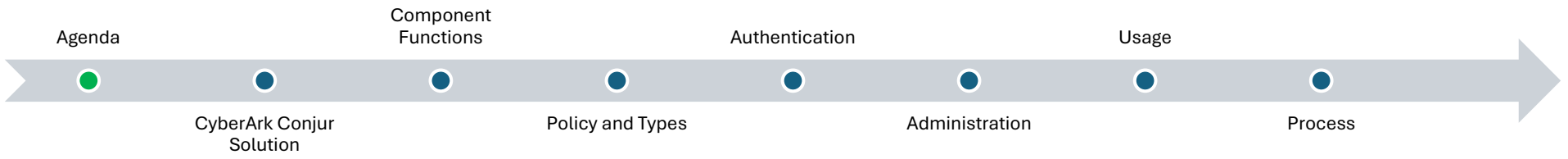
Conjur Secrets Management

Agenda

- CyberArk Conjur Solution
  Solution Overview
  Feature Overview
  Architecture
- Component Functions
- Policy and Types
- Authentication
- Administration
- Usage
- Process

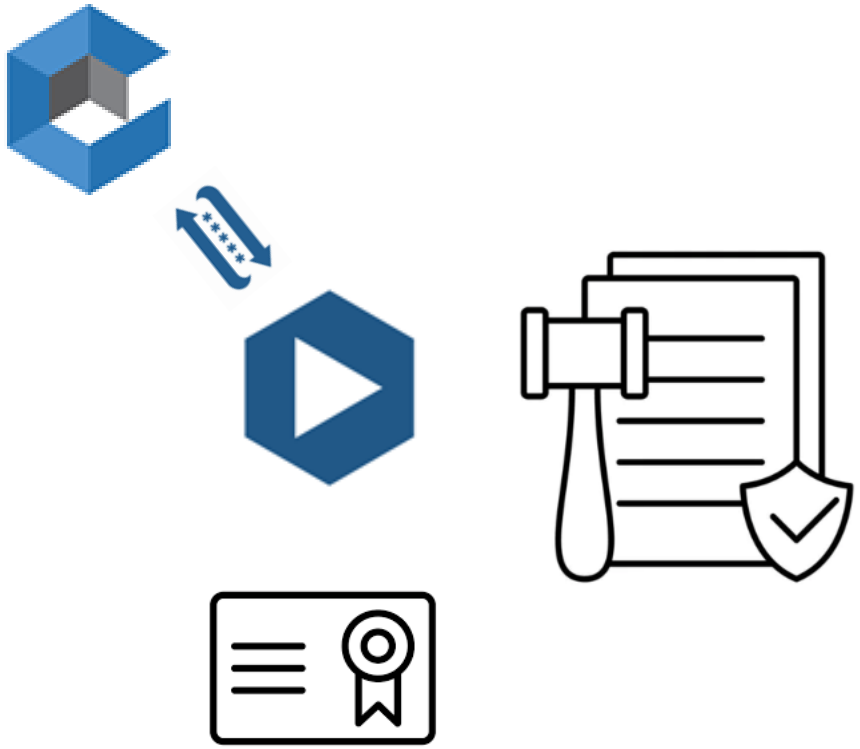
Further Reading:

DANGER: ADVANCED

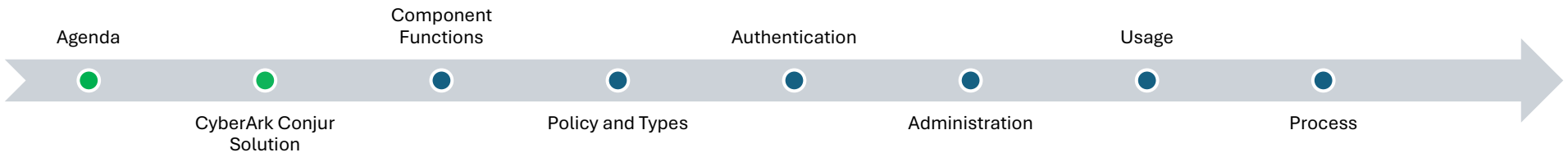


CyberArk Conjur Solution: Feature Overview

- Auditing and Integrity
  Tamper Proof Audit log*.
  SIEM Integration.
  Cloud and on-prem coverage.
- Integration with Tools
  Can Use multiple tools to interact with Conjur: CLI Tool, REST API, Summon, in addition to Java, Go, Ruby, and .NET API's.
- Encryption at rest and in-transit
- Policy as Code
- Role based access control (RBAC)



*NOTE: Logs from Conjur are not sent to CyberArk PAM.



CyberArk Conjur Solution: Architecture

Vault Synchronizer

Retrieves select credentials for CyberArk.

Auto-Failover Cluster

Leader: The primary system that allows policy management and integrates directly with the Vault Synchronizer (and indirectly with the CyberArk Vault).

Sync Standby: A warm system that can take over if the leader fails.

Async (potential) Standby: A warm system that can take over if the leader fails, replicates slightly behind the Sync Standby.

Async Standbys

DR systems that act as manual failover from Auto-Failover Cluster.

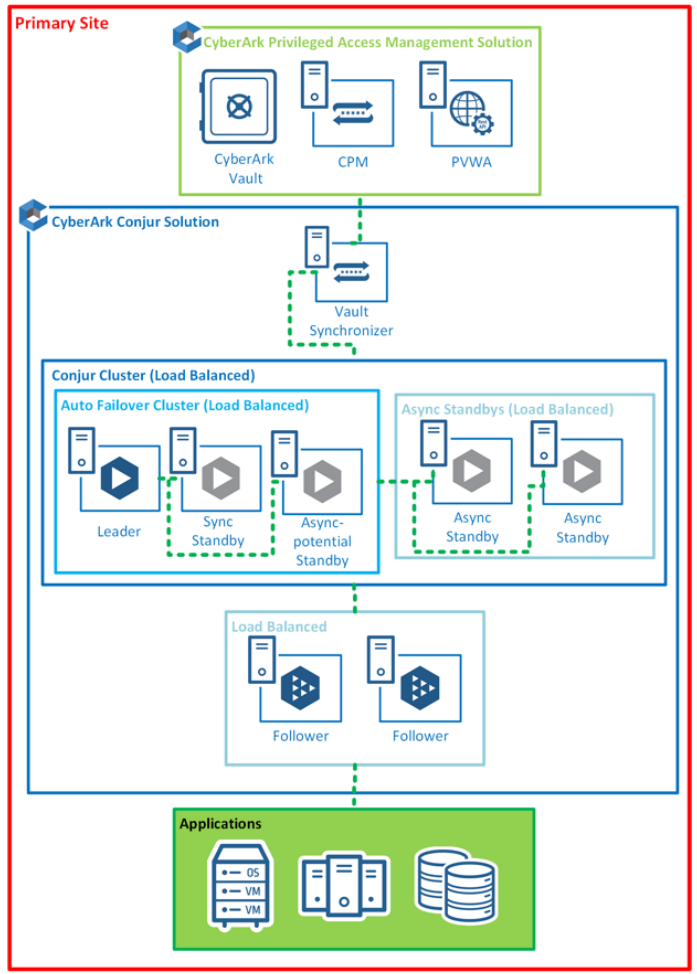
Followers

Read only access, used to retrieve credentials

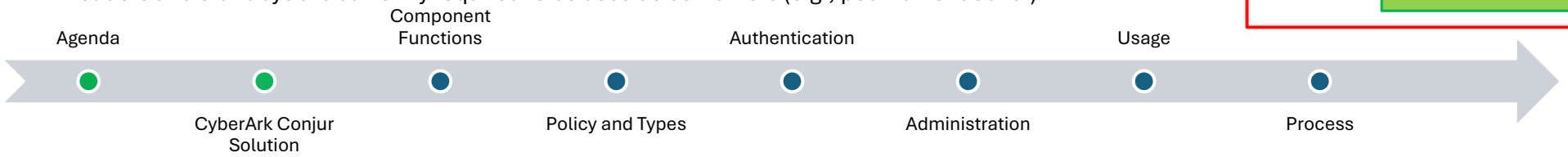
Quick deployment to various datacenters/location (e.g., CCP).

NOTE: All credentials brought into Conjur are replicated to all Conjur systems.

Sample Architecture



NOTE: All leaders and standbys are currently required to be used as containers (e.g., podman or docker).



Component Functions

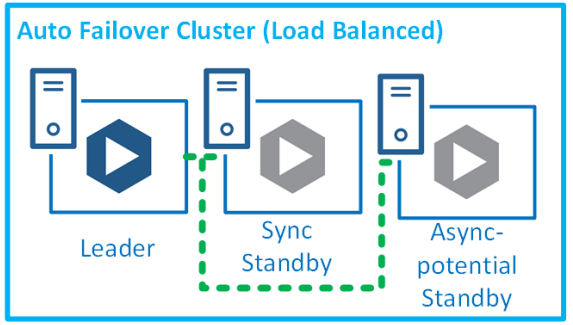
Component Functions: Auto-failover Cluster

Leader

Function: Read/Write Endpoint controlling authentication, authorization, and secrets.

- Can preform read and write functions.
- Retrieves all audit data from Conjur Followers.
- Replicates all data to Conjur Standbys.
- Integrates with CyberArk via Vault Synchronizer.
- Enabled health and info endpoints.
- Can provide secrets.
- Communicates with hosts, applications, and other integrations via webservices.
- Holds time to live value (for failover checking)

From Sample Architecture



Sync Standby

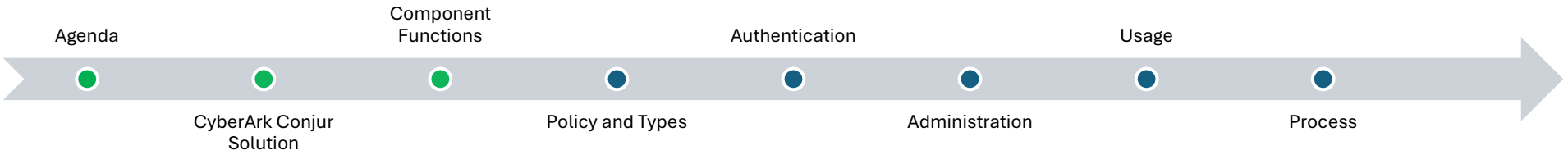
Function: Synchronous backup to Leader

- Intended to take over as leader, if current leader fails (inactive by default).
- Enabled health and info endpoints.

Async (potential) Standby

Function: Asynchronous backup to sync standby

- Intended to take over as leader, if current leader and sync standby fails (inactive by default).
- Enabled health and info endpoints.



Component Functions: Async Standbys and Followers

Async Standby

Function: Asynchronous backup to auto-failover cluster

- Intended to be available for manual fail-over as leader, if auto-failover cluster fails (Async Standby is inactive by default).
- Enabled health and info endpoints.

Async Standbys point to Auto-failover cluster in many configurations.

Followers

Function: Read only endpoint for authentication and secrets retrieval.

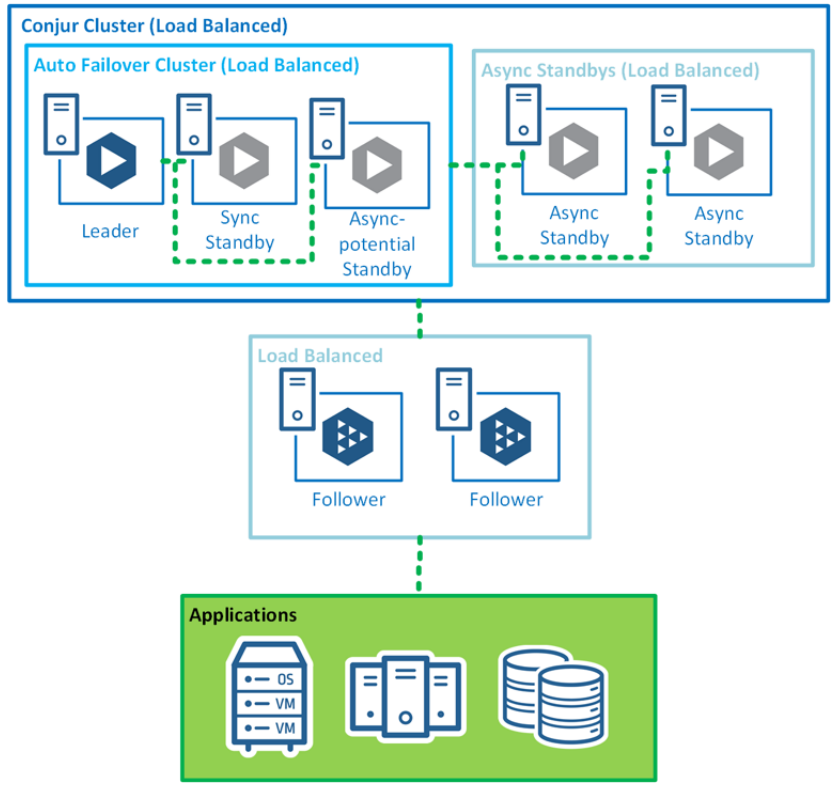
Enabled health and info endpoints (Kubernetes container image-based uses status not health).

Communicates with hosts, applications, and other integrations via webservices.

Default is to deploy two followers per location where apps intended to consume Conjur secrets are located.

Can be deployed to Kubernetes via either 1) Appliance Image or 2) Container Images.

From Sample Architecture



Component Functions: Vault Sync

Vault Synchronizer

Function: Push secrets to Conjur from PVWA/Vault.

Used with LOB Users to sync Vault safes.

Cleanup feature enables accounts to be removed from Conjur when removed from CyberArk Vault.

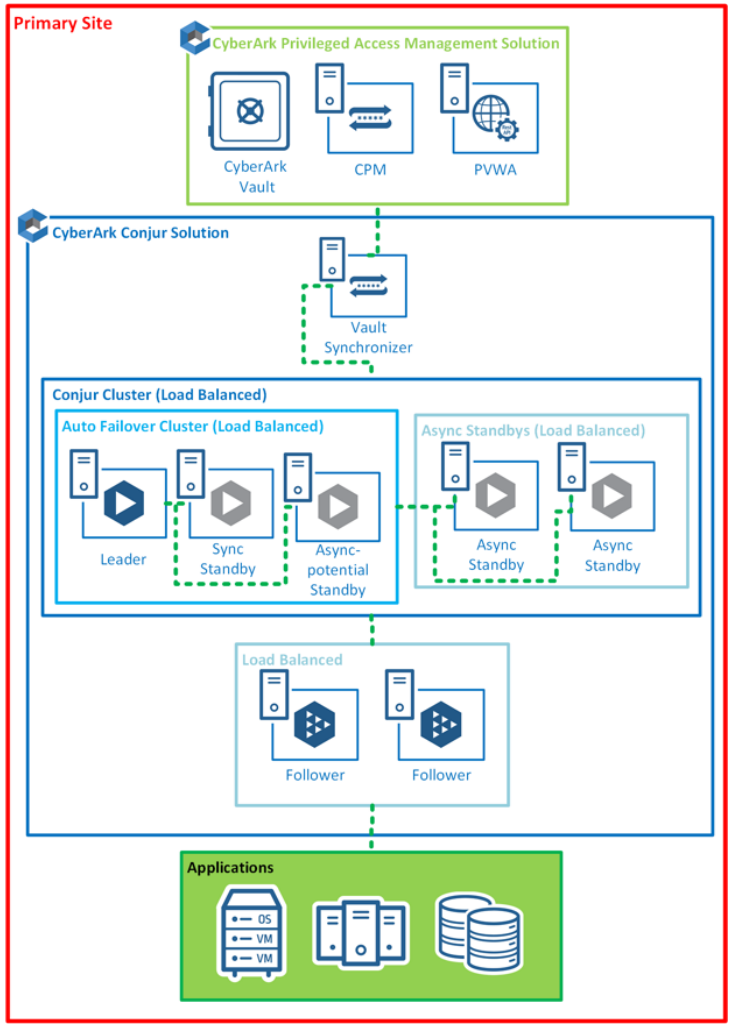
Disabled by default (reference).

Can sync all required and optional platform values for accounts.

Disabled by default only syncs Username and Password (reference).

- Creates Conjur Policy for Vault Admins, Safe groups (Admins, Consumers, and Viewers).
- Limitations may apply for large scale environments (reference).

Sample Architecture



Component Functions: Integrations

HSM

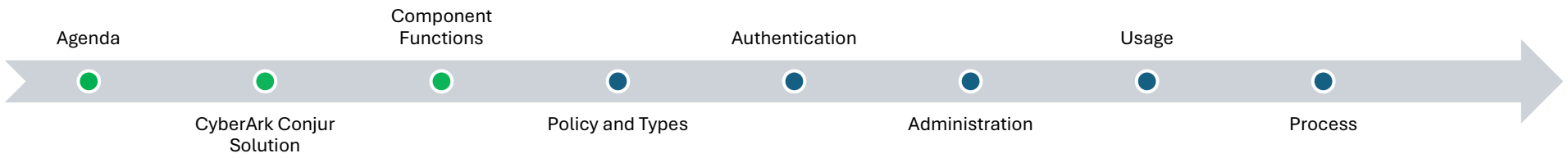
Function: Secures Conjur Encryption Keys with Hardware Security Module.
Reason: Conjur Encryption Keys are stored on the Leader in plaintext.

SIEM

Function: Logs sent to central location.
Reason: Conjur logs will be lost if Conjur systems are unavailable.
Log retrieval from files (e.g. /opt/cyberark/conjur/logs/audit.log or audit.json).
Logs forwarded from followers to currently active leader.

Identity (Users, e.g., LDAP or OIDC)

Function: Allows alternative identity provider to be responsible for authentication.
Reason: Fewer credentials for users.



Policy and Policy Types

Policy and Types: Policy

- Policy

  Function: Maintain Conjur roles, resources, and permissions.

  All Conjur Policy is in YAML (YAML Ain't Markup Language).

  Unique ID defined by policy:

  Planning out structure can limit future overhead.

  Policy is often maintained via CI/CD or related repo*:

  In development/testing branches, other teams may be open to manage their own individual policy.

  In staging/production branches, we should have more stringent policy.

  IMPORTANT: When policy creates users and hosts, this provides an API Key.

  Policy can be loaded in the following modes:

  Load: Default for new policies, crates new data (requires 'create' privilege on policy),

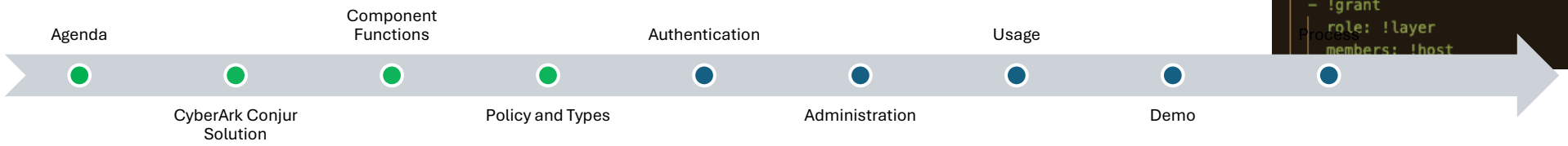
  Update: Will update existing policy, can either create or delete data (requires 'update' privileges on policy).

  Replace: Replaces an entire policy branch (requires 'update' privileges on policy).

Sample Policy

```
- !policy
  id: test
  annotations:
    description: "Testing branch policy."
    editable: "true"
  body:

  - !group
    id: administrators # Creates test/administrators
    annotations:
      description: "Conjur Administrators in test branch."
      editable: "true"

  - !user
    id: conjur-admin  # creates "conjur-admin@test"
    owner: !group administrators
    annotations:
      description: "Conjur Administrator in test branch."
      editable: "true"

  - !variable
    id: var  # creates test/var
    kind: password   # does not have to be defined.

  - !permit
    resource: !variable var
    privileges: [ read, execute ]
    roles !group administrators

  - !grant
    role: !group administrators
    members:
    - !user conjur-admin
    - !user /admin

  - !host

  - !layer

  - !grant
    role: !layer
    members: !host
```

*NOTE: Conjur only maintains the last 20 versions of policy (reference).



Policy and Types: Policy Types (1/2)

- Role (Identity)

  An identity capable of being allocated permissions:

  Examples: User, Group, Host, Layer, and Policy.

  Usage: Users, Groups, Hosts, and Layers can be granted or revoked access to a resource.

  Role Grant Examples

  User → Grant → Group

  Host → Grant → Group

  Host → Grant → Layer

  NOTE: Think of ”grant”, as granting membership.

- Resource (Object)

  Identity that is being secured by Conjur

  The identity of this object is defined in Conjur as [account]:[kind of object]:[id].

  E.g., company:group:conjur-admins

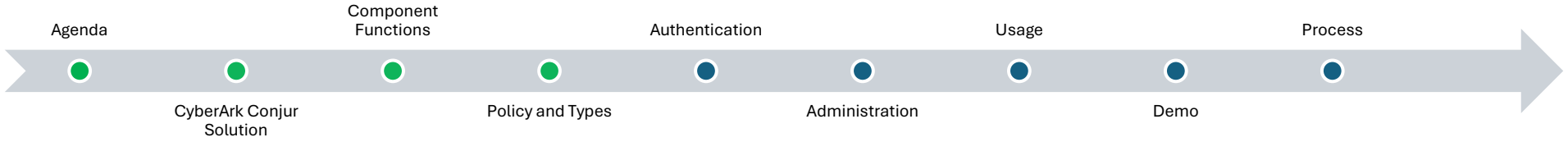
  This object can be allocated permissions

  Objects: User, Group, Host, Layer, **Variable**, **Webservice**, and Policy.

  Usage: A resource is defined, and a role is permitted to access this resource.

  Each resource can have an owner role* (full permissions).

*NOTE: Resources created under the same policy, by default, inherit the ownership value of the role creating the policy.



Policy and Types: Policy Types (2/2)

- Permission

  Permissions are how a role or resource is provided authorization to interact with this role or resource. Permissions currently are as follows on each object:

  Can permit read, execute, and update on **variable** type.
  Can permit read and update on **user**, **host**, **group**, and **layer** types.
  Can permit read, update, and create on **policy** type.
  Can permit read, update, and authenticate on **webservice** type.

  NOTE: Think of ”permit” as allowing a specific type of action. By default, 0 permissions.

- Statements

  Statements are used to update how roles/resources and permissions interact with policy. Statements currently include the following:

  Grant, Revoke, Permit, Deny, and Delete.

Read:
Allows this role to view metadata about the variable, but not the value.

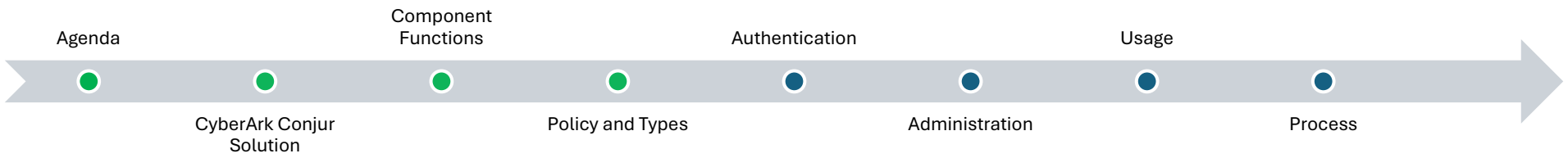
Update:
Allows this role to update the value of variable or update the resource.

Create (policy only):
Allows this role to create objects in this policy.

Execute (variable only):
Allows this role to retrieve

Authenticate (webservice only):
Provides role authentication for this service.

Custom (**DANGER: ADVANCED**)



Policy and Types: Users and Groups

- Users

  Function: Human user login.
  Login via API Key, or Password (optional)*
  Unique ID defined by policy:
  Individual, can only change your password (unless admin).
  If user created under branch other than root,
  login will use <ID>@<branch-path>.

- Groups

  Function: Grouping of users.
  Unique ID defined in policy.
  Collection of identities (normally Users).
  Access to variables and more often provided via group, for users in said group.
  To add users, grant users into groups.

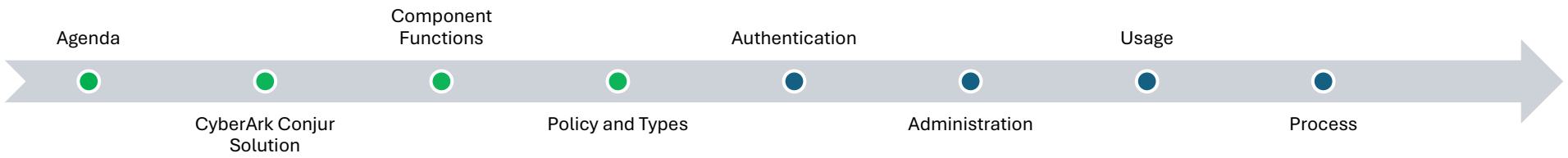
*NOTE: API Keys are retrieved using your login + password combination

Sample Policy

```
- !user
  id: conjur-admin  # creates "conjur-admin@test"
  owner: !group administrators
  annotations:
    description: "Conjur Administrator in test branch."
    editable: "true"
```

Sample Policy

```
- !group
  id: administrators # Creates test/administrators
  annotations:
    description: "Conjur Administrators in test branch."
    editable: "true"
```



Policy and Types: Hosts and Layers

- Hosts

  Function: Application or related identity.
  Login via API Key.
  Unique ID defined by policy:
  - Individual identity.
  - Annotations define additional authentication to webservice.
  - Can add ‘restricted_to’ depending on your LB settings.

- Layers

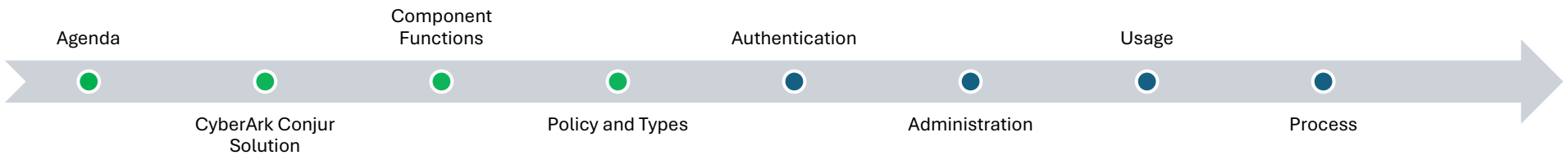
  Function: Grouping of hosts.
  Unique ID defined in policy:
  - Collection of identities (normally hosts).
  - Access to variables often provided via layer, for hosts in said layer.
  - To add hosts, grant hosts membership into a layer.
  - Layers can have Host Factories…

Sample Policy

```
- !host
  id: test
  owner: !user administrators
  annotations:
    editable: "true"
  restricted_to: [10.0.0.0/24]
```

Sample Policy

```
- !layer
  id: test
  owner: !user administrators
  annotations:
    editable: "true"
```



Policy and Types: Webservices

- Webservice

  Function: create/provide a URL endpoint.

  Generally used for either:

  - Authentication to an external service (e.g., JWT Authentication).
  - Creation of Internal Services (e.g., Seed Generation Service – used with K8s Follower).

  Uses ‘groups’ for hosts instead of layers for webservice.

  - Can later add ‘layers’ or ‘groups’ to this group.

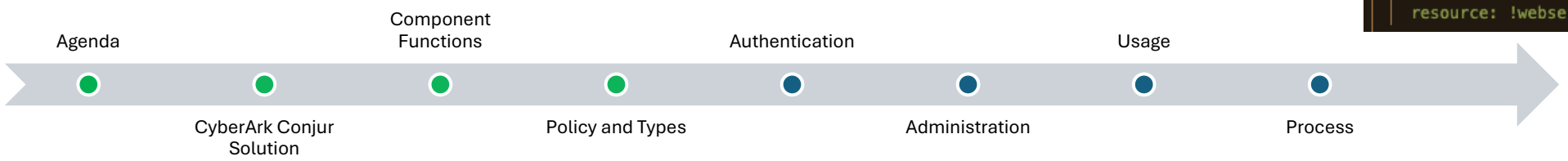
  Example Usages:

  - Authn-jwt: Add hosts/layers to authentication group created with policy.
  - Authn-ldap: Add users/groups to authentication group created with policy.

  NOTE: For user authentications, users must be defined in policy (often at the root level).

Sample Policy

```
- !policy
  id: conjur/authn-jwt/gitlab-demo
  body:

  - !webservice

  - !webservice
    id: status

  - !variable
    id: jwks-uri

  - !variable
    id: token-app-property

  - !variable
    id: identity-path

  - !variable
    id: issuer

  - !group
    id: apps

  - !permit
    role: !group apps
    privilege: [ read, authenticate ]
    resource: !webservice

  - !group
    id: operators

  - !permit
    role: !group operators
    privilege: [ read ]
    resource: !webservice status
```



Policy and Types: Variables

- Variables

  Function: Secret Data Storage.

  Vault Synchronized Conjur (per save):

  Default: Username and Password pulled into Conjur

  All Fields: All fields on password object pulled into Conjur

  Normally defined updated in policy for only webservices.

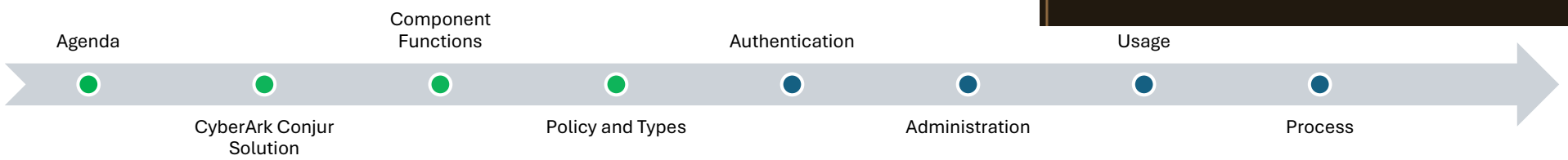
  Variables can be used as configuration information in Conjur.

  Must permit access to variable

  – Inheritance (unless from Vault Sync Group) is not used to provide access to multiple variables.

Sample Policy

```
- !variable
  id: var  # creates test/var
  kind: password   # does not have to be defined.

- !permit
  resource: !variable var
  privileges: [ read, execute ]
  roles !group administrators
```



Policy and Types: Permit and Deny

- Permit

  Function: Provides privileges on a resource.

  Must denote permissions to be added to role (e.g., update or other permission to group or layer).

  NOTE: Requires 'create' permission on the policy to create a permit statement.

  NOTE: Requires 'update' permission on the policy to update a permit.

- Deny

  Function: Removes a permit statement.

  Must denote permission to be removed from role (e.g., group or layer).

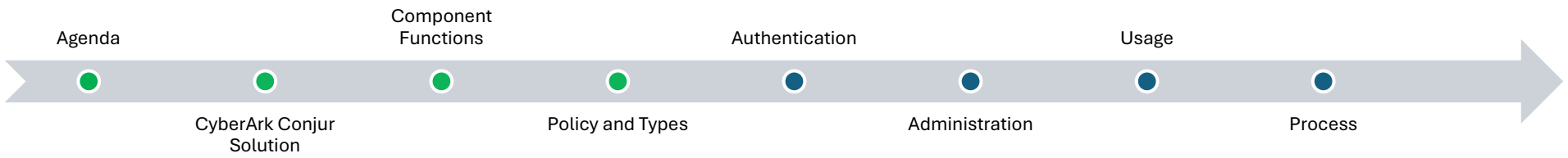
  NOTE: Requires 'update' permission on the policy where the permit was issued.

Sample Policy

```
- !permit
  resource: !variable var
  privileges: [ read, execute
  roles !group administrators
```

Sample Policy

```
- !deny
  resource: !variable var
  privileges: [ read, execute
  roles !group administrators
```



Policy and Types: Grant and Revoke

- Grant

  Function: Grants privileges of a role to another role.

  Can denote roles to be added from role (e.g., group or layer).

  Normally permissions are provided to a role (e.g., group or layer), and inherited on the roles granted (e.g., users and hosts).

  NOTE: Requires 'update' permission on the policy.

- Revoke

  Function: Removes a grant statement.

  Must denote roles to be removed from role (e.g., group or layer).

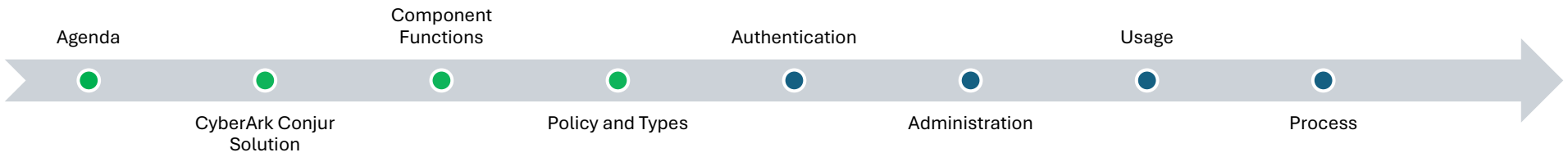
  NOTE: Requires 'update' permission on the policy where the grant was issued.

Sample Policy

```
- !grant
  role: !group administrators
  members:
  - !user conjur-admin
  - !user /admin
```

Sample Policy

```
- !revoke
  role: !group administrators
  members:
  - !user conjur-admin
  - !user /admin
```



Policy and Types (10/11): Delete

- Delete

  Function: Deletes an object from the Conjur database.

  Policy must be loaded in either:

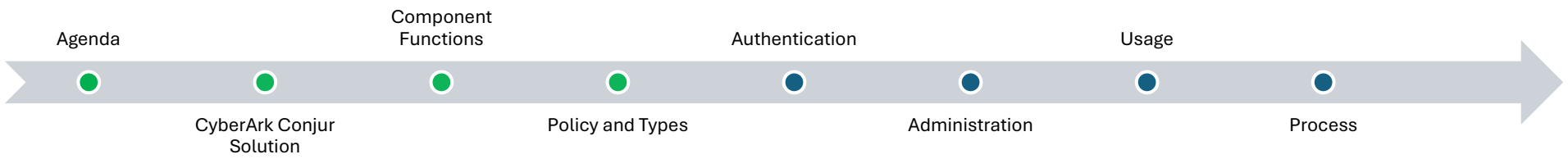
  - Replace Mode.
  - Update Mode.

  Should be used as sperate policy file.

  NOTE: Requires 'update' permission on the policy.

Sample Policy

```
- !delete
  record: !host oss-gitlab-apps-consumers/gitlab-instance-38ba2fc9
```



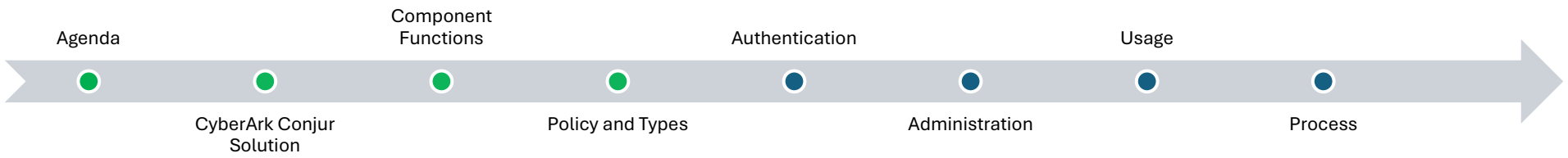
Policy and Types: Host Factory

- Host Factory (reference)

  Function: The Host Factory creates hosts, intended to be short lived.
  Uses a 'Host Factory token' to create hosts.
  Hosts created from short lived Host Factory tokens.
  Each new host is granted membership to the layer of the Host Factory.
  Requires 'execute' permissions on the Host Factory.
  Tokens can be revoked with 'update' permissions.
  Example Usage: Ansible.
  One host to create other hosts for secrets retrieval. Permissions are inherited from the layer that the hostfactory is used in.

Sample Policy

```
- !layer
  id: test
  owner: !user administrators
  annotations:
    editable: "true"

- !host-factory
  id: test
  annotations:
    description: "Host Factory for test VMs."
    editable: "true"
  layers: [ !layer test ]
```



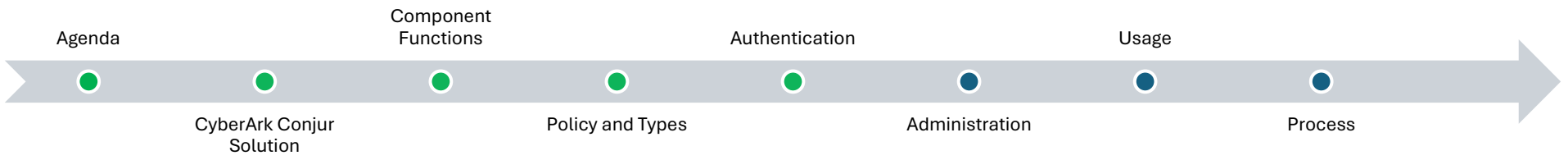
Authentication, Administration, Usage, and Process (+ Lessons Learned)

Authentication: Users

- Users

  Function: Authentication of users required to login to Conjur.
  Authentication methods include authn (basic authentication),
  authn-ldap, authn-oidc, etc.

  NOTE: Limitations apply to OIDC and LDAP.



Authentication : Hosts

- Hosts

  Function: Often application or system integration with Conjur.

  Purpose focused authenticators:

  Specific Authenticators (e.g., authn-jwt, authn-k8s, authn-iam, authn-azure, authn-gcp).

  Initial secret required

  Summon + Summon-Conjur (Requires manually building configuration files).

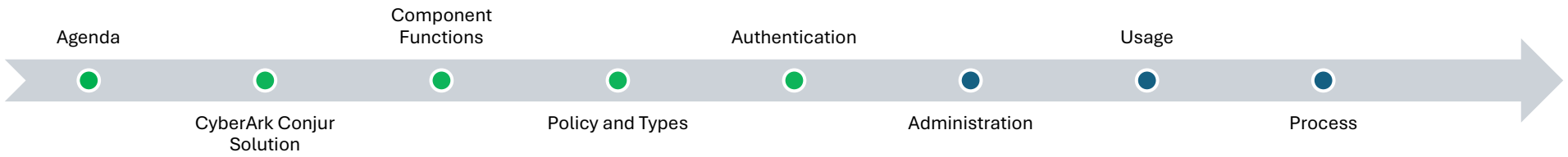
  REST API: Normally requires recode or additional wrapper script

  Language Specific Library (Java, Go, Ruby, and .NET API's): Normally requires recode.

- JWT

  Many applications leverage JWT.

- Kubernetes (k8s)
  - Certificate Authentication: Namespace, Deployment, Service Account, etc.
  - JWT Authentication: Namespace, Service Account, etc.
- AWS IAM
  - AWS IAM Role ARN for STS, EC2, and Lambda.
- Azure
  - Azure IMDS JWT Auth, for VMs, App Services, Functions, and Container Instances.
- OIDC
- GCP
  - JWT Authentication for GCE and GCF.



Administration: Policy Specific

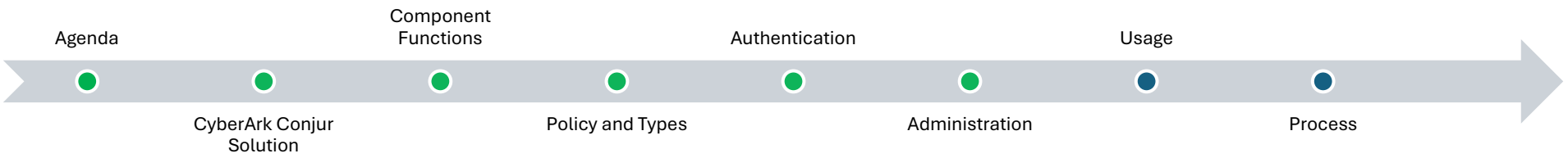
- Conjur Policy Maintenance
  - CI/CD Policy Storage, Versioning, and Deployment
    - Conjur only maintains the latest 20 versions of policy (ref).
    - CI/CD platform often used to maintain root policy and/or policy branches.
  - Policy Maintenance Methods:
    - Conjur CLI (pointing to Conjur Cluster)
    - Conjur REST API calls (pointing to Conjur Cluster)
  - Policy Updates:
    - New Policy can be used via Policy 'load' mode.
    - Updates, New, and removal of Policy and objects can be used via Policy 'update' mode.
    - Using a singular Policy file, to replace/refresh each item can be loaded via policy 'replace' mode.
    - Creation via sample sources like CyberArk Policy Generator and related Github references.

Sample Policy

```
conjur policy update -f remove-old-gitlab.yaml -b root
```

Sample Policy

```
conjur policy load -f authn-oss-gitlab-jwt-hosts-new.yaml -b oss-gitlab-apps-consumers
```



Usage: Administrators, Users and Auditors

- All Users

  Authentication Flow:

  Username and Password to retrieve API Key.
  API Key to retrieve Authentication Token (JWT based).

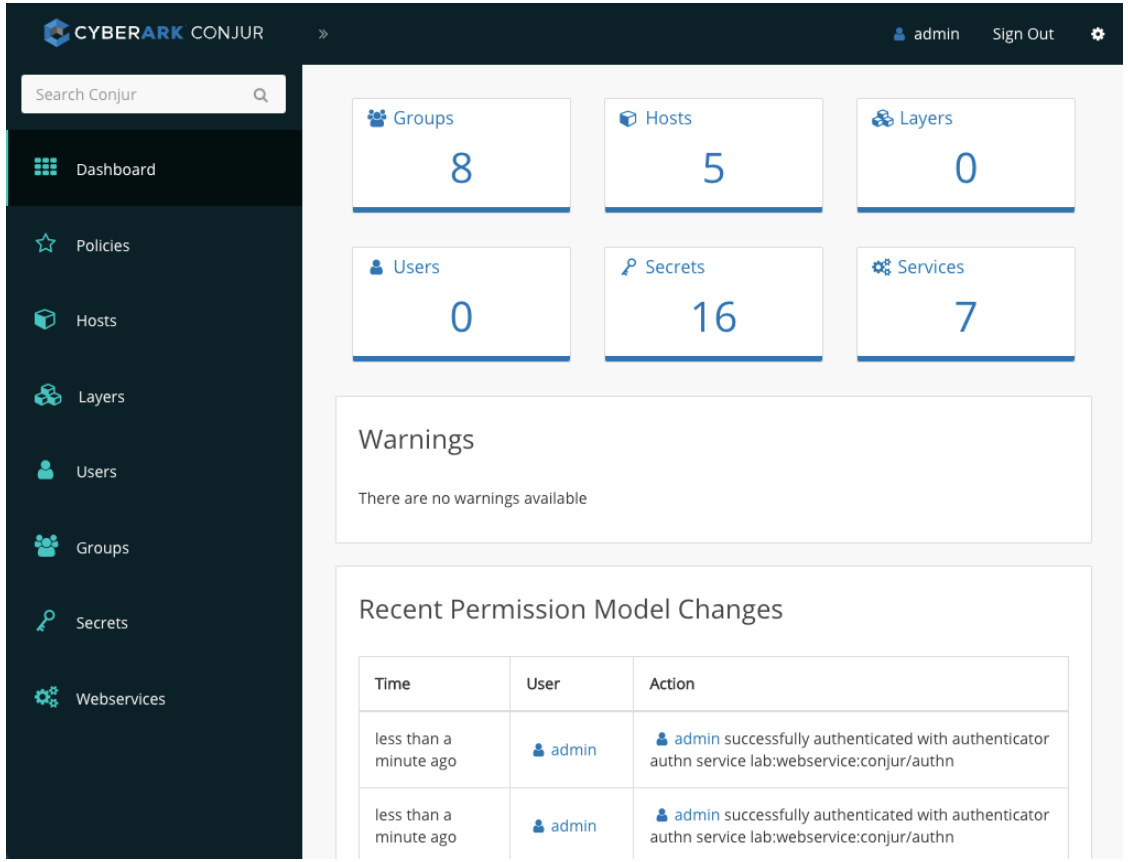
- Administrators

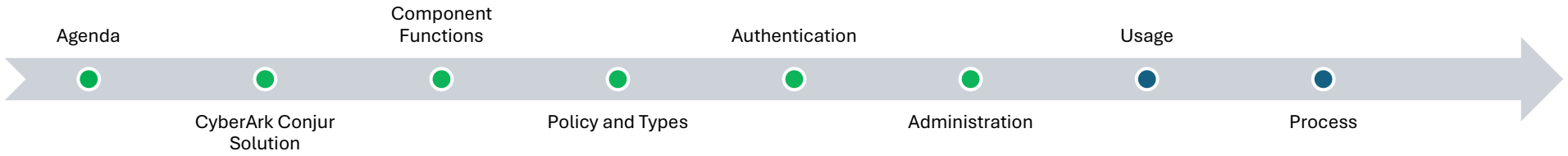
  Maintain Conjur Policy.
  Add, Update, Remove users from Conjur.

- Users

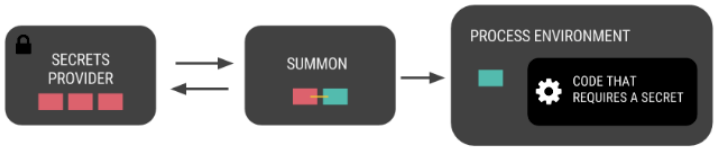
  Review Conjur Policy and minor updates if applicable.

- Auditors
  - Review Conjur Policy.





Usage: Hosts



- Tool Usage
  - Summon and Summon-Conjur
  - REST API
  - Language Specific Library (Java, Go, Ruby, and .NET API's)

- Kubernetes
  - Secrets Integrations
    - Conjur Images: conjur-authn-k8s-client and secrets-provider-for-k8s
    - Secrets Options (e.g., Kubernetes Secrets): Environment Variables, Push to file.
  - Container Settings
    - Init Container
    - Sidecar Container

    NOTE: With Kubernetes, the 'secrets' are often stored as base64 strings.

Sample Policy

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: example-app
  namespace: example
spec:
  replicas: 1
  selector:
    matchLabels:
      role: example
  template:
    metadata:
      labels:
        role: demo
    spec:
      serviceAccountName: example-app-account
      containers:
      - name: authenticator
        image: localrepo/conjur-authn-k8s-client
        imagePullPolicy: IfNotPresent
        envFrom:
          - configMapRef:
              name: conjur-connect-cm
        env:
          - name: CONJUR_AUTHN_LOGIN
            value: "host/data/cd/kubernetes/team1/applications/example-app"
          - name: MY_POD_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: MY_POD_NAMESPACE
            valueFrom:
              fieldRef:
                fieldPath: metadata.namespace
          - name: MY_POD_IP
            valueFrom:
              fieldRef:
                fieldPath: status.podIP
        volumeMounts:
          - mountPath: /run/conjur
            name: conjur-access-token
      - name: example-app
        image: localrepo/example-app:latest
        imagePullPolicy: IfNotPresent
        envFrom:
          - configMapRef:
              name: conjur-connect-cm
        env:
          - name: CONJUR_AUTHN_TOKEN_FILE
            value: /run/conjur/access-token
          - name: CONJUR_USER_OBJECT
            value: data/example-vault1/safe1/secret/username
          - name: CONJUR_PASS_OBJECT
            value: data/example-vault1/safe1/secret/password
        volumeMounts:
          - mountPath: /run/conjur
            name: conjur-access-token
      volumes:
        - name: conjur-access-token
          emptyDir:
            medium: Memory
```

Usage: Dual Accounts (Active/Passive)

- Dual Accounts

  Can allow less potential downtime during password rotation of two accounts.

  Active/Inactive:

  Dual Accounts have virtual username mapped in Conjur as singular account (with singular password).

  Requires two accounts with the same permissions on target system.

Dual Account properties

The Dual Account solution uses two account properties to determine which accounts are valid for use at any given time.

Property	Description
DualAccountStatus	This property flags accounts as **Active** or **Inactive**. Dual accounts pairs will always have one active account and one inactive account.
VirtualUsername	This property identifies two identically provisioned accounts in a dual accounts pair under one virtual username.

NOTE: Reference: Configure and Manage.

How it works

Two accounts with identical privileges are assigned: one active (**A**), one inactive (**B**). There is always an active account, which remains untouched during password rotation. This ensures business continuity, with no delays.

Rotation 1

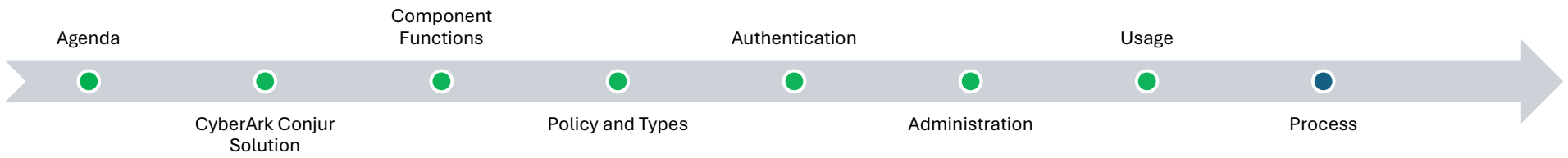
At the set date for password rotation, account **A**, the first account in use, is deactivated, and **B** is activated.

While the second account **B** is active, there is a grace period. At the end of the grace period the password of the deactivated first account **A** is reset. This allows all applications to register the change and switch to using the newly active account.

Rotation 2

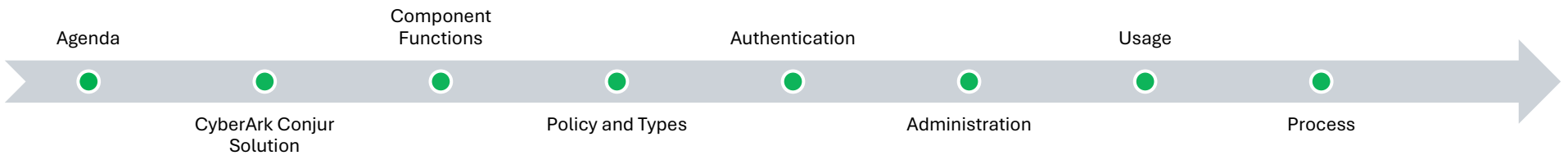
At the next set date for password rotation, account **B** is deactivated. Account **A** is now active.

Deactivated account **B** has its password reset at the end of this grace period.



Process

- Hardcoded Secrets
  Secrets in code, config files, other places it's not managed in.
- Stored Secrets (+ Rotation)
  Secret values stored in CyberArk PAM and integrated to Conjur for retrieval.
- Securely Retrieved via Conjur
  Application Process updated to allow Conjur secrets retrieval as needed.



Lessons Learned

- ✓ Conjur Tools for Secrets Retrieval + DevOps.
- ✓ Conjur Solution Architecture.
- ✓ Conjur Component Functions:
  - ✓ Leader, Sync, Async (potential), Async, Followers, and Vault Synchronizer
- ✓ Overview of Common Conjur Infrastructure Integrations.
- ✓ Conjur Policy Overview with Examples:
  - ✓ Roles: User, Group, Host, Layer, and Policy.
  - ✓ Resources: User, Group, Host, Layer, Variable, Webservice, and Policy.
  - ✓ Permissions: Read, Update, Create, Execute, Authenticate.
  - ✓ Statements: Grant, Revoke, Permit, Deny, Delete, and Host Factory.
  - ✓ Administration: Load, Update, Replace
- ✓ Conjur Hosts/Application Integration Basics.
- ✓ Conjur Hosts and Users Authentication.
- ✓ Conjur Administration Fundamentals.
- ✓ CyberArk Dual Account role in Secrets Retrieval.