Exam Objective 8

Secure Services with Access Control Lists (ACLs)

Secure Services with Access Control Lists (ACLs)

Objective 8a: Set up and configure a basic ACL system

Objective 8b: Create Policies

Objective 8c: Manage token lifecycle: multiple policies, token revoking, ACL roles, service identities

Objective 8d: Perform a CLI request using a token

Objective 8e: Perform an API request using a token

1	2	3	4	5

Difficulty Level

Introduction to the Consul ACL System

Consul ACL system is a built-in but optional feature of Consul

- Controls access to data and the Consul API
- Relies on tokens that are associated with policies which define access

Key components of the ACL system

- Basic
  - Token – a bearer token used during the UI, CLI, or API request
  - Policy – Grouping of rules that determine fine-grained rules to be applied to token
- Advanced
  - Roles – group of a set of policies and service identities applied to many tokens
  - Service Identities – policy template to link a policy to a token or role

Core Components of Consul ACLs

Service Identities

- An ACL policy template that links a policy for services in Service Mesh (connect)
- Helps avoid boilerplate policy creation since similar policies tend to look identical when you have many services registered and using the Service mesh feature
- Helps services/sidecars to be discovered and easy discover other services
- Can be used on tokens and roles
- Applies preconfigured ACL rules

Service Identities are composed of:

- Service – the name of the service (and possibly sidecar proxy)
- Datacenters – a [list] of datacenter(s) that policy is valid for

Core Components of Consul ACLs

Roles

- A named set of policies and service identities
- Sort of a "grouping" of multiple policies & service identities that can be assigned to many tokens

Roles are composed of the following elements:

- ID – auto generated identifier
- Name – unique name within Consul
- Description – human readable description
- Policy Set – a [list] of policies you want to apply to the role
- Service Identities – a [list] of service identities for the role

Enable the Consul ACL System

Consul ACLs must be enabled

- By default, the Consul ACL system is not enabled
- ACLs are enabled in the agent configuration file for Consul servers and clients
- Configuration parameters include default policy and other parameters

Terminal — Consul Agent Config

```
 "acl": {
   "enabled": true,
   "default_policy": "deny",
   "down_policy": "extend-cache",
   "tokens": {
      "agent": "aba7cbe5-879b-999a-07cc-2efd9ac0ffe"
   }
},
```

https://github.com/btkrausen/hashicorp/blob/master/consul/config.hcl

Set Up and Configure a Basic ACL System

Bootstrapping the ACL System

- Required administrative action before ACL system can be used
- Usually done one time during initial configuration
- Default_policy should be set to Allow during bootstrapping and configuration phase

- Bootstrapping creates the bootstrap/master token and the anonymous token
- Creates the Global Management Policy

Enable the Consul ACL System

1. Provision the Consul Cluster
2. Bootstrap the ACL System
3. Create Policies
4. Create Tokens
5. Configure Agents/Clients with Token
6. Update default policy to Deny

What are Policies?

Policies

- Named set of rules that a token is bound by (policies are attached to tokens)
- Policies are reusable – meaning they can be assigned to multiple tokens
- Many policies can be created as needed (depending on different use cases)
- Multiple policies can be attached to a single token (combination of permissions)
- Policies include:
  - ID – public identifier (auto-generated)
  - Name – unique name for the policy
  - Description – (optional) description of the policy
  - Rules – set to rules that grant or deny permissions in Consul
  - Datacenters – the datacenters the policy is valid for
  - Namespace – the namespace the policy resides in (Enterprise)

Default Policies

Global-Management

- Grants unrestricted access to Consul
- Assigned the reserved policy ID of 00000000-0000-0000-0000-000000000001
- Cannot delete or modify this policy
- Can rename the policy if you want
- Assigned to the bootstrap/master token upon ACL system bootstrapping process

Namespace-Management (Ent)

- Policy created on every namespace when the namespace is created
- Somewhat privileged policy for the namespace (can create tokens & new policies)
- Use it, don't use it. Completely up to you
- Can be managed just like any other user-defined policy

Policy Control Levels (Permissions)

allow the resource to be read

READ

WRITE

Allow the resource to be read and modified

Do not allow any permissions to resource

DENY

LIST

Allows access to all keys under a segment in the Consul K/V

ACL Resources Available for Rules

ACL	Operations for Managing the ACL System
AGENT	Utility Operations in the Agent API
EVENT	Listing and Firing Events in the Event API
★ KEY	Key/Value Store Operations
KEYRING	Keyring Operations
NODE ★	Node-Level Catalog Operations
OPERATOR	Cluster-Level Operations
QUERY	Prepared Query Operations
SERVICE ★	Service-Level Catalog Operations
SESSION	Session Operations

★ Commonly Used for Basic Operations

Creating Policies

Rules using <resource> and <resource_prefix>

- Rules written using just the <resource> name must match exactly in Consul

Terminal — Client Policy

```
key "kv/apps/web-app-01" {
  policy = "write"
}
service "customer-db" {
  policy = "read"
```

- Will permit write & read access to the exact path of kv/apps/web-app-01

Will permit to read information about the one service named customer-db

Creating Policies

Rules using <resource> and <resource_prefix>

- Rules written using the <resource_prefix> name can match multiple/all values

Terminal rules.hcl

```
key_prefix "kv/" {
  policy = "read"
}
service_prefix "" {
  policy = "read"
```

- Will permit read access to any path after kv/
  - Example: kv/apps/web01, kv/db/sql01, etc.

Will permit to read information about any service in Consul

Creating Policies

Example Client Policy (eCommerce-Front-End)

Terminal rules.hcl

```
node "web-server-01" {
  policy = "write"
}
key_prefix "kv/apps/eCommerce" {
  policy = "write"
}
session_prefix "" {
  policy = "write"
}
service "eCommerce-Front-End" {
  policy = "write"
}
```

Allow registration of new node

Allow write access to specific data in KV

Initialize a new session

Register/Manage the service named eCommerce-Front-End

Creating Policies

Creating a Policy (CLI)

Use `consul acl policy` command:

- `create` – create a new policy
- `delete` – delete a policy
- `list` – list all policies
- `read` – read the details about a policy
- `update` – update a policy (default merges the old and new rules)

Terminal

```
$ consul acl policy create -name "eCommerce" -description "eCommerce App" -rules @rules.hcl

ID:           06acc965-df4b-5a99-58cb-3250930c6324
Name:         eCommerce
Description:  eCommerce App
Datacenters:
Rules:
service "eCommerce" {
  policy = "write"
}
```

Rules can be provided inline or via file

Creating Policies

Creating a Policy (API)

Create Policies with the Consul API

- Method: PUT
- Endpoint: /acl/policy
- Response: Includes the ID, Name, Description, Rules, etc.

Terminal

```
$ curl -X PUT \
  --header "X-Consul-Token: 45a3bd52-07c7-47a4-52fd-0745e0cfe967"
  --data @payload.json \
  https://consul.example.com:8500/v1/acl/policy
```

Required if ACLs are enabled (assuming Deny default policy)

Terminal

```
$ cat payload.json
{
  "Name": "eCommerce",
  "Description": "eCommerce App",
  "Rules": "service \"eCommerce\" { policy = \"write\"}",
}
```

Create a Policy for the Anonymous Token

Policy for the Anonymous Token

- Anonymous token is used whenever a request is made without a bearer token
- You probably have actions in mind that you want unauthenticated clients to do in Consul
  1. Example might be to query services for IP/hosts
  2. Read a prepared query to determine IP/hosts for a service
  3. Maybe you don't want to provide a token to run a `consul members` command

Terminal rules.hcl

```
service_prefix "" {
  policy = "read"
}
query_prefix "" {
  policy = "read"
}
node_prefix "" {
  policy = "read"
}
```

Create a Policy for Specific Nodes

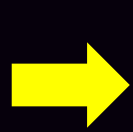
Policies for Specific Consul nodes (server or client)

- Policies can be written so they only apply to that node by name
- Recommended approach to create policies (create one policy per node)

Terminal rules.hcl

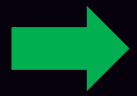
```
node "web-server-01" {
  policy = "write"
}
service_prefix "" {
  policy = "read"
}
```

Create Token

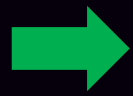


Terminal rules.hcl

```
node "web-server-02" {
  policy = "write"
}
service_prefix "" {
  policy = "read"
}
```



Create Token



Core Components of Consul ACLs

Tokens

- Bearer token used during the UI, CLI, or API request (assuming default deny)
- Tokens are created and attached to a policy
- Used to determine if the request is authorized to perform the request action

Basic components of a token include:

- Accessor – the name or ID of the token
- Secret ID – the actual token used during a request
- Policy Set – the policy or policies attached to the token
- Description – human readable description of the token

Default ACL Tokens

Bootstrap/Master Token

- Bootstrap Token has unrestricted privileges (linked to Global-Management Policy)
- Initial method of authentication for ACL configuration after bootstrapping
- Bootstrap token should NOT be used on a day-to-day basis and securely protected
- If the bootstrap token is lost, there is a "reset" to recreate one
- SecretID will be unique

Anonymous Token

- Used when a request is made that does not specify a bearer token
- Cannot delete the token but you can update the description and privileges
- Commonly set to read services (DNS/API) for unauthenticated clients (possibly more)
- ID is always 00000000-0000-0000-0000-000000000002

Additional Token Attributes

Expiration Time

- Optional configuration that sets the time when the token is revoked and will no longer be valid
- Set as a duration of time (i.e., 30m, 24h, 3d)

Roles

- When tokens are created, they can be assigned a pre-configure role(s) that will be used for the token
- Can specify the role name or the role ID

Service Identities

- Tokens can also be assigned to one or many service identities during creation

Tokens Needed for Production

Example of Required Tokens

- Consul Server Nodes – nodes need to communicate within the cluster
- Consul Clients – need access to specific services and/or KV
- Consul Snapshot Agent – needs access to take snapshots
- Consul Administrative Tasks – any task, including consul members will require a token

The anonymous token can be configured to permit certain actions that will not require a token

Creating Tokens

Creating a Token (CLI)

Use `consul acl token` command:

- `clone` – clone an existing token to create a new one
- `create` – create a new token
- `delete` – delete a token
- `list` – list all tokens
- `read` – display the details of a token
- `update` – update an existing token (such as change policy or other attributes

Terminal

```
$ consul acl token create -description "Token for eCommerce Web" -policy-id 06acc965-df4b-5a99-58cb-3250930c6324

AccessorID:   986193b5-e2b5-eb26-6264-b524ea60cc6d
SecretID:     ec15675e-2999-d789-832e-8c4794daa8d7
Description:  Policy for eCommerce App
Local:        false
Create Time:  2021-02-14 02:14:314.21421 -0400 EDT
Policies:
   06acc965-df4b-5a99-58cb-3250930c6324 - eCommerce
```

Actual Token used for Consul requests

Creating Tokens

Creating a Policy (API)

Create Policies with the Consul API

- Method: PUT
- Endpoint: /acl/token
- Response: Includes the AccessorID, SecretID, Policies, etc.

Terminal

```
$ curl -X PUT \
    --header "X-Consul-Token: 45a3bd52-07c7-47a4-52fd-0745e0cfe967"
    --data @payload.json \
    https://consul.example.com:8500/v1/acl/token
```

Required if ACLs are enabled (assuming Deny default policy)

Terminal

```
$ cat payload.json
{
  "Description": "Token for eCommerce Web Servers",
  "Policies": [
     {
       "ID": "06acc965-df4b-5a99-58cb-3250930c6324"
     }
   ]
}
```

Can be linked to a policy by using the ID and/or Policy Name

Perform a UI task using a Token

ACL Tab

1

dc1 Services Nodes Key/Value ACL Intentions Documentation

Access Controls

The Access Control List (ACL) system, which controls permissions for this UI, is enabled for this cluster. Your token ID will be saved locally and persist through visits.

SecretID or Token

SecretID or Token

Save

Enter Valid Token Here

2

dc1 Services Nodes Key/Value ACL Intentions Documentation

Access Controls

Tokens Policies

Create/Manage Tokens and Policies

Create

Search

Accessor ID	Scope	Description	Policies	Actions
b08f50ab	global	Bootstrap Token (Global Management)	global-management	Your token •••
00000000	global	Anonymous Token		•••

Perform a CLI request using a Token

Many options for providing the ACL token when using the CLI

- Set the environment variable CONSUL_HTTP_TOKEN
  - `$ export CONSUL_HTTP_TOKEN=ec15675e-2999-d789-832e-8c4794daa8d7`
- Use the `-token` argument
  - `consul members -token ec15675e-2999-d789-832e-8c4794daa8d7`
- Set the environment variable CONSUL_HTTP_TOKEN_FILE
  - `$ export CONSUL_HTTP_TOKEN_FILE=/etc/consul/token.txt`
- Refer to the token stored in a file using the `-token-file` argument
  - `consul members -token-file @token.txt`

Perform an API request using a Token

Two Options for setting the token for the API

- Set the header X-Consul-Token
  - `--header "X-Consul-Token: ec15675e-2999-d789-832e-8c4794daa8d7"`
- Set the header Authorization: Bearer
  - `--header "X-Consul-Token: ec15675e-2999-d789-832e-8c4794daa8d7`

Terminal

```
$ curl -X PUT \
    --header "X-Consul-Token: ec15675e-2999-d789-832e-8c4794daa8d7"
    --data @payload.json \
    https://consul.example.com:8500/v1/acl/token
```

Terminal

```
$ curl -X PUT \
    --header "Authorization: Bearer ec15675e-2999-d789-832e-8c4794daa8d7"
    --data @payload.json \
    https://consul.example.com:8500/v1/acl/token
```

Secure Services with Access Control Lists (ACLs)

Objective 8a: Set up and configure a basic ACL system

Objective 8b: Create policies

Objective 8c: Manage token lifecycle: multiple policies, token revoking, ACL roles, service identities

Objective 8d: Perform a CLI request using a token

Objective 8e: Perform an API request using a token

1	2	3	4	5

Difficulty Level

END OF SECTION