Exam Objective 3

Register Services and Use Service Discovery

Register Services and Use Service Discovery

Objective 3a: Interpret a service registration

Objective 3b: Differentiate ways to register a single service

Objective 3c: Interpret a service configuration with health check

Objective 3d: Check the service catalog status from the output of the DNS/API interface or via the Consul UI

Objective 3e: Interpret a prepared query

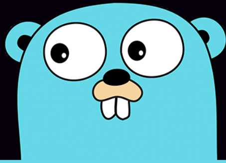
Objective 3f: Use a prepared query

1	2	3	4	5

Difficulty Level

What is a Service?

Web Application



Java App Microservice

Database

Search

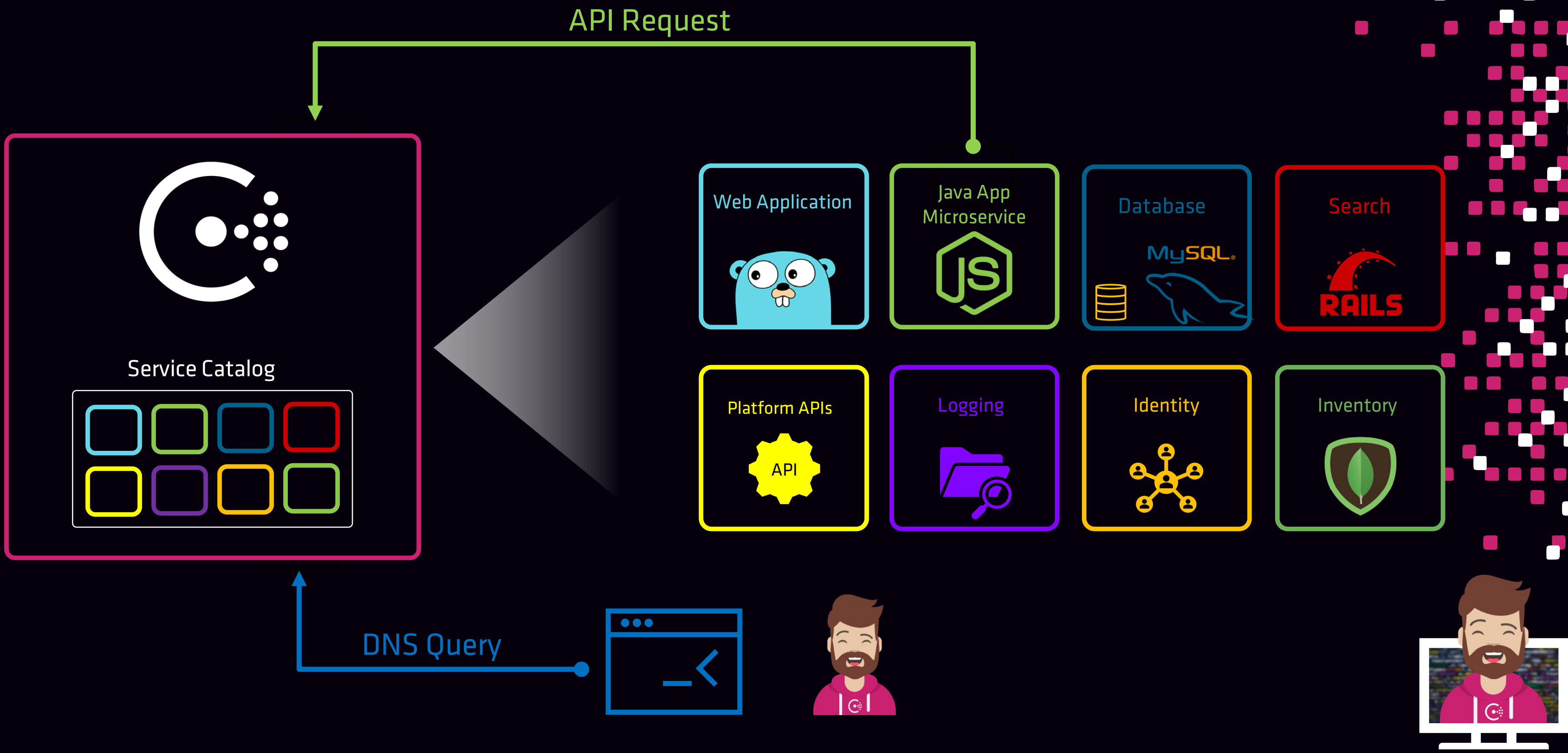
Platform APIs

Logging

Identity

Inventory

What is a Service?



Registering a Service

- How do I register a service in Consul?
  - Register with the local agent using:
    - Service Definition File
    - HTTP API

- Service Registration typically happens when a new service is provisioned
  - Container is scheduled by Kubernetes
  - Instance is deployed via Terraform
  - Jenkins provisions new VMs on a VMware cluster

Registering a Service

- Register with the Consul API
  - Method: PUT
  - Endpoint: /v1/agent/service/register

Terminal

```
$ curl \
    --request PUT \
    --data @payload.json \
    https://consul.example.com:8500/v1/agent/service/register
```

Terminal

```
$ cat payload.json
{
  "service": {
    "name": "retail-web",
    "port": 8080
   }
}
```

Registering a Service

Register with a service definition

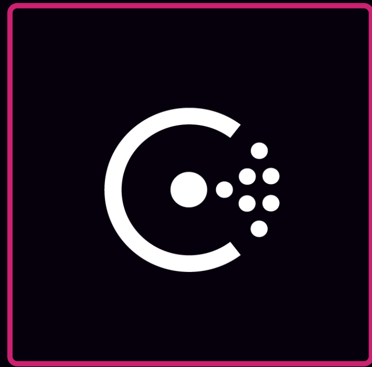
- Define a service using a service definition file
  - .hcl
  - .json

Multiple options to register the service using a service definition:

1. Create a single file and set using the -config-file parameter
2. Place file inside of the -config-dir directory <read at startup>
3. Run the `consul services register` command using file
4. Execute a `consul reload` command after adding file

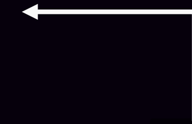
Registering a Service

Consul Clients



Consul Server

Service
Registration



Terminal

```
$ consul services register
```

Service
Definition

web-app-01

Service
Definition

web-app-01

Service
Definition

web-app-01

Creating a Service Definition

- File that defines a service to be registered in Consul
- Once registered, the service is added to the Consul service registry as an available* service

- Parameters included in the service definition may include:
  - Service Name
  - ID of the agent
  - Tags
  - IP Address and Port of the service
  - Health Checks

Creating a Service Definition

```
{
  "service": {
    "id": "web-server-01",
    "name": "front-end-eCommerce",
    "tags": ["v7.05", "production"],
    "address": "10.3.13.112",
    "port": 8080,
    "checks": [
      {
        "args": ["/usr/local/bin/check_mem.py"],
        "interval": "30s"
      }
    ],
  }
}
```

Name of the node (server/container)

Name of the service to be registered

Optional Tags for the service

IP Address/Interface for the Service

Port the Service is running on

Health Check for the Service

https://www.consul.io/docs/discovery/services

Creating a Service Definition

- Defaults
  - ID will be set to the Name if not set
  - Address will be set to the default address of Consul agent
- Default namespace for a registered service:
  - <name>.service.consul
  - front-end-eCommerce.service.consul

```
{
  "service": {
    "id": "web-server-01",
    "name": "front-end-eCommerce",
    "tags": ["v7.05", "production"],
    "address": "10.3.13.112",
    "port": 8080,
    "checks": [
      {
        "args": ["/usr/local/bin/check_mem.py"],
        "interval": "30s"
      }
    ],
  }
}
```

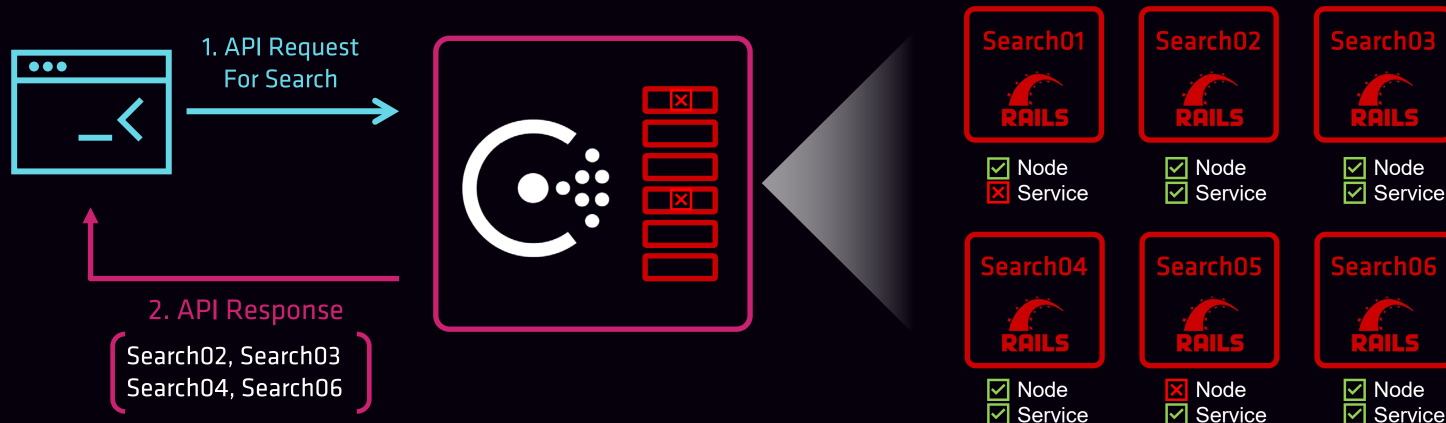
Creating a Service Definition

- Each ID should be unique per agent
  - web-server-01
  - web-server-02
  - web-server-03
- Default namespace for a registered service:
  - <name>.service.consul
  - front-end-eCommerce.service.consul

```
{
  "service": {
    "id": "web-server-01",
    "name": "front-end-eCommerce",
    "tags": ["v7.05", "production"],
    "address": "10.3.13.112",
    "port": 8080,
    "checks": [
      {
        "args": ["/usr/local/bin/check_mem.py"],
        "interval": "30s"
      }
    ],
  }
}
```

Creating a Service Definition

- Multiple nodes in the catalog providing the same service
  - Provides high-availability and elasticity
  - Only registered services passing health checks will be returned



Configuring a Service Health Check

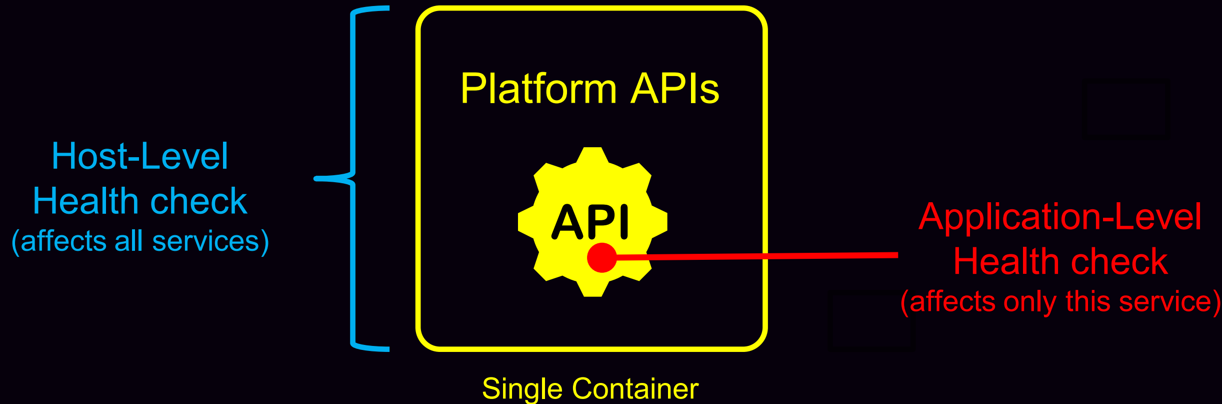
- Health checks determine when the node or service is healthy
- Health checks can be created/updated via API or a Service config

- Health check configuration may include:
  - Name
  - Arguments based on the type of health check
  - Interval (how often the check will run)
  - Additional parameters based on the type of health check

Configuring a Service Health Check

Types of health checks

- Application-level (service) health check
- System-level (node) health check

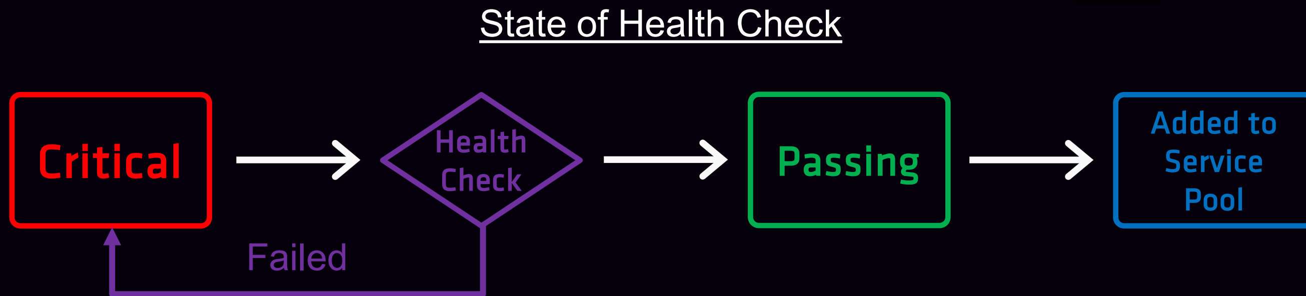


Configuring a Service Health Check

A service may have multiple health checks defined

- If any health check(s) are failing, the node is omitted from service queries

- By default, newly registered health checks are set to 'critical'
  - Ensures that services aren't added to service pool before they are confirmed to be healthy



Types of Health Checks

- **Script Health Check** – Execute a specified script
- **HTTP Health Check** – Perform GET looking for a 2xx return code
- **TCP Health Check** – Make TCP connection to IP/Port
- **TTL Health Check** – Relies on app to report health to endpoint
- **Docker Health Check** – Invoke app in a Docker container
- **gRPC Health Check** – Probe a gRPC health check endpoint
- **Alias Health Check** – Determine health/state of another registered service

Types of Health Checks

Script Health Check

```
{
  "check": {
    "id": "mem-util",
    "name": "Memory Utilization",
    "args": ["/opt/check_mem.py", "-limit", "256MB"],
    "interval": "10s",
    "timeout": "1s"
  }
}
```

Fail if above this limit

Run this script

HTTP Health Check

```
{
  "check": {
    "id": "api",
    "name": "HTTP API on port 5000",
    "http": "https://localhost:5000/health",
    "tls_skip_verify": false,
    "method": "POST",
    "header": {"Content-Type": ["application/json"]},
    "body": "{\"method\":\"health\"}",
    "interval": "10s",
    "timeout": "1s"
  }
}
```

Query this URL for a 2xx return code

Do this every 10 sec

https://www.consul.io/docs/discovery/checks

Types of Health Checks

TCP Health Check

```
{
  "check": {
    "id": "ssh",
    "name": "SSH TCP on port 22",
    "tcp": "localhost:22",
    "interval": "10s",
    "timeout": "1s"
  }
}
```

Establish connectivity to this address/port

Do this every 10 sec

Docker Health Check

```
{
  "check": {
    "id": "mem-util",
    "name": "Memory utilization",
    "docker_container_id": "f972c95ebf0e",
    "shell": "/bin/bash",
    "args": ["/usr/local/bin/check_mem.py"],
    "interval": "10s"
  }
}
```

Connect to this Docker container

Run this script on the Docker container

https://www.consul.io/docs/discovery/checks

Checking the Service Status from Catalog

- Multiple ways to determine the status of services
  1. DNS Query – most commonly used
  2. API Request – requires app integration
  3. Consul UI – least commonly used

Checking the Service Status from Catalog

DNS Query

Java App
Microservice

5. Establish Connectivity

Database

MySQL

4. Respond to
DNS Query

3. Return Healthy Nodes

java-app

Database

DNS

1. DNS Request

2. Forward Request
to Consul

DNS Forwarder for .consul = <consul nodes>

Checking the Service Status from Catalog

DNS Query

DNS Query via 'Dig'

(1) Healthy Node Found

IP Address of Available Service

Terminal

```
$ dig @10.0.3.45 –p 8600 front-end-eCommerce.service.consul

; <<>> DiG 9.10.6 <<>> @10.0.3.45 -p 8600 front-end-eCommerce.service.consul
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 28340
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 2
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
; front-end-eCommerce.service.consul.        IN  A

;; ANSWER SECTION:
front-end-eCommerce.service.consul. 0   IN  A   10.3.15.67

;; ADDITIONAL SECTION:
front-end-eCommerce.service.consul. 0   IN  TXT "consul-network-segment="

;; Query time: 2 msec
;; SERVER: 10.0.3.45 #8600(10.0.3.45)
;; WHEN:  Fri Jan 01 00:00:00 EDT 2021
;; MSG SIZE  rcvd: 99
```

Checking the Service Status from Catalog

API Request

Java App
Microservice

3. Establish Connectivity

Database

MySQL

2. API Response

Java-app

Database

1. API Request

Checking the Service Status from Catalog

API Request

Terminal

```
$ curl --request GET http://10.0.3.45:8500/v1/catalog/service/front-end-eCommerce
[
  {
    "ID": "c3efa2a6-226f-c304-bff9-2869da16431a",
    "Node": "web-server-01",
    "Address": "10.3.15.67",
    "Datacenter": "dc1",
    "TaggedAddresses": {
      "lan": "10.3.15.67",
      "lan_ipv4": "10.3.15.67",
      "wan": "10.3.15.67",
      "wan_ipv4": "10.3.15.67"
    },
    "NodeMeta": {
      "consul-network-segment": ""
    },
    "ServiceKind": "",
    "ServiceID": "web-server-01",
    "ServiceName": "front-end-eCommerce",
    "ServiceTags": [
      "v7.05",
      "production"
    ],
    "ServiceAddress": "",
    "ServiceWeights": {
      "Passing": 1,
      "Warning": 1
    },
.
.
.
```

IP Address of a Health Node

Tags

API Endpoint for Service

# of Nodes Passing Health Checks

Checking the Service Status from Catalog

Consul UI

Introduction to Prepared Query

- Allows you to create and register a more complex service query so it can be executed later
  - Allows for richer queries than just DNS alone
  - Used to filter the results of a service request
  - Objects defined at the datacenter level

- Created by using the /query API endpoint
- Consumed by either API or DNS query
  - <name>.query.consul

Introduction to Prepared Query

I need a new vehicle

It must be a car

It must be red

It must be the latest model

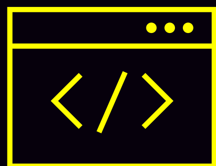
Introduction to Prepared Query

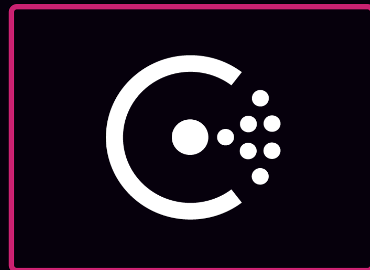
I need to connect to a service

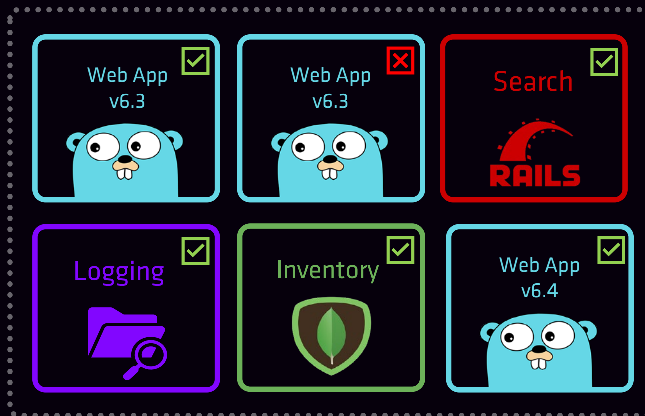
It must be healthy

It must web-app

It must have tag v6.4







Service Catalog

Introduction to Prepared Query

My First Prepared Query

```
{
  "Name": "web-app-v64",
  "Service": {
    "Service": "web-app",
    "Tags": ["v6.4"]
   }
}
```

Name of the Prepared Query

The Service We Want

Only Return Services with Matching Tags

Executing the Prepared Query

- DNS: web-app-v64.query.consul
- API: https://consul.example.com:8500/v1/query/<uuid>/execute

Adding Failover Policies

- When multiple datacenters are federated, we can extend prepared queries to return services in other datacenters
  - Extension of Prepared Queries
  - Transparent to Applications
  - Determines Target for a Service Request

Failover Policies - Example

Azure

aws

Types of Failover Policies

- Multiple options for configuring a failover policy
  - Static Policy – fixed list of the order of failover
  - Dynamic Policy – send to nearest DC based on RTT
  - Hybrid Policy – use shortest RTT first, then use other DCs
- Failover Policies will try to return a LOCAL service first before returning a service from a Federated datacenter

Configuring Failover Policies

My First Prepared Query + Failover Policy

```
{
  "Name": "web-app-v64",
  "Service": {
    "Service": "web-app",
    "Tags": ["v6.4"],
     "Failover": {
        "Datacenters": ["dc2", "dc3"]
      }
   }
}
```

Name of the Prepared Query

New Failover Policy

Static Failover Policy

If local isn't available, try dc2 first, then dc3

Adding Failover Policies

My First Prepared Query + Failover Policy

```
{
  "Name": "web-app-v64",
  "Service": {
    "Service": "web-app",
    "Tags": ["v6.4"],
     "Failover": {
        "NearestN": 2,
     }
  }
}
```

Dynamic Failover Policy

If local isn't available, try 2 other datacenters starting with the lowest RTT

Hybrid Failover Policy

If local isn't available, try 2 other datacenters starting with the lowest RTT. If that fails, use dc2 and then dc3.

```
{
  "Name": "web-app-v64",
  "Service": {
    "Service": "web-app",
    "Tags": ["v6.4"],
     "Failover": {
        "NearestN": 2
        "Datacenters": ["dc2", "dc3"]
     }
  }
}
```

Each datacenter is only queried one time during a failover

Compare Options for Querying Services

DNS	Simple, but Not Flexible
Prepared Query	Dynamic, but Local
Failover Policy	Ideal Solution for Multi-Cloud Apps

Using a Prepared Query

- Prepared Query can be used via API or DNS
  - Default namespace = <name>.query.consul
  - API uses the UUID for the prepared query after creation
- Executing the Prepared Query
  - DNS: web-app-v64.query.consul
  - API: https://consul.example.com:8500/v1/query/<uuid>/execute

Using a Prepared Query

- Order of Operations
    - Local service is returned first
    - If local is not available, failover policy is used

Register Services and Use Service Discovery

Objective 3a: Interpret a service registration

Objective 3b: Differentiate ways to register a single service

Objective 3c: Interpret a service configuration with health check

Objective 3d: Check the service catalog status from the output of the DNS/API interface or via the Consul UI

Objective 3e: Interpret a prepared query

Objective 3f: Use a prepared query

1	2	3	4	5

Difficulty Level

END OF SECTION