Consul is a distributed service mesh and service discovery tool designed for modern cloud environments. It provides a centralized platform for service networking and configuration management, allowing organizations to easily deploy and manage complex applications across multiple datacenters and cloud environments.

Consul's key features include:

Service discovery: Consul provides a dynamic service registry that automatically detects and tracks the availability of services across multiple datacenters and cloud environments.

Health checking: Consul includes a built-in health checking system that allows services to be monitored for availability and performance.

Load balancing: Consul supports multiple load balancing algorithms, allowing traffic to be evenly distributed across multiple instances of a service.

Key/value store: Consul includes a distributed key/value store that can be used to store configuration data and other shared resources.

Secure service communication: Consul provides built-in support for TLS encryption and mutual authentication, allowing services to securely communicate with each other.

Service segmentation: Consul allows services to be segmented into distinct network segments, improving security and performance.

Consul can be used with a variety of popular deployment tools and frameworks, including Kubernetes, Docker, and Terraform. It also provides a robust API and command-line interface, making it easy to integrate with existing automation and monitoring tools.

Overall, Consul is a powerful tool for managing and orchestrating complex distributed applications in modern cloud environments, providing a unified platform for service discovery, configuration management, and secure communication.

Created by Hashicorp, Consul solves the problem of service discovery and configuration for distributed, high availability applications.

Consul has four primary responsibilities:

Service Discovery: Locate where services, such as containers, are running within your cluster via a api or dns.

Key/Value Store: Use Consul as a highly available data store for data such as configuration details or feature switching.

Health Checking: Automatically route traffic away from unhealthy nodes.

Multi Datacentre: Out the box, Consul is data centre aware.

In this scenario, we will use the official Docker image to launch a cluster of Consul agents. We'll then replace a node to indicate the health checking options via the command line and UI.

Start Agent
To start, we'll launch a primary Consul agent which others will connect too. Each of these agents will run as separate containers on the same node. In production, each of the containers would run on different machines.

The agents will form the initial cluster and Consul Quorum for ensuring data stored is consistent and remains available after outages.

Task: Launch Agent
To start an launch we use the official Docker image and provide the arguments agent -dev. To make the UI accessible we tell Consul to bind to the public IP instead of 127.0.0.1. The UI runs on port 8500 which we map to the host.

```
docker run -d --name=c1 -p 8500:8500 consul agent -dev -client=0.0.0.0 -bind=0.0.0.0
```

Because we haven't told this agent to join a cluster, it will create one for us. In the next step we'll add additional agents.

This docker run command starts a Docker container running Consul with the following options:

-d: Runs the container in detached mode, meaning that it runs in the background.
--name=c1: Names the container "c1".
-p 8500:8500: Maps port 8500 on the container to port 8500 on the host, allowing access to the Consul web UI.
consul: Specifies the name of the Docker image to use (in this case, the official Consul image).
agent: Specifies that the Consul agent should be started.
-dev: Starts the agent in development mode, which disables security features and runs in a single-node configuration.
-client=0.0.0.0: Sets the client interface to 0.0.0.0, allowing incoming connections from any IP address.
-bind=0.0.0.0: Sets the bind address to 0.0.0.0, allowing Consul to listen on all available network interfaces.
When this command is run, a new Docker container running Consul is created with the specified options. The container runs in the background, and the Consul web UI can be accessed by opening a web browser and navigating to http://localhost:8500.

Add Additional Agents
Once the agent has started, we can tell other agents to join the newly created cluster. Agents register using the provided the IP address of how to communicate with another Consul agent.

Task: Add two additional Consul agents
We can use Docker to obtain the IP address of the first container we started. We also output the IP address of the container to the terminal, it should be in the 172.18.0.0 range.

IP=$(docker inspect --format '{{ .NetworkSettings.IPAddress }}' c1); echo $IP

When we launch our two additional agents we tell them how to locate the cluster.

docker run -d --name c2 consul agent -dev -bind=0.0.0.0 -join=$IP

docker run -d --name c3 consul agent -dev -bind=0.0.0.0 -join=$IP

```
tp://localhost:8500.^C
$ docker inspect c1 --format '{{.NetworkSettings.IPAddress}}'
172.18.0.2
$ IP=$(docker inspect c1 --format '{{.NetworkSettings.IPAddress}}')
$ echo $IP
172.18.0.2
$ docker run -d --name c2 consul agent -dev -bind=0.0.0.0 -join=$IP
763baeb1dd29f7e4a6ebd066f3be7e450a171e3a1481b668d094631b9dd22f29
$ docker run -d --name c3 consul agent -dev -bind=0.0.0.0 -join=$IP
e89cb58581c9b93f399169e4c0b4d54c0ae0999623afc425f19af91eac91bf6a
$ docker logs c1
==> Starting Consul agent...
               Version: '1.15.2'
            Build Date: '2023-03-30 17:51:19 +0000 UTC'
               Node ID: 'ca6c8769-6177-2e84-41fb-590cc608957f'
             Node name: '7fb7fb8c013c'
            Datacenter: 'dc1' (Segment: '<all>')
                Server: true (Bootstrap: false)
           Client Addr: [0.0.0.0] (HTTP: 8500, HTTPS: -1, gRPC: 8502, gRP
          Cluster Addr: 172.18.0.2 (LAN: 8301, WAN: 8302)
     Gossip Encryption: false
```

We will now have a Consul cluster running on our Docker daemon. You can see the communicate and events via the logs
docker logs c1

In the next step we explore what happens if nodes fail.
Simulate Node Failure
We can list all the members of a Consul cluster by asking one of our Containers
docker exec -t c1 consul members

The docker exec -t c1 consul members command executes the consul members command inside the Docker container named "c1". The consul members command is used to list the members of a Consul cluster.

When the command is run, Docker locates the running container named "c1" and executes the consul members command inside that container. The output of the command is then displayed in the terminal.

The consul members command lists the members of the Consul cluster, including their IP address, node name, status, and other details. This can be useful for monitoring the health of the cluster and identifying any issues that may arise.

```
$ docker exec -t c1 consul members
Node          Address          Status  Type    Build   Protocol  DC   Partition  Segment
763baeb1dd29  172.18.0.3:8301  alive   server  1.15.2  2         dc1  default    <all>
7fb7fb8c013c  172.18.0.2:8301  alive   server  1.15.2  2         dc1  default    <all>
e89cb58581c9  172.18.0.4:8301  alive   server  1.15.2  2         dc1  default    <all>
$
```

Task: Simulate Outage
We can simulate a network outage or machine failure, via
docker kill c3.

After a couple of seconds, the heartbeat and health check will fail. The state of the member will change from active to failed.

docker exec -t c1 consul members

If this node was elected the leader then another election would have taken place. This can be viewed in the logs docker logs c1

```
00/tcp   c1
$ docker exec -t c1 consul members
Node          Address          Status  Type    Build   Protocol  DC   Partition  Segment
763baeb1dd29  172.18.0.3:8301  alive   server  1.15.2  2         dc1  default    <all>
7fb7fb8c013c  172.18.0.2:8301  alive   server  1.15.2  2         dc1  default    <all>
e89cb58581c9  172.18.0.4:8301  alive   server  1.15.2  2         dc1  default    <all>
$ docker kill c3
c3
$ docker exec -t c1 consul members
Node          Address          Status  Type    Build   Protocol  DC   Partition  Segment
763baeb1dd29  172.18.0.3:8301  alive   server  1.15.2  2         dc1  default    <all>
7fb7fb8c013c  172.18.0.2:8301  alive   server  1.15.2  2         dc1  default    <all>
e89cb58581c9  172.18.0.4:8301  left    server  1.15.2  2         dc1  default    <all>
$ docker logs c1
```

Consul UI
Along with the CLI, you can see the state of the cluster via the UI. It is accessible on port 8500 via this link
https://fa2b70102d6c44aba76db41c3f2731bc-167772166-8500-host08nc.environments.katacoda.com/ui/

The Services tab list provides a catalog of services registered with Consul. A service can be running on multiple nodes. In this case, Consul would be the server running on two ones, with one failed node. Services could be web applications, databases, brokers, etc that other applications need to discover to communicate with.

The Nodes tab lists all nodes where Services are running. If any Nodes have previously failed they'll be listed here along with health check was the reason it was marked unhealthy. This gives you an overview of your cluster state.

You now have a Consul cluster running as Docker containers.

In this scenario, we explored how to launch a Consul cluster running as Docker containers. Once running, we simulated a node failure and how to identify the health via the CLI and UI.

In future scenarios, we'll explore how to Consul to manage API and DNS requests.