HashiCorp
Consul

# Consul Foundations

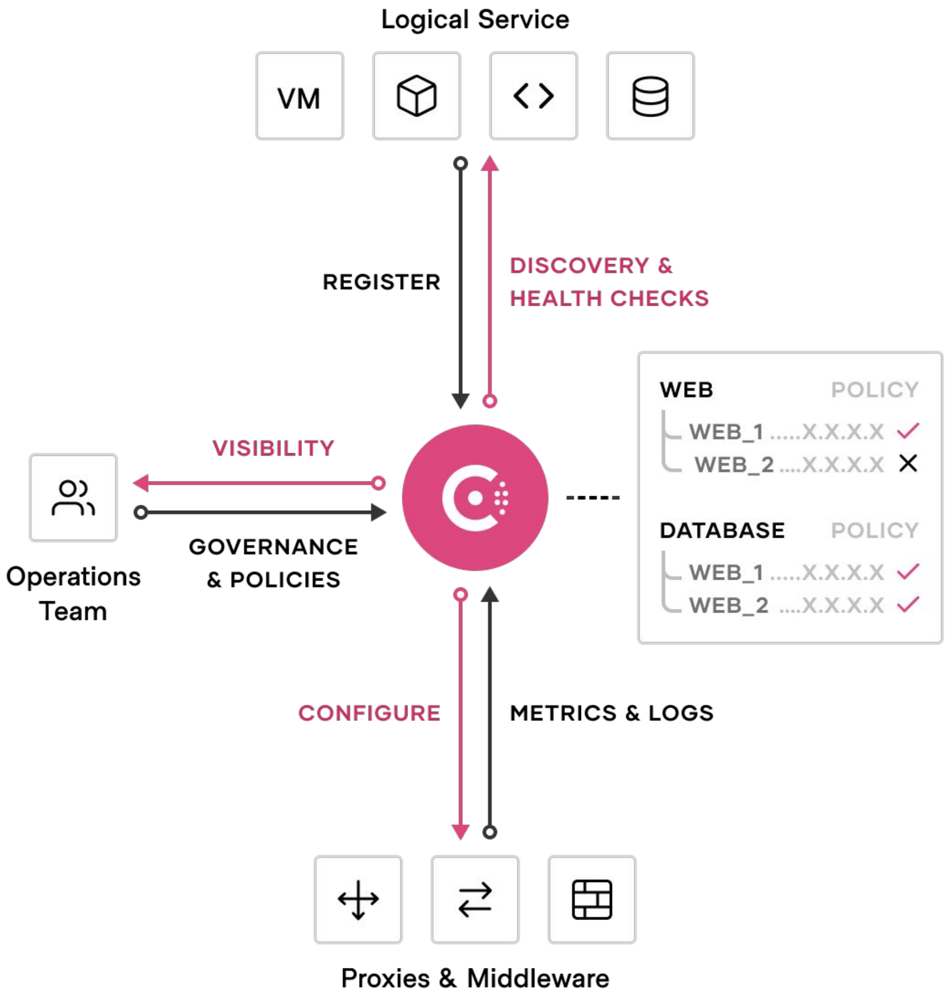
## Agenda

# Consul Enterprise Path to Production

| Week 1 | Week 2 | Week 3 | Week 4 | Week 5 |
|---|---|---|---|---|
| **Kickoff** | **Consul Foundations** | **Consul Deployment & Operations** | **Office Hours** | **Advanced Concepts** |
| ● Consul overview & Architectural deep dive | ● Service Discovery, Health Checks & Consul DNS | ● Deployment, Autopilot, Backup, Telemetry & Monitoring | ● Open forum for Q&A | ● Federation, Network Segments, Namespaces, & Runbooks<br>● Exit Ramp & Operational Readiness Checklist (asynchronous) |

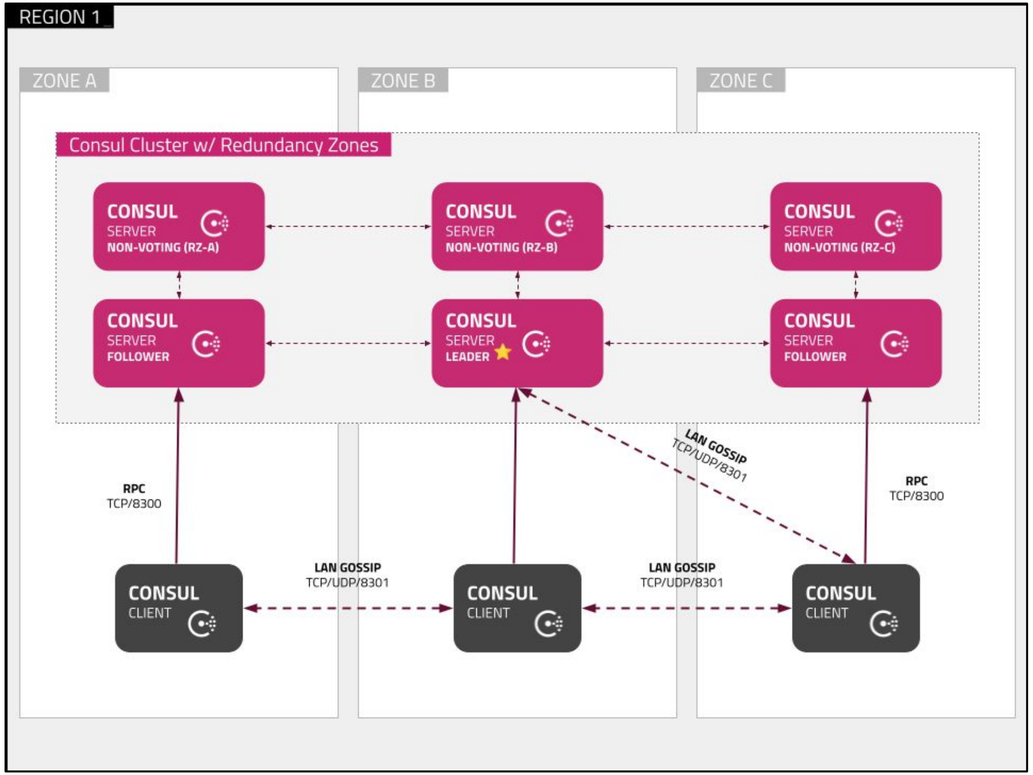
# Service Networking

Discover and securely connect any service on any cloud or runtime



# Consul Enterprise Reference Architecture

- Provides a highly resilient and scalable deployment for a single Consul cluster
- 6 node cluster with 3 non-voting nodes is capable of withstanding the loss of two nodes or an entire Availability Zone (AZ)
- Uses Consul Enterprise Autopilot and non-voting nodes for redundancy
- [Consul and Kubernetes Deployment Guide]()



01

## Service Discovery & Registration

## Service Registry & Discovery

- Discover, track, & monitor health of services in a network
- Register & maintain a record of all services in a service catalog
- A single source of truth for services to query & communicate with each other
- Dynamically locate any application or infrastructure service to simplify network connectivity

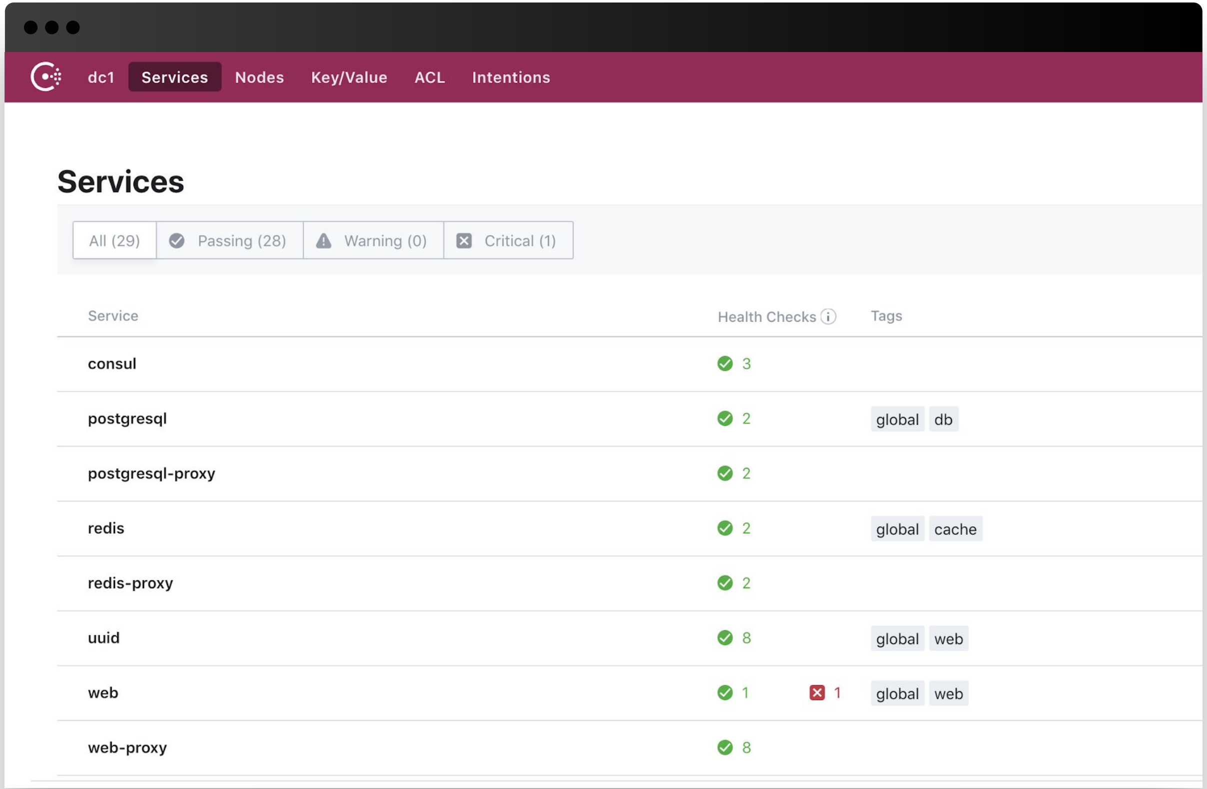
## Service Registry & Discovery

- Eliminate the need for East/West Load Balancing
- Enable other Consul use cases
    - Core building block of a Service Mesh
    - Software Load Balancing
    - Network Infrastructure Automation
- Automate Geographic Failover using Prepared Queries

# Service Registry

Consul catalog provides a real-time directory which includes:

- What services are running
- Service network location
- Service health status
- Platform agnostic views



## Services

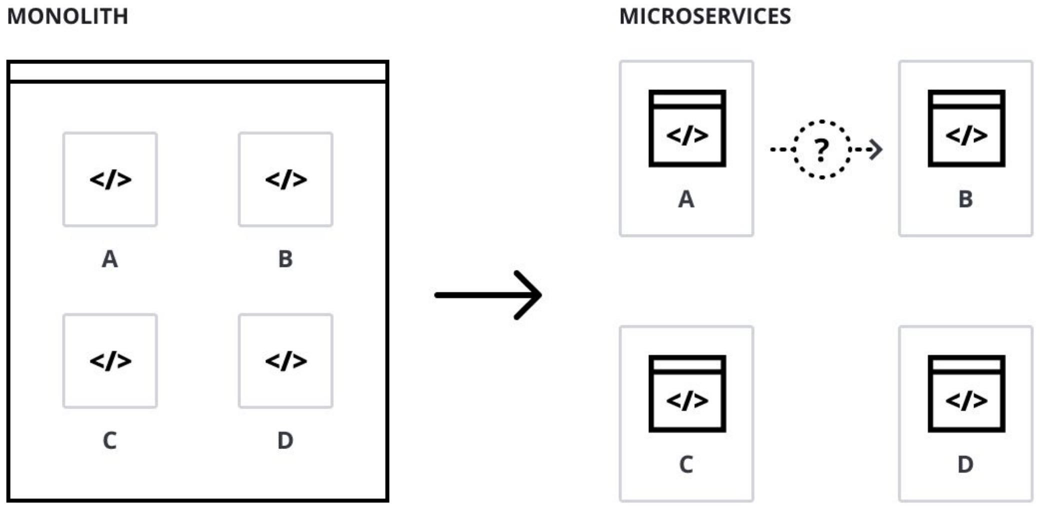
All (29) | ✓ Passing (28) | ⚠ Warning (0) | ☒ Critical (1)

| Service | Health Checks | | Tags |
|---|---|---|---|
| consul | ✓ 3 | | |
| postgresql | ✓ 2 | | global db |
| postgresql-proxy | ✓ 2 | | |
| redis | ✓ 2 | | global cache |
| redis-proxy | ✓ 2 | | |
| uuid | ✓ 8 | | global web |
| web | ✓ 1 | ☒ 1 | global web |
| web-proxy | ✓ 8 | | |

## Service Registry

**Microservices cause east-west traffic growth**

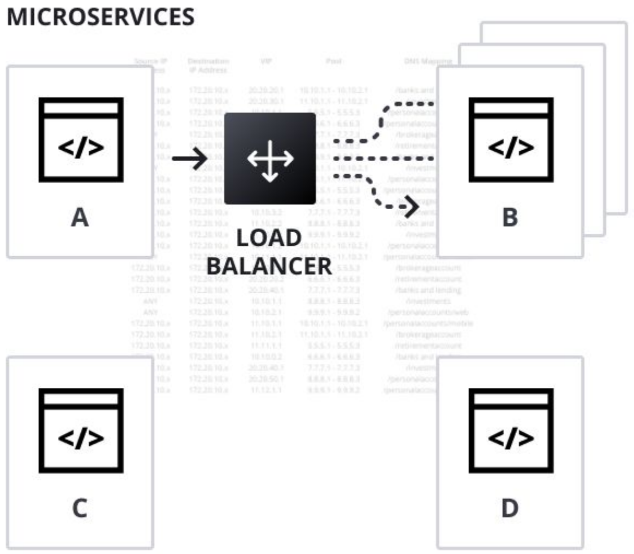
- Microservices communicate over the network in east-west traffic patterns
- Service-to-service traffic needs to be routed dynamically as services scale up and down frequently without long-lived IPs.



# Service Registry

**Load balancers for east-west traffic scale poorly**

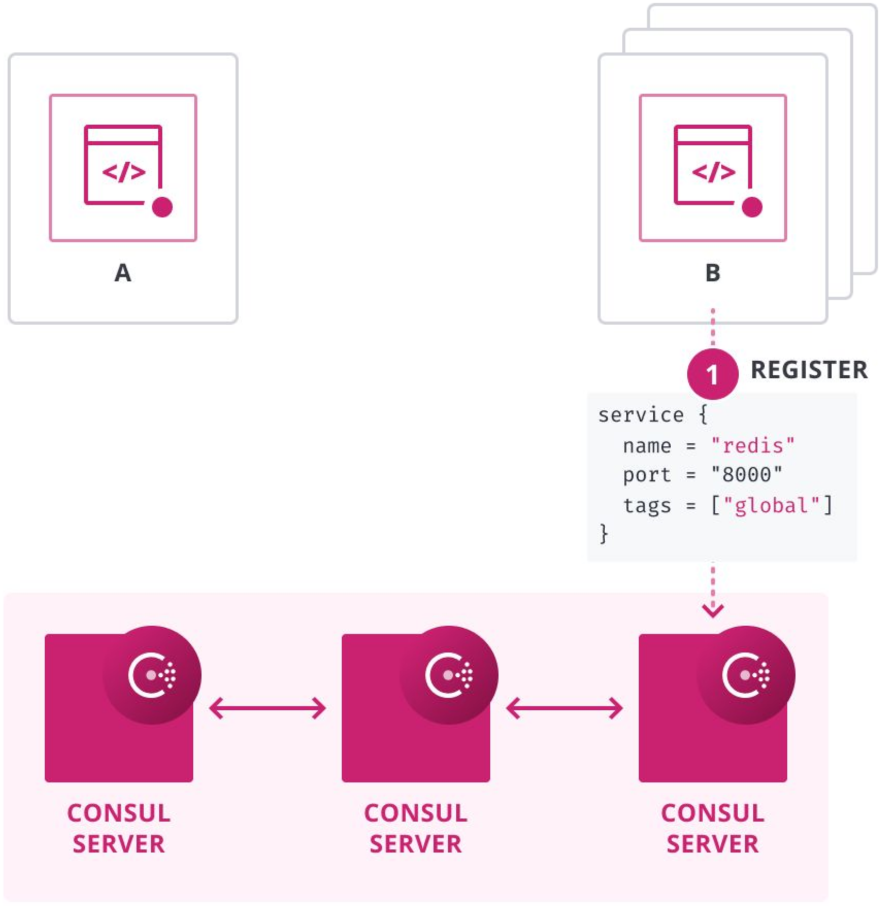
- Load balancers can front a service tier and provide a static IP
- Load balancers add cost, latency, single points of failure, and must be updated as services scale up/down.



# Service Registration

## Service discovery for connectivity

- Consul provides a registry of all the running nodes and services with current health status
- Services can register to mark themselves (IP + port) as available via config files or API

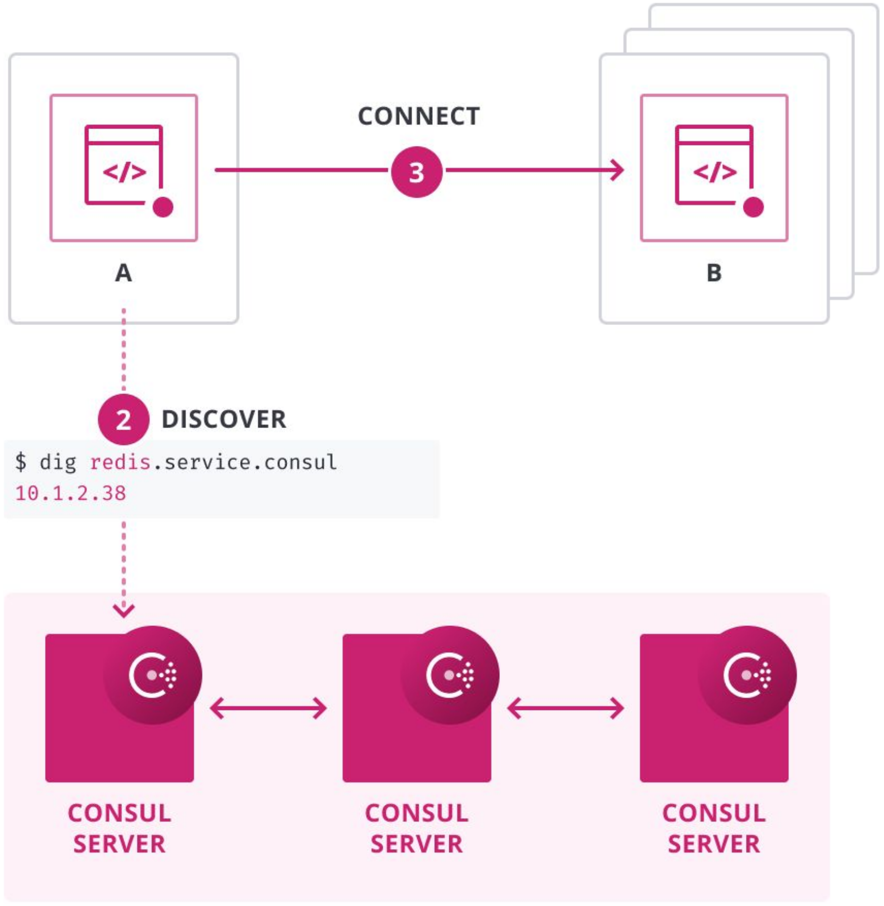


```
service {
  name = "redis"
  port = "8000"
  tags = ["global"]
}
```

# Service Registration

**Allow services to connect directly**

- For a service to communicate with any other service it queries the registry for the healthy instances of those services
- Two services can connect directly without any operator intervention
- Service catalog can be queried via DNS or API



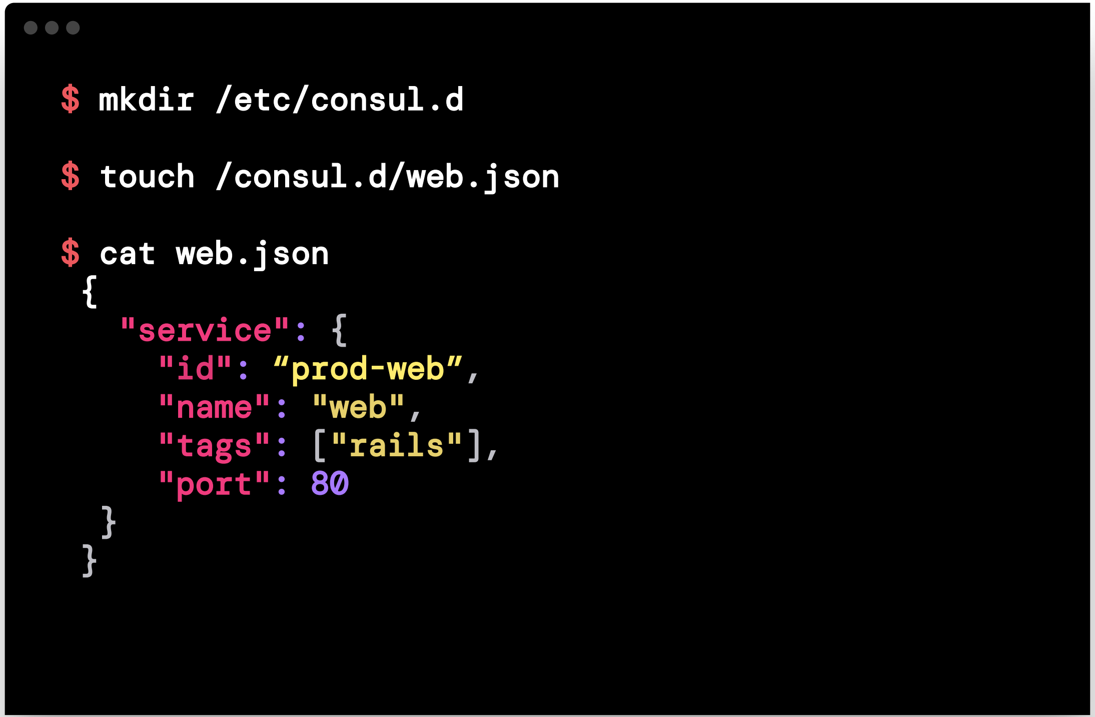
```
$ dig redis.service.consul
10.1.2.38
```

## Service Discovery for Deployment

- Cross Platform Deployment
  - Make applications deployed across multiple platforms and clouds available for consumption
  - Simplify operations
- Blue / Green Deployments
  - On-premise to cloud migration
  - Upgrade of a set of hosts for routine maintenance
- Blue / Green / Yellow / Grey
  - Exposing a specific version of an app
  - Leveraging rich metadata to target specific instances of a service

# Define a Service

Sample service definition



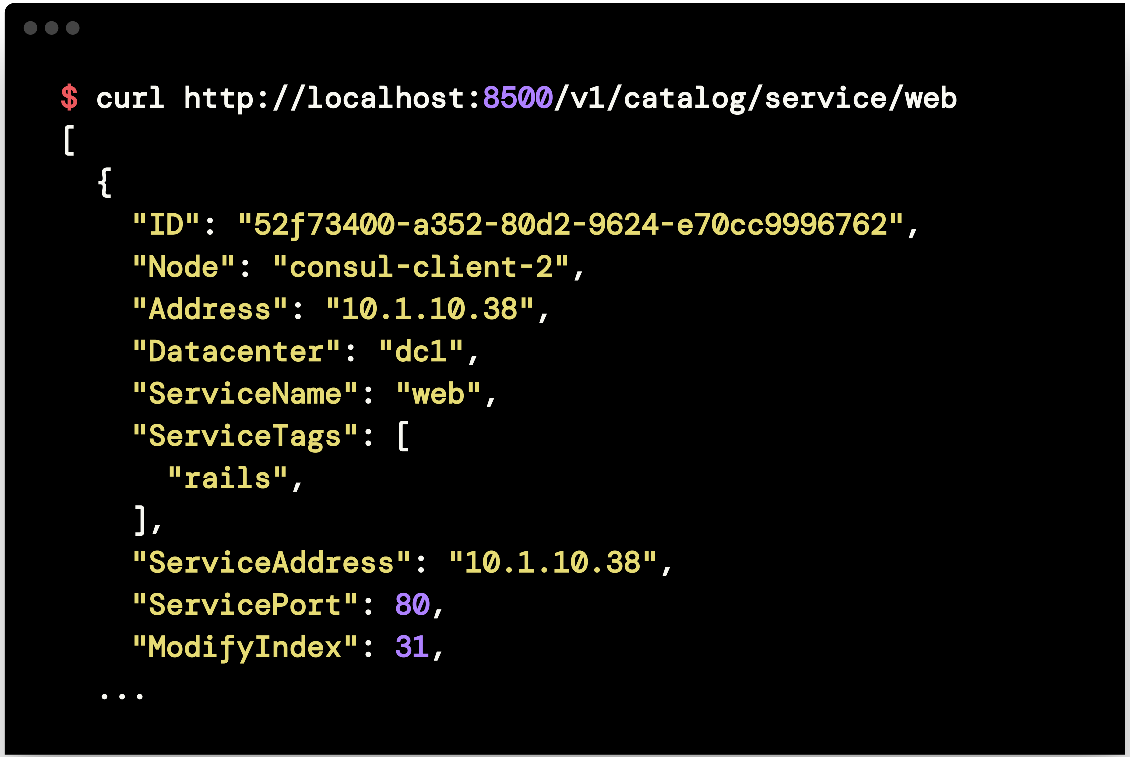
```
$ mkdir /etc/consul.d

$ touch /consul.d/web.json

$ cat web.json
 {
   "service": {
     "id": “prod-web”,
     "name": "web",
     "tags": ["rails"],
     "port": 80
  }
 }
```

# HTTP API Interface

- The Consul service registry API allows for more complex tasks beyond basic DNS functionality
- API calls can query the service registry for nodes, services, and health check information
- API supports blocking queries, or long polling, for changes
- Automation and IAC tools can respond to service registrations or health status changes to update configurations or traffic routing in real time



```
$ curl http://localhost:8500/v1/catalog/service/web
[
  {
    "ID": "52f73400-a352-80d2-9624-e70cc9996762",
    "Node": "consul-client-2",
    "Address": "10.1.10.38",
    "Datacenter": "dc1",
    "ServiceName": "web",
    "ServiceTags": [
      "rails",
    ],
    "ServiceAddress": "10.1.10.38",
    "ServicePort": 80,
    "ModifyIndex": 31,
  ...
```

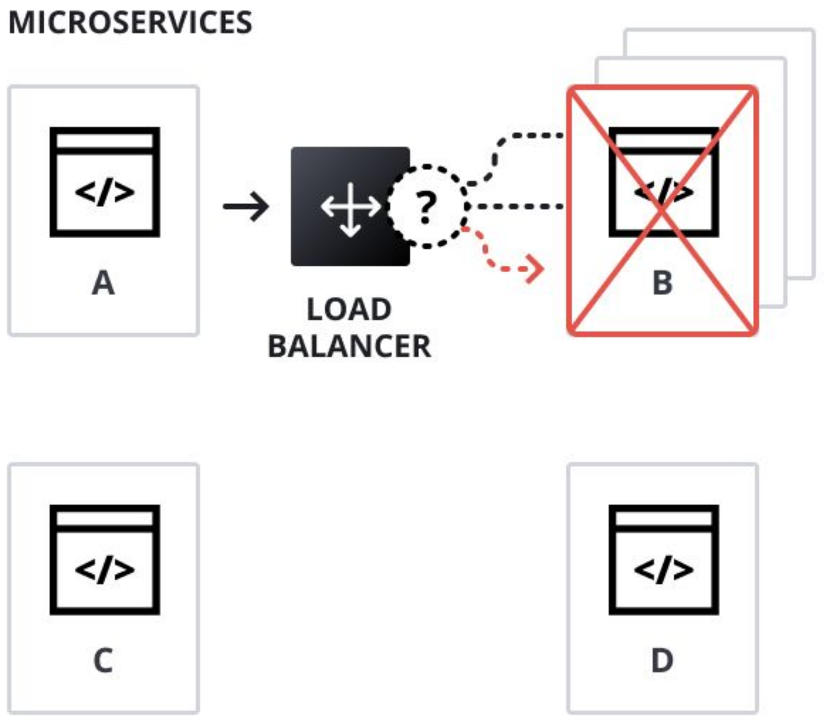
02

## Health Checks

## Health Checks

**Visibility into service health status**

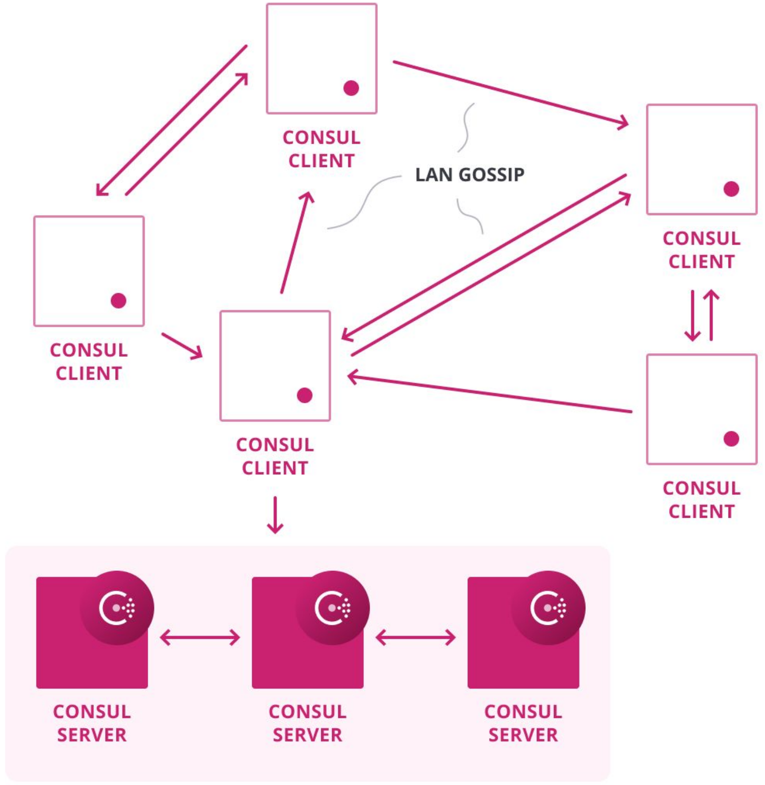
- Health checks are critical to prevent routing to services that are unhealthy
- Centralized approaches relying on heartbeating or periodic updates easily overload servers and lead to scaling issues



# Distributed Health Checking

Consul's Gossip Protocol provides a failure detector that scales massively

- Consul Agent runs health check locally
  - Only state changes get pushed to Consul servers
  - Prevents concentrating work on servers
  - Removes unhealthy nodes from service discovery layer
- Rich set of health checks beyond basic liveness



## Health Checks

***Application-level checks*** - associated with a specific service

***Node checks*** - monitor the health of the entire node

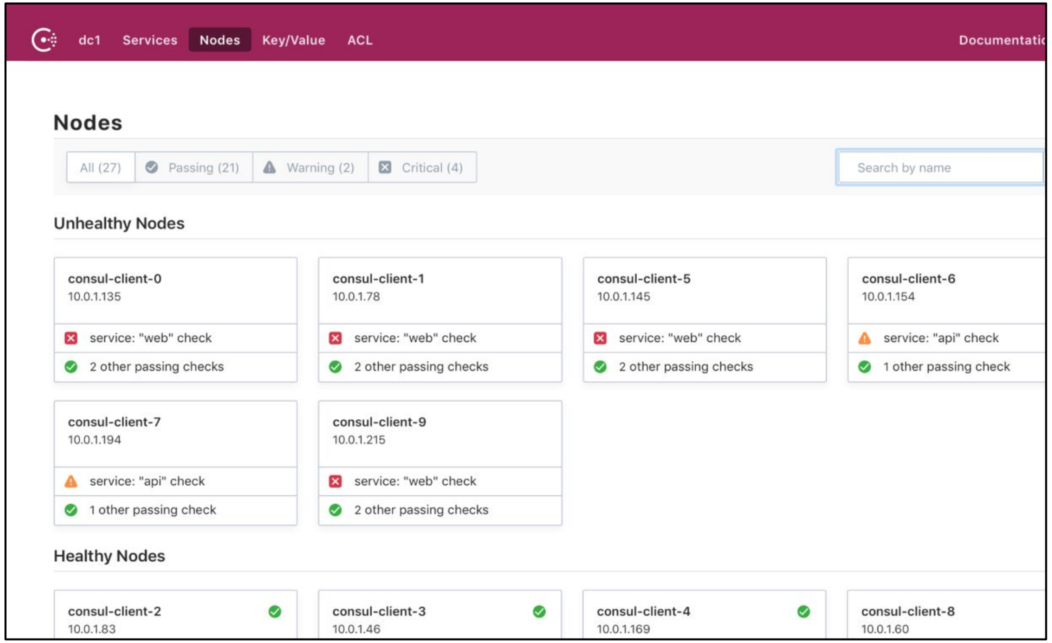
- **Defined via**
    - Configuration file
    - HTTP interface - persist with the node
- **Initially set to “critical”**

    Can be override by specifying the “status” field in the definition
- **Multiple check definitions**

    Multiple check definitions can be defined in a configuration file

# Health Check Types

- **Script** - Invokes an external application that performs the health check
- **HTTP** - “GET” request to specified URL, wait specified interval between requests
- **TCP** - connection attempt to IP/hostname & port, configurable interval between attempts, defaults to localhost if no hostname set
- **UDP** - send UDP datagrams to the specified IP/hostname & port, configurable interval between attempts



Nodes

All (27) | Passing (21) | Warning (2) | Critical (4)

Unhealthy Nodes

| | | | |
|---|---|---|---|
| consul-client-0 10.0.1.135 | consul-client-1 10.0.1.78 | consul-client-5 10.0.1.145 | consul-client-6 10.0.1.154 |
| service: "web" check | service: "web" check | service: "web" check | service: "api" check |
| 2 other passing checks | 2 other passing checks | 2 other passing checks | 1 other passing check |
| consul-client-7 10.0.1.194 | consul-client-9 10.0.1.215 | | |
| service: "api" check | service: "web" check | | |
| 1 other passing check | 2 other passing checks | | |

Healthy Nodes

| | | | |
|---|---|---|---|
| consul-client-2 10.0.1.83 | consul-client-3 10.0.1.46 | consul-client-4 10.0.1.169 | consul-client-8 10.0.1.60 |

# Health Check Types

- **Time to Live (TTL)** - passive checks that await updates from a service, if update not received before duration marks service “critical”, sometimes called “dead man’s switch”
- **Docker** - invoke an external application packed in a Docker Container
- **gRPC** - gRPC health checking protocol based, updates configured endpoint with configurable interval, can be TLS enabled
- **H2ping** - http2 based ping, assumed to be TLS by default
- **Alias** - check the health state of another node or service
- **Health checks for Consul on Kubernetes** - can sync the status of Kubernetes health probes of pods to Consul

# Health Check Definitions

- Multiple checks for a service can be defined in a single block
- Consul enables services to easily provide circuit breakers with custom scripts

JSON

```
{
  "check": {
    "id": "mem-util",
    "name": "Memory Utilization",
    "script": "/usr/local/bin/check_mem.py",
    "interval": "10s"
  }
}
```

HCL

```
check = {
 id = "web-app"
 name = "Web App Status"
 notes = "Web app does a curl internally every 10
seconds"
 ttl = "30s"
}
```

03

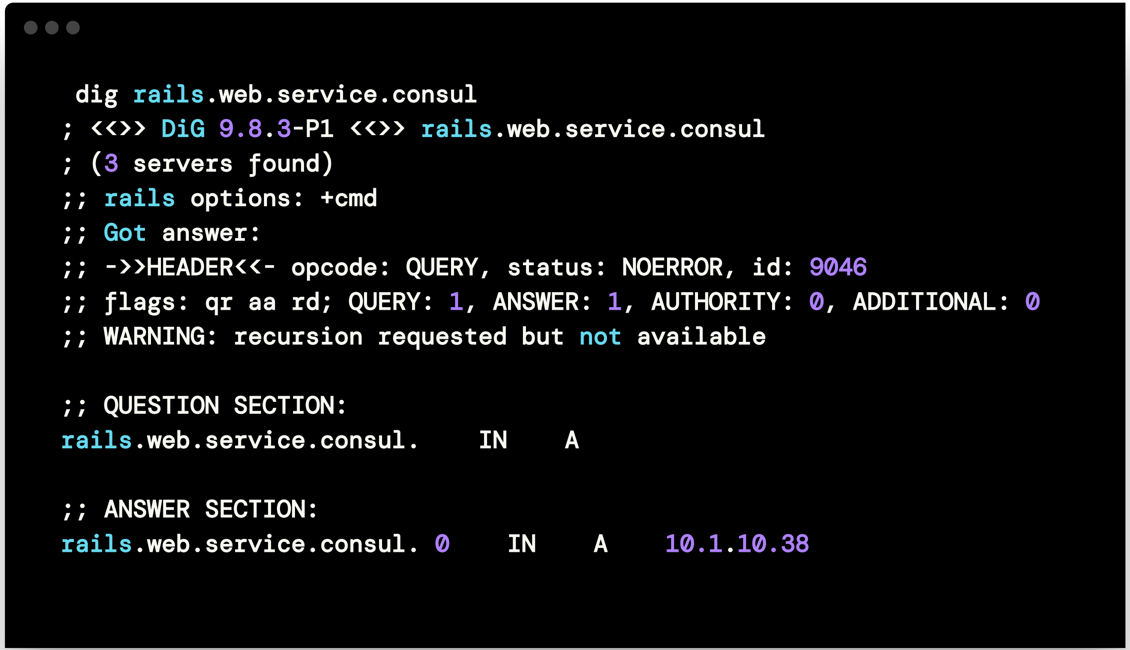
## Consul DNS

## Consul DNS

- One of the primary query interfaces for Consul
- Allows applications to use service discovery without any high-touch integration with Consul
- Hosts can use the DNS server directly via name lookups
- Supports both Service and Node lookups

# DNS Query Interface

- Commonly used to enable service discovery for legacy applications
- Leverage existing DNS deployments for service discovery
- Defaults to respond in the consul domain, is configurable for multiple domains



```
 dig rails.web.service.consul
; <<>> DiG 9.8.3-P1 <<>> rails.web.service.consul
; (3 servers found)
;; rails options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 9046
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
rails.web.service.consul.    IN    A

;; ANSWER SECTION:
rails.web.service.consul. 0    IN    A    10.1.10.38
```

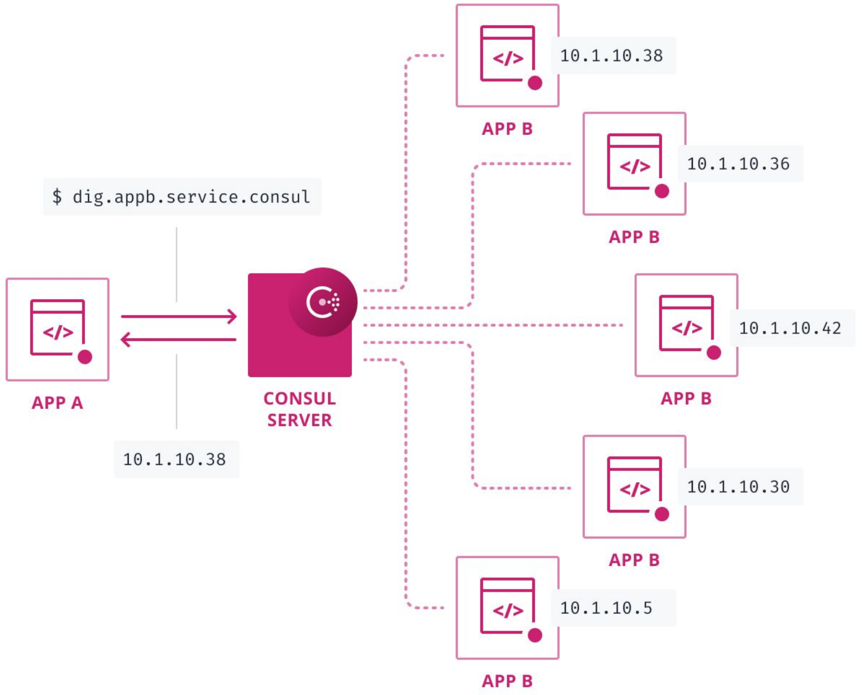
# DNS Query Interface

## Methods for using the Consul DNS interface

1. Custom DNS resolver library pointed at Consul
2. Set Consul as the DNS server for node(s) and use a recursive configuration so that non-Consul queries also resolve
3. Forward all queries for the "consul." domain to a Consul agent from the existing DNS server

# Load Balancing via Consul DNS

- Leverage Consul's zero-touch DNS interface
- Randomized Round-Robin load balancing
- Integrated with health checks, entries for services that fail health checks are automatically filtered out so traffic doesn't route to unhealthy hosts



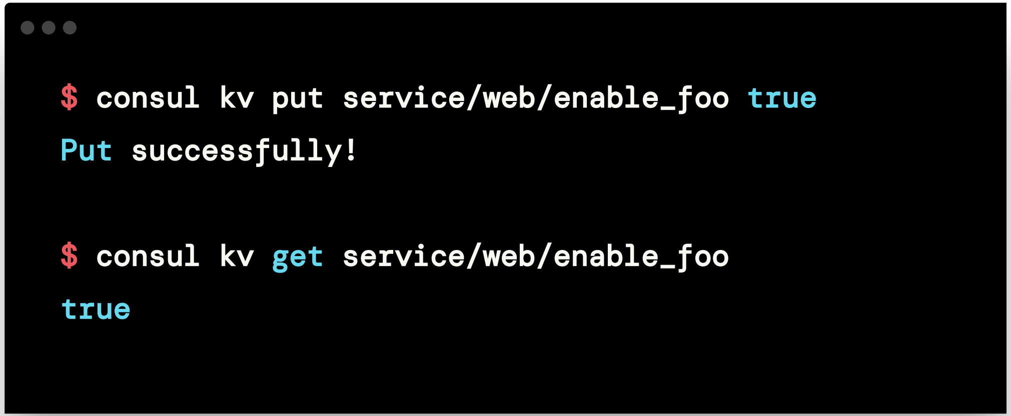
04

## Consul KV & Service Configuration

## Hierarchical Key Value Store

**Store and retrieve dynamic configuration, feature flagging, coordination and more metadata**

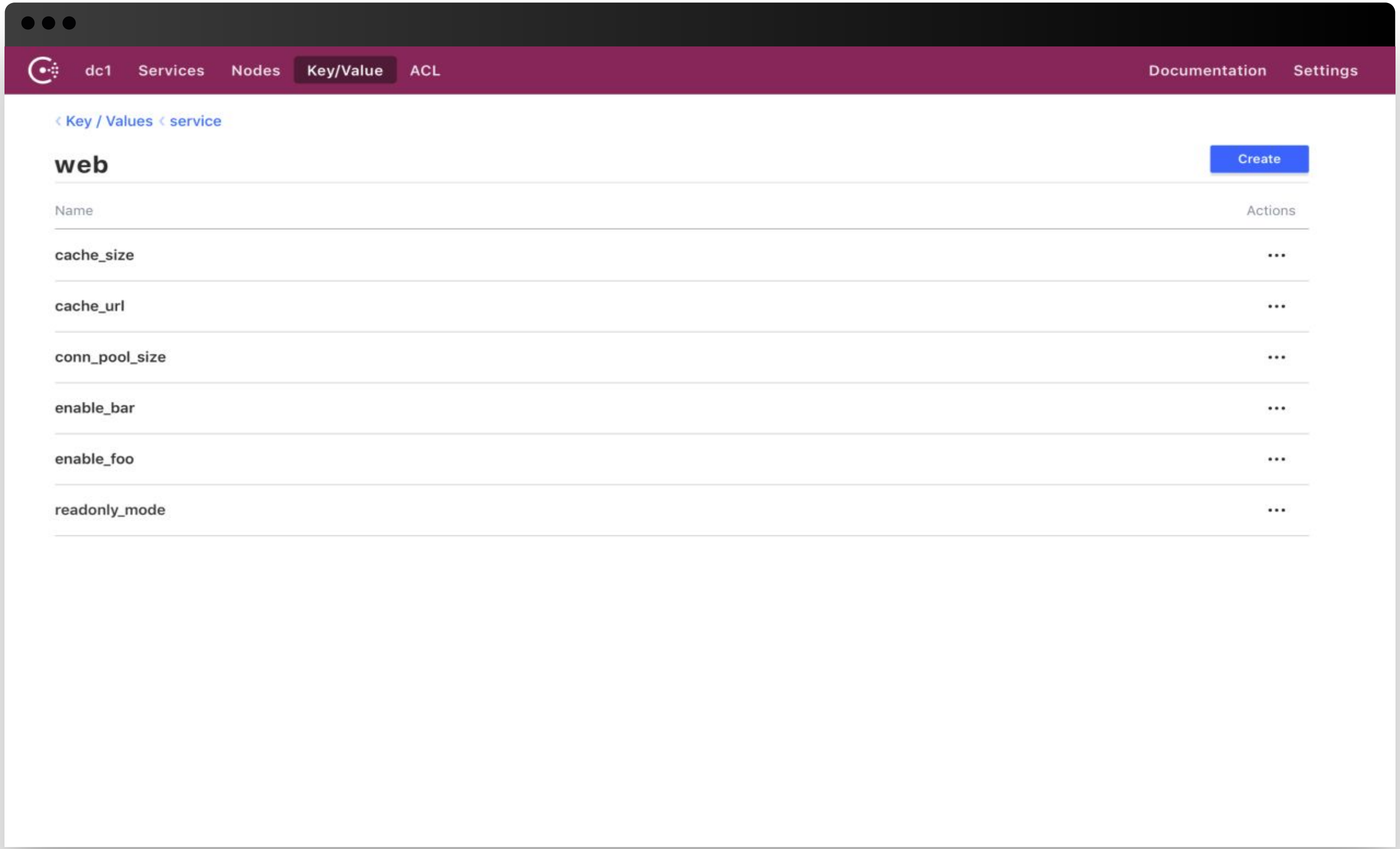
- Highly-available, globally accessible key-value store
- Folder-like architecture allows for easy organization
- ACLs to enforce policy and access
- Bulk export and import of key value pairs
- Accessible via HTTP API
- Can be used via the CLI or tools like curl
- Automated backup via snapshot agent



```
$ consul kv put service/web/enable_foo true
Put successfully!

$ consul kv get service/web/enable_foo
true
```

# K/V Store Web UI



‹ Key / Values ‹ service

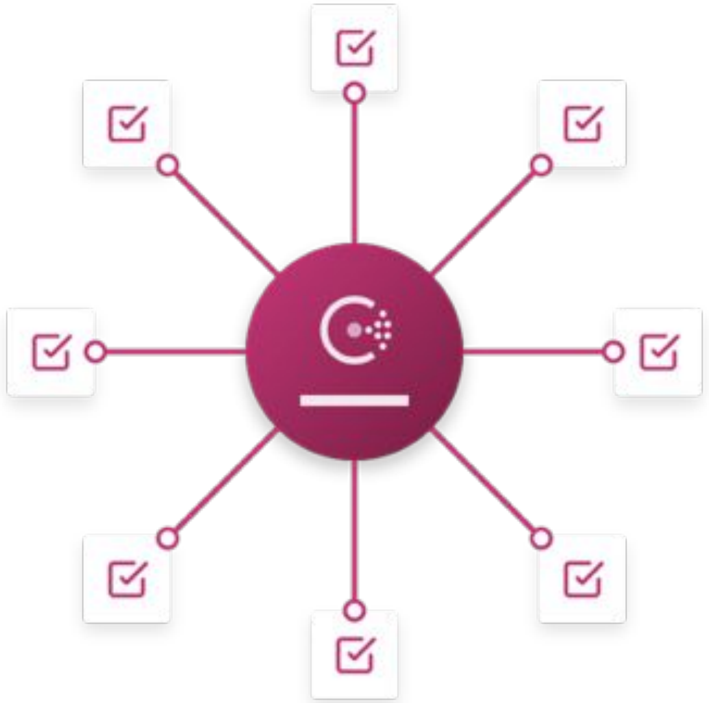
web

| Name | Actions |
|---|---|
| cache_size | ... |
| cache_url | ... |
| conn_pool_size | ... |
| enable_bar | ... |
| enable_foo | ... |
| readonly_mode | ... |

# Service Configuration

Dynamic configuration across distributed services in milliseconds

- **Improve Productivity** by avoiding manual updates to thousands of service instances
- **Reduce Risk** by pushing consistent configuration changes across all distributed services in real-time
- **Reduce Cost** by eliminating the need for config management tools for runtime configuration

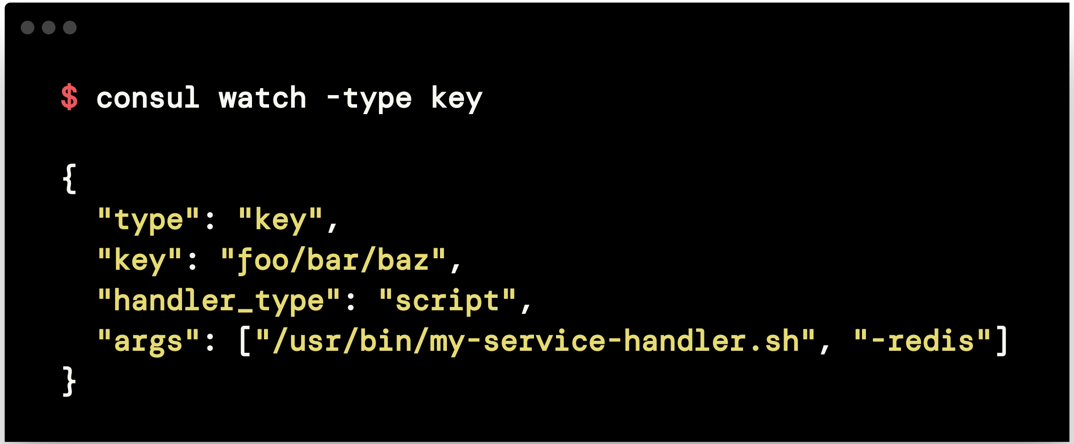


## Watches

### React to changes dynamically

Watches are the simplest way to react to changes using Consul

- Watch for changes in K/V, services, nodes, health checks, and events
- Invoke external handlers when a change is detected. The handler can be any executable, letting operators customize behavior



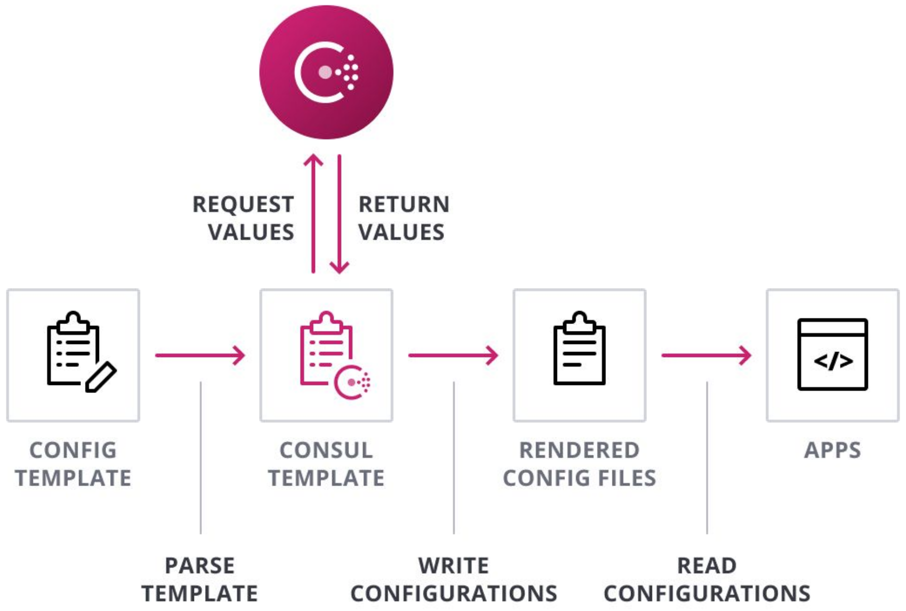
```
$ consul watch -type key

{
  "type": "key",
  "key": "foo/bar/baz",
  "handler_type": "script",
  "args": ["/usr/bin/my-service-handler.sh", "-redis"]
}
```

## Consul Template

### Link 3rd party config files to Consul K/V

- Standalone application that populates values from Consul and dynamically renders updates to third party configurations
- Automatically triggers a reload of third party tools when the template is updated

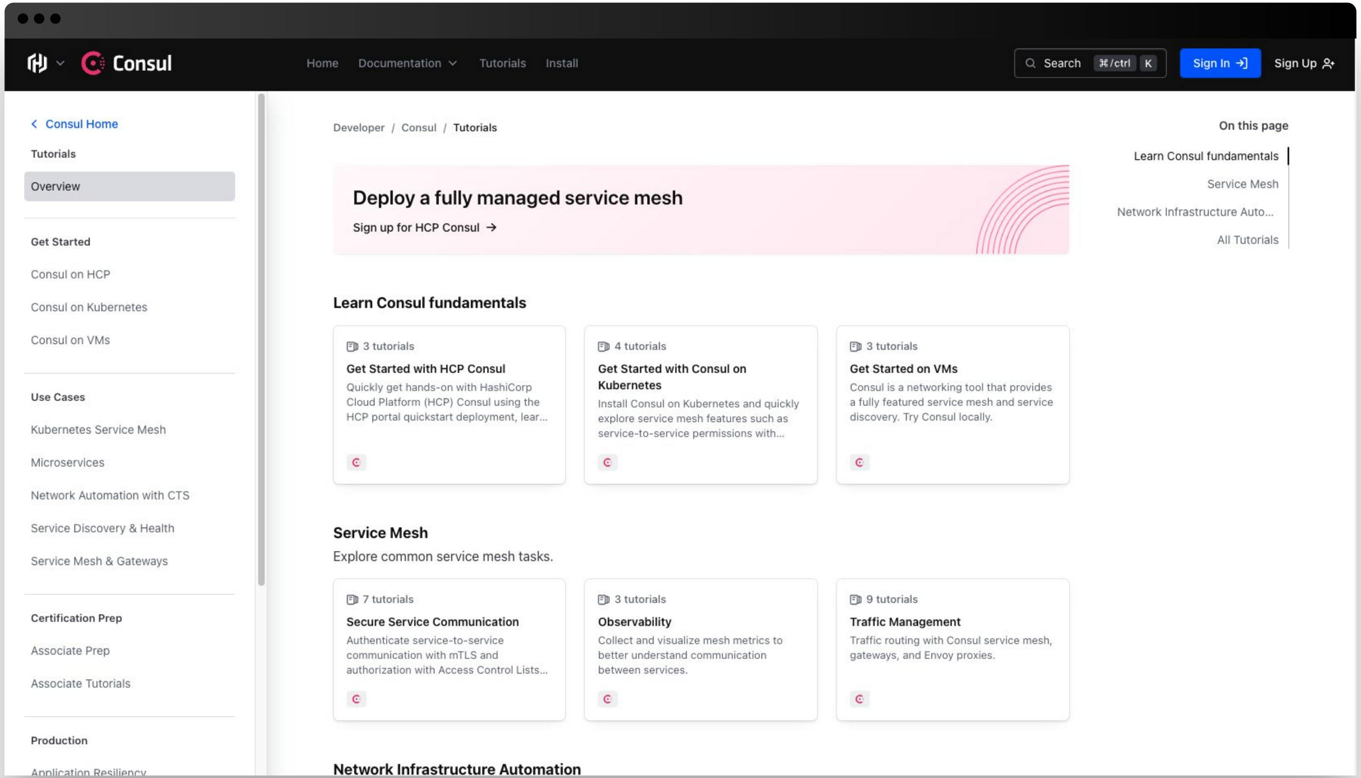


## Next Steps

# Tutorials

https://developer.hashicorp.com/consul/tutorials

Step-by-step guides to accelerate deployment of Consul



< Consul Home

Tutorials

Overview

Get Started

Consul on HCP

Consul on Kubernetes

Consul on VMs

Use Cases

Kubernetes Service Mesh

Microservices

Network Automation with CTS

Service Discovery & Health

Service Mesh & Gateways

Certification Prep

Associate Prep

Associate Tutorials

Production

Application Resiliency

Developer / Consul / Tutorials

**Deploy a fully managed service mesh**

Sign up for HCP Consul →

On this page

Learn Consul fundamentals

Service Mesh

Network Infrastructure Auto...

All Tutorials

**Learn Consul fundamentals**

3 tutorials
**Get Started with HCP Consul**
Quickly get hands-on with HashiCorp Cloud Platform (HCP) Consul using the HCP portal quickstart deployment, lear...

4 tutorials
**Get Started with Consul on Kubernetes**
Install Consul on Kubernetes and quickly explore service mesh features such as service-to-service permissions with...

3 tutorials
**Get Started on VMs**
Consul is a networking tool that provides a fully featured service mesh and service discovery. Try Consul locally.

**Service Mesh**

Explore common service mesh tasks.

7 tutorials
**Secure Service Communication**
Authenticate service-to-service communication with mTLS and authorization with Access Control Lists...

3 tutorials
**Observability**
Collect and visualize mesh metrics to better understand communication between services.

9 tutorials
**Traffic Management**
Traffic routing with Consul service mesh, gateways, and Envoy proxies.

**Network Infrastructure Automation**

# Additional Resources

- Consul Service Registration Tutorial (VMs)
- Register Services on Kubernetes
- Register and Discover Services within Namespaces
- Service Definition Documentation
- Find Services with Consul DNS
- Consul DNS Caching
- Define Health Checks
- Register services and health checks
- Consul KV Learn Guide
- Consul Template & Load Balancers

# Need Additional Help?

## Customer Success

Contact our Customer Success Management team with any questions. We will help coordinate the right resources for you to get your questions answered.

customer.success@hashicorp.com

## Technical Support

Something not working quite right? Engage with HashiCorp Technical Support by opening a ticket for your issue at:

support.hashicorp.com

## Discuss

Engage with the HashiCorp Cloud community including HashiCorp Architects and Engineers

discuss.hashicorp.com

## Upcoming Webinars

**Consul Deployment & Operations**

Take a deep dive into deployment and operational best practices including: Consul Autopilot, the Consul Agent and ACLs, Backup, Disaster Recovery, and Telemetry and Monitoring

**Office Hours**

An open forum with Consul Subject Matter Experts to answer questions that have arisen during the program and your deployment

**Advanced Concepts**

A detailed examination of Consul Federation, Namespaces & Admin Partitions, content also cover cluster operations and runbooks along with managing geographic failover and prepared queries

## Action Items

- If not done, please share to customer.success@hashicorp.com
  - Authorized technical contacts for support
  - Stakeholders contact information (name and email addresses)
- Email raquel.peterson@hashicorp.com summarizing where you are at with your Consul deployment & implementation
- Deploy first cluster(s) and start onboarding first use case

## Q&A

# Thank you

customer.success@hashicorp.com

www.hashicorp.com/customer-success