HashiCorp Consul

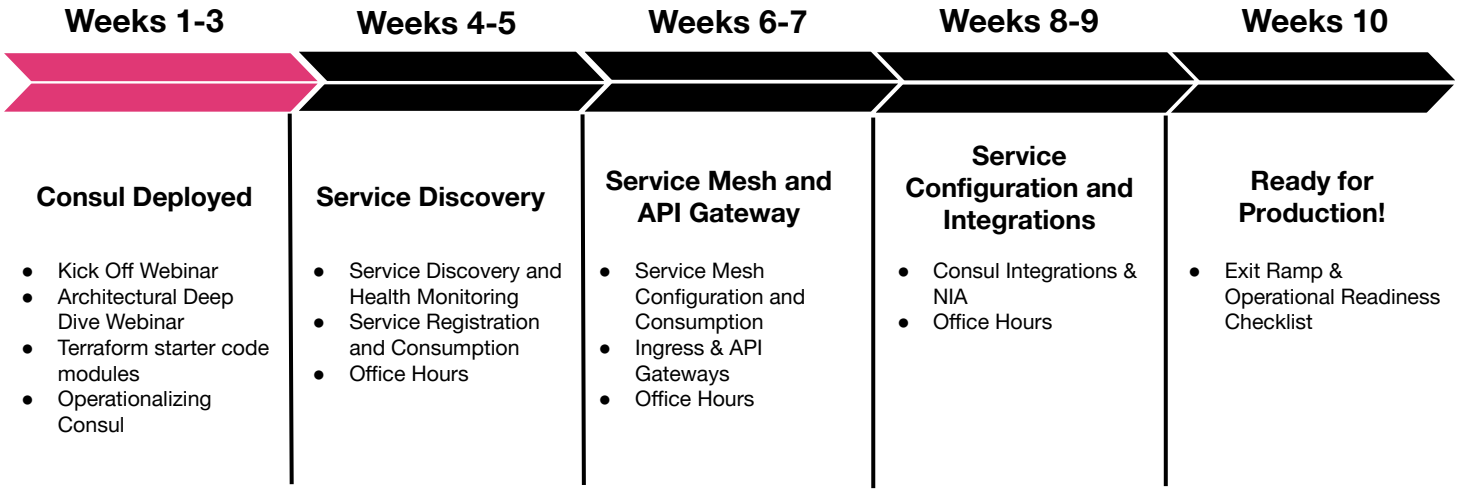
Operationalizing Consul

July 2022

Agenda

1. Autopilot
2. DR Operations & Snapshot Agent
3. Telemetry & Monitoring
4. Federation
5. Namespaces & Admin Partitions
6. ACL Configuration

Consul Enterprise Path to Production



01

Consul Autopilot

Consul Autopilot

- Designed for automatic, operator-friendly management of Consul servers.
- Functionality includes:
  - Dead server cleanup
  - Server Stabilization
  - Redundancy Zone Tags
  - Automated upgrades
- Enabled by default in Consul Enterprise

Autopilot Default Values

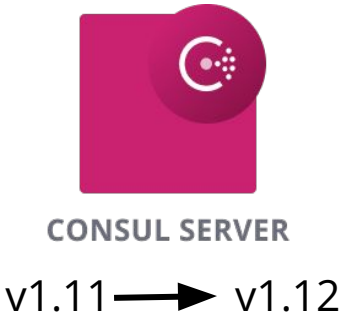
TERMINAL

```
$ consul operator autopilot get-config

  CleanupDeadServers = true
  LastContactThreshold = 200ms
  MaxTrailingLogs = 250
  MinQuorum = 0
  ServerStabilizationTime = 10s
  RedundancyZoneTag = ""
  DisableUpgradeMigration = false
  UpgradeVersionTag = ""
```

Consul Upgrade Patterns

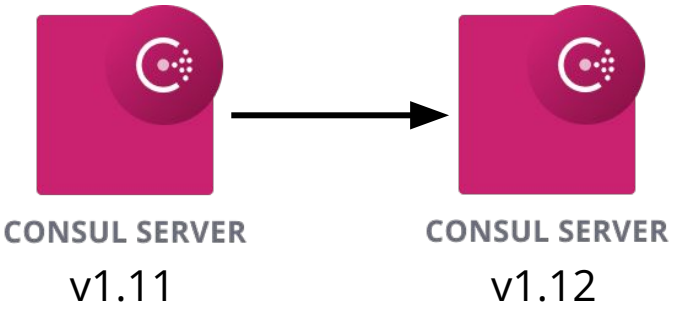
Rolling Restart

1. Replace the old binary
2. Rolling restart of the server
3. Check health
   1. `consul version`
   2. `consul members`
   3. `consul operator raft peer-list`
4. Repeat

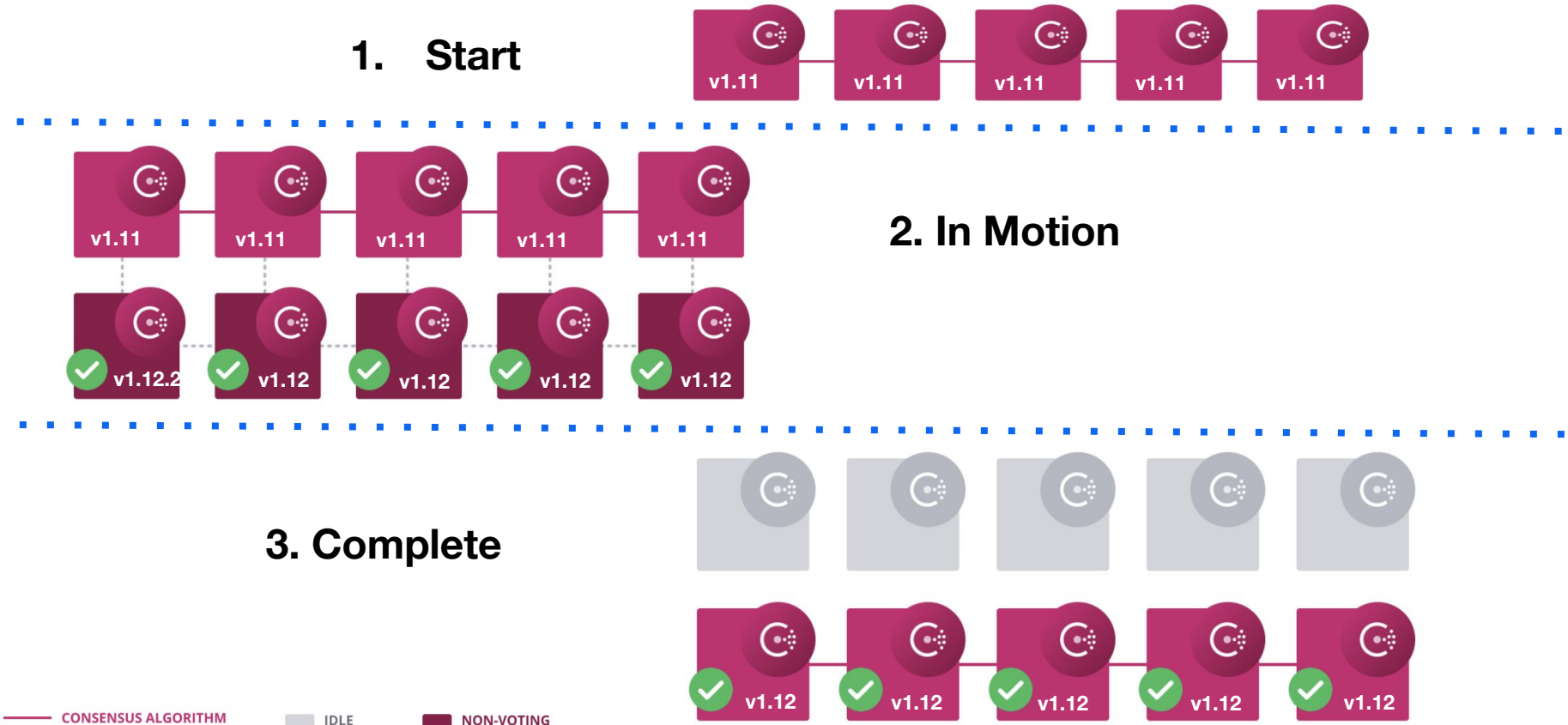


Add & Remove Servers

1. Add a new server to the existing peer set
2. Gracefully remove one of the followers
3. Repeat



Automated Upgrades with Autopilot



02

DR Operations

Consul Backups

- Consul snapshots are the primary backup and DR solution for Consul
- Consul snapshots are atomic, point-in-time, datacenter specific copies of Consul state.
- Snapshots create a gzipped tar archive that include:
  - Key/Value Store Entries
  - Service Catalog Registrations
  - Prepared queries
  - Intentions
  - ACLs
  - Service Catalog
  - Sessions
  - Namespaces
- By default, snapshots are performed from the cluster leader
- Follower nodes are capable of taking snapshots but “stale” flag is required

Backing up Consul on Kubernetes

- Consul on Kubernetes requires backing up 4 essential secrets:
  - The last active Consul ACL bootstrap token
  - The last active Consul CA cert
  - The last active Consul CA key
  - The last active gossip encryption key

- Without these 4 secrets you **cannot recover** from a disaster
- These secrets need to be secure and stored **outside** the Kubernetes secrets engine
- Update and take a new Consul snapshot whenever secrets are rotated
- Automate the rotation and backup procedure(s)

Snapshot Backup Methods

- Consul Snapshot Agent allows for scheduled automatic process
- Automated Snapshots using the Agent should be enabled to meet RPO
- Manual snapshots can be done via CLI or API commands
- Snapshots require a valid ACL token with write permissions when ACLs are enabled
- Snapshots should be taken prior to upgrades or major configuration changes

Consul Snapshots

TERMINAL

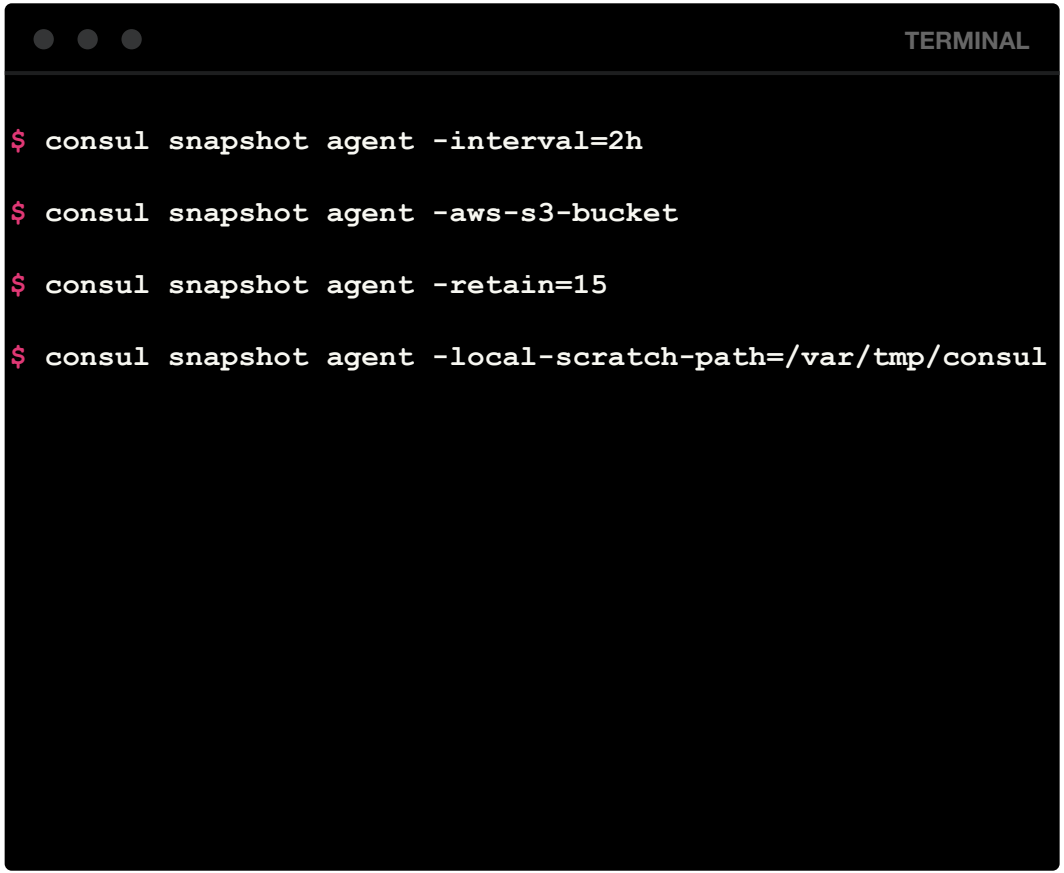
```
$ consul snapshot save backup.snap
Saved and verified snapshot to index 1176

$ consul snapshot inspect backup.snap
ID           2-1182-1542056499724
Size         4115
Index        1182
Term         2
Version      1

$ consul snapshot save -stale backup.snap
Saved and verified snapshot to index 2276

$ export CONSUL_HTTP_TOKEN=<your ACL token>
$ consul snapshot save -stale -ca-file=</path> backup.snap
Saved and verified snapshot to index 2287
```

Snapshot Agent

Enterprise Feature for automatic snapshots

- Manages leader election for HA and failover if leader becomes unavailable
- Includes automated backup retention and rotation
- Backups can be saved to local files system, AWS S3, Azure Blob, or GCP Cloud Storage

Snapshot Agent Documentation



Snapshot Restore

- Is a turbulent process, all communication with Consul halts until snapshot is restored.
- Is not selective to a feature or data element, it is an all-or-nothing process
- Only needs to be run once, on the leader node. Raft consensus ensures all servers restore the same state
- Before performing make sure cluster is stable and has a leader, run **"`consul operator raft list-peers`"** and review logs and telemetry for anomalies
- Process is a low-level Raft operation and is not designed to handle server failures when process is running

```
$ consul snapshot restore backup.snap
Restored snapshot
```

Snapshot Restore Documentation

03

Telemetry & Monitoring

Monitoring Consul

Ensuring the state and health of your Consul datacenter(s) requires a multi-layered approach

- Consul CLI and API for initial and manual use
- Visualize metrics for real-time monitoring
- Collect and store metrics for comparison over time

Methods for Collecting Metrics

- Send SIGUSR1 to the Consul process
  - Via command line `"kill -USR1 <process_id>"`
  - Collects one-time dump of current metric values. Sends output to the system logs
  - If running `"consul monitor"` command in terminal, metrics will be in output
- API GET Request
  - Curl can be used to collect metrics via HTTP API
  - Example command `"curl http://127.0.0.1:8500/v1/agent/metrics"`
  - Can be added to a script for monitoring agents like Prometheus that support HTTP scraping
  - Production usage should use an ACL token and enable TLS
- Enable Telemetry (recommended solution)
  - Sends telemetry to a remote monitoring solution monitor to gather data over time and spot trends
  - Metrics are aggregated on a 10s interval and retained for 1 minute
  - Supported telemetry agents:
    - Circonus
    - DataDog (via dogstatsd)
    - StatsD (via statsd, statsite, telegraf, etc.)

Example DataDog Configuration

TERMINAL

```
$ cat server.hcl

telemetry {
  dogstatsd_addr = "localhost:8125"
  disable_hostname = true
 }

$ consul reload
```

Monitoring Strategy

- **Consul Datacenter Health** - information about the Consul datacenter
  - Transaction timing
  - Leadership changes
  - Autopilot status
  - Garbage collection
- **Server Health** - information about each server node in the cluster
  - File handles
  - CPU usage
  - Network activity
  - Disk activity
  - Memory usage
- Establish a baseline from a healthy cluster so you have comparison points

Telemetry & Monitoring - Key Metrics

Some key metrics emitted to understand cluster health at a glance, the Consul team maintains a Grafana Dashboard to visualize the data

Transaction timing	
consul.kvs.apply	Measures the time it takes to complete an update to the KV store.
consul.txn.apply	Measures the time spent applying a transaction operation.
consul.raft.apply	Counts the number of Raft transactions applied during the interval. This metric is only reported on the leader.
consul.raft.commitTime	Measures the time it takes to commit a new entry to the Raft log on the leader.

Consul metrics list

Leadership changes	
consul.raft.leader.lastContact	Measures the time since the leader was last able to contact the follower nodes when checking its leader lease.
consul.raft.state.candidate	Increments whenever a Consul server starts an election.
consul.raft.state.leader	Increments whenever a Consul server becomes a leader.
Autopilot	
consul.autopilot.healthy	Tracks the overall health of the local server cluster. If all servers are considered healthy by Autopilot, this will be set to 1. If any are unhealthy, this will be 0.
Memory Usage	
consul.runtime.alloc_bytes	Measures the number of bytes allocated by the Consul process.
consul.runtime.sys_bytes	Measures the total number of bytes of memory obtained from the OS.
License Expiration	
consul.system.licenseExpiration	Number of hours until the Consul Enterprise license will expire.

Garbage Collection	
consul.runtime.total_gc_pause_ns	Number of nanoseconds consumed by stop-the-world garbage collection (GC) pauses since Consul started.
Network activity - RPC Count	
consul.client.rpc	Increments whenever a Consul agent in client mode makes an RPC request to a Consul server
consul.client.rpc.exceeded	Increments whenever a Consul agent in client mode makes an RPC request to a Consul server gets rate limited by that agent's limits configuration.
consul.client.rpc.failed	Increments whenever a Consul agent in client mode makes an RPC request to a Consul server and fails.
Raft Replication Capacity Issues	
consul.raft.fsm.lastRestoreDuration	Measures the time taken to restore the FSM from a snapshot on an agent restart or from the leader calling installSnapshot
consul.raft.leader.oldestLogAge	The number of milliseconds since the oldest log in the leader's log store was written. In normal usage this gauge value will grow linearly over time until a snapshot completes on the leader and the log is truncated.
consul.raft.rpc.installSnapshot	Measures the time taken to process the installSnapshot RPC call. This metric should only be seen on agents which are currently in the follower state.

04

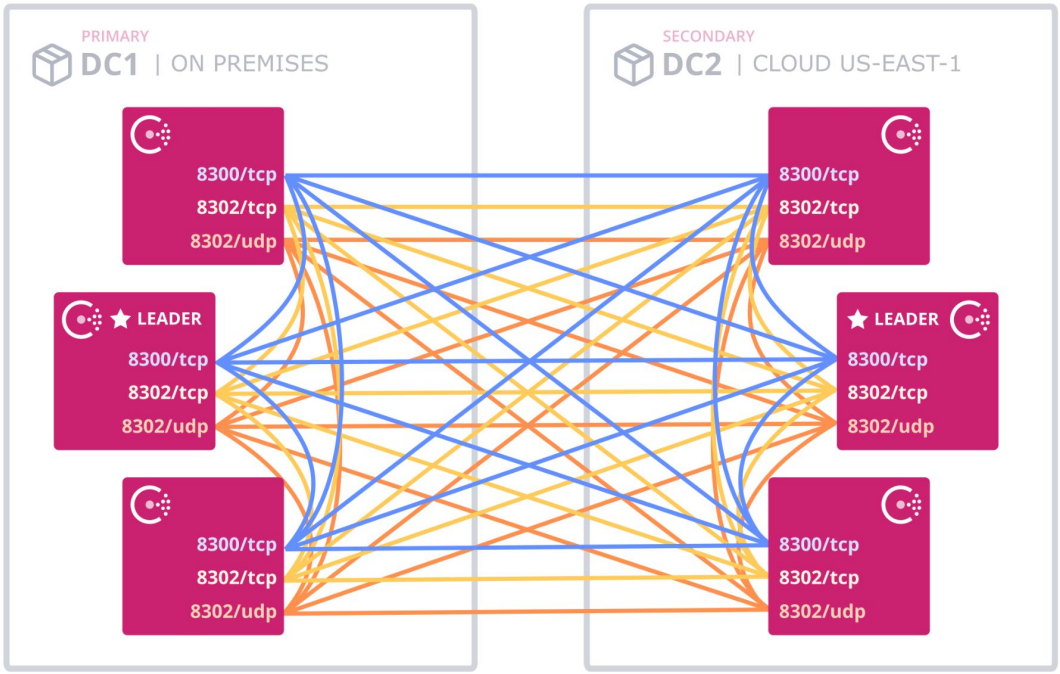
Federation

Consul Federation

- Joins two or more Consul datacenters into a WAN cluster
- Consul servers in federated clusters can communicate with each other
- Federation allows:
  - Services on all clusters can make calls to each other through Consul service mesh
  - Intentions can be used to enforce rules about which services can communicate across all clusters
  - L7 routing rules can enable multi-cluster failover and traffic splitting
  - Access any datacenter via a drop-down in the UI or a CLI flag
  - Consistent ACL policies across all federated clusters
- Consul has two mechanisms for WAN Federation
  - Traditional WAN Federation
  - WAN Federation via Mesh Gateways (Consul 1.8.0+)

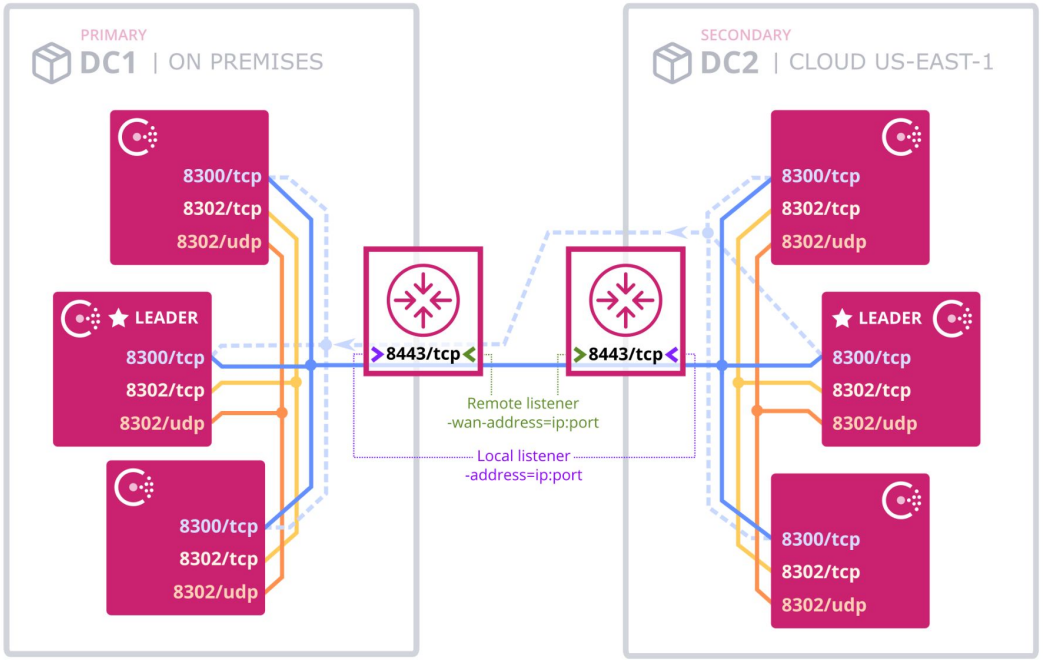
Traditional WAN Federation

- All Consul servers must be exposed on the WAN
- Does not support overlapping IP addresses across federation.
- Difficult to implement with Kubernetes, requires proxying UDP and TCP
- Enables Cross datacenter service discovery, providing the ability to setup multi-dc failover, blue/green deployments, and canary testing.



WAN Federation over Mesh Gateways

- All Consul traffic passes through Mesh Gateway - including WAN Gossip
- Requires that mesh gateways are exposed with routable addresses, Instead of Consul servers.
- Secure service communication between clouds without VPN or cloud specific services
- Mesh gateway must be configured for all clusters in the federation



Replication

- In general, data is not replicated between federated Consul datacenters.
- Replicated data includes:
  - ACL policies
  - ACL global tokens
  - Service Mesh Configuration Entries
- Consul minimizes the data that is replicated by design

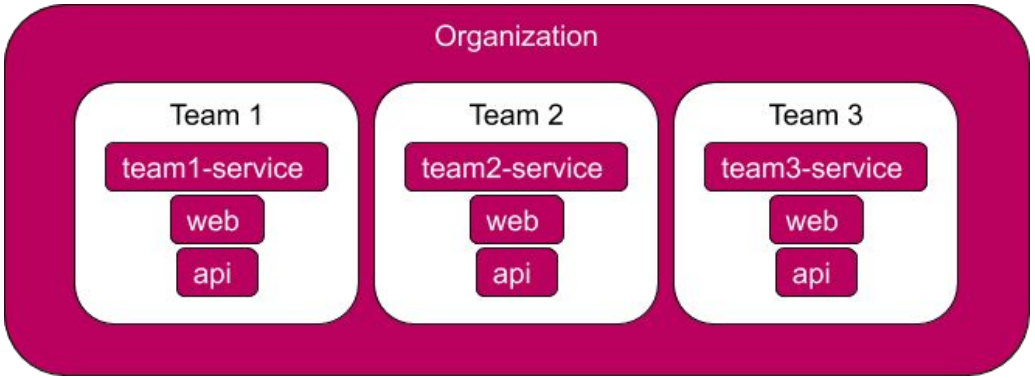
05

Namespaces & Admin Partitions

Consul Namespaces

Provide separation and segmentation of Consul to allow teams to share Consul datacenters without conflict

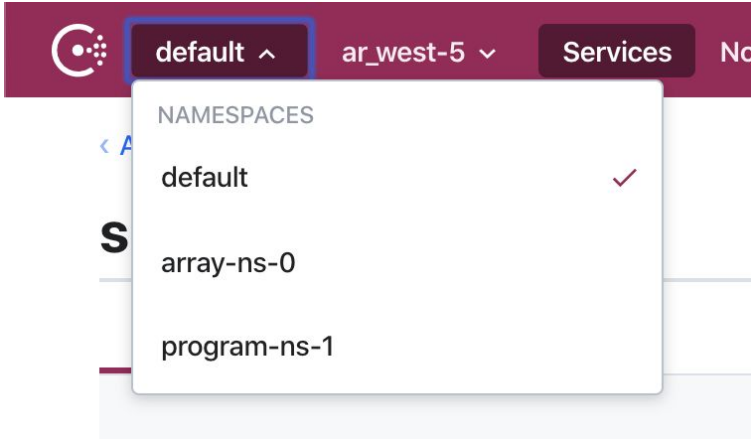
- Creates parallel instances of the service catalog and Key/Value Store
- Allow service deployment without coordination between teams of names of services or K/V paths
- Allows for self service via delegation of admin privileges



Consul Namespaces

Simplify team coordination and delegate authority

- Help reduce operational challenges.
- Provide separation for teams within an organization
- Central operators can delegate authority without worrying about impacts to services that belong to other teams of lines of business
- Rules like “You can set the intention for *any service in your namespace*” or “You can only query services which exist *in your namespace*” are possible



Namespace Configuration

Namespaces must be managed via API or the Consul CLI. API management uses JSON while the CLI will parse either JSON or HCL

JSON TERMINAL

```
{
  "Name": "team-1",
  "Description": "Namespace for Team 1",
  "ACLs": {
    "PolicyDefaults": [
      {
        "ID": "77117cf6-d976-79b0-d63b-5a36ac69c8f1"
      },
      {
        "Name": "node-read"
      }
    ],
    "RoleDefaults": [
      {
        "ID": "69748856-ae69-d620-3ec4-07844b3c6be7"
      },
      {
        "Name": "ns-team-2-read"
      }
    ]
  },
  "Meta": {
    "foo": "bar"
  }
}
```

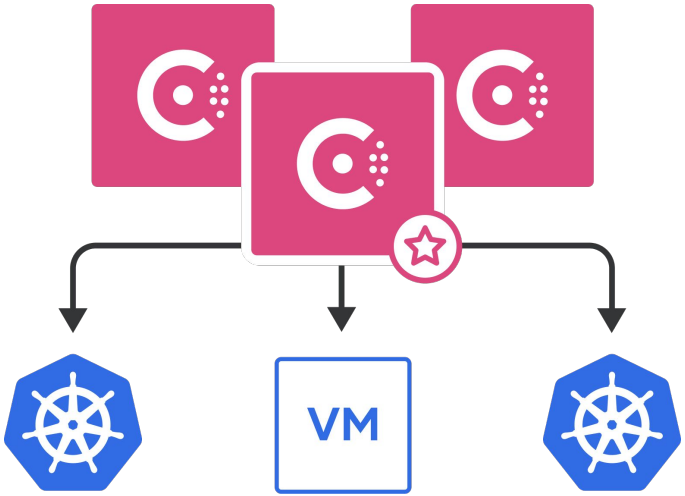
Admin Partitions

- Designed to provide multi-tenancy on a single Consul Datacenter
- Admin partitions exist a level above namespaces in the identity hierarchy, each admin partition contains a default namespace
- Contain one or more namespaces and allow multiple independent tenants to share a Consul server cluster
- Enable operators to define administrative and communication boundaries between services managed by separate teams or belonging to separate stakeholders
- Can be used to segment production and non-production services within a Consul deployment
- Known limitations
    - Only the default admin partition is supported when federating multiple Consul datacenters in a WAN
    - Admin partitions have no theoretical limit, HashiCorp has planned large-scale testing to identify a recommended max

Before Admin Partitions

Single Consul DC - Multiple Client Cluster

- Reduces management/cost **BUT:**
- No overlapping IPs (ex: Kube Pods)
- Some resources are global (default-proxy)
- Teams no longer autonomous
- Namespaces help but still limited



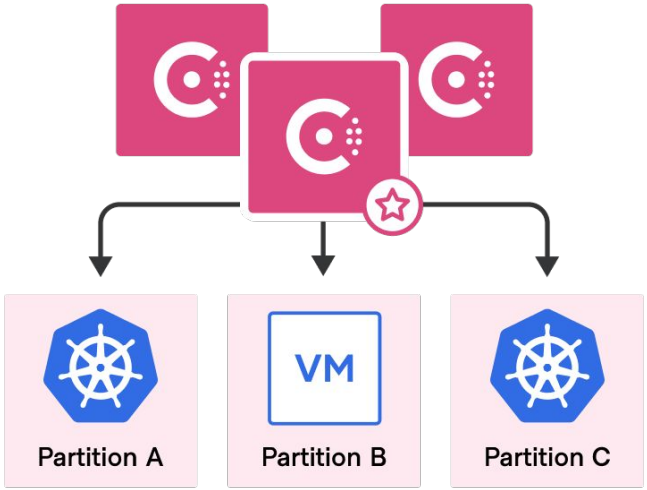
With Admin Partitions

Central operations team manages single Consul DC

- Reduced Management
- Reduce Cost
- Governance/Compliance

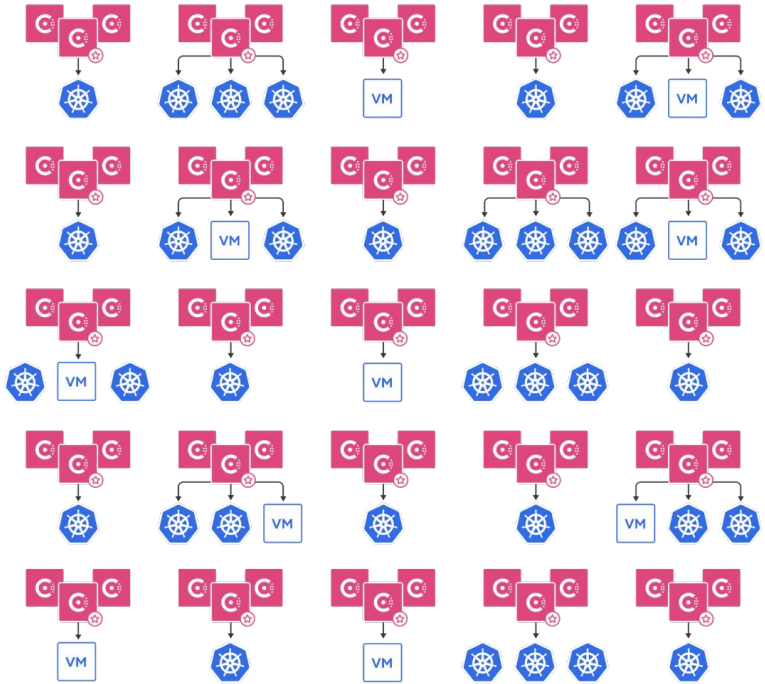
Allows autonomy between teams

- Allows overlapping IPs (ex: Kube Pods)
- More resources scoped inside Partitions (node, proxy-defaults, mesh config entries, namespaces)

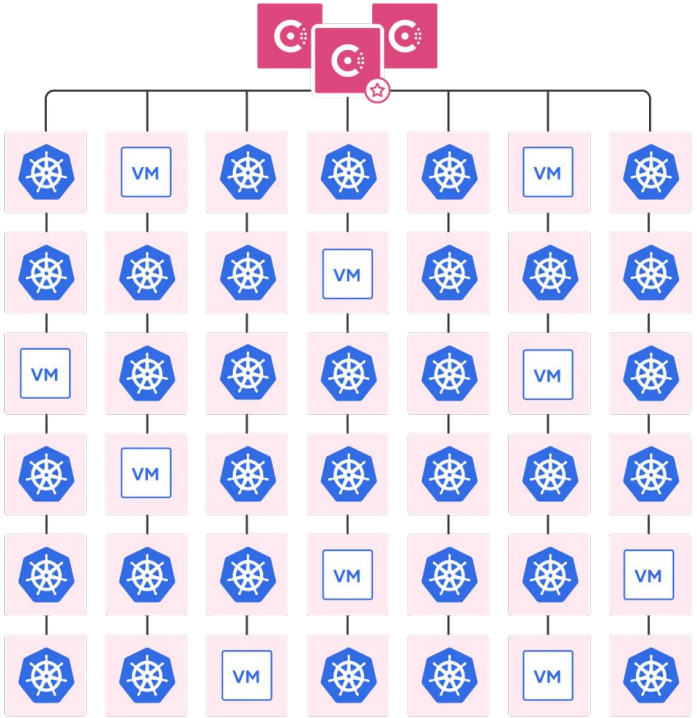


At Scale

Before Admin Partitions



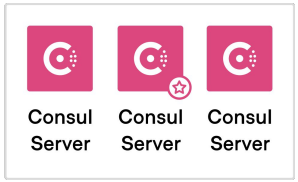
After Admin Partitions



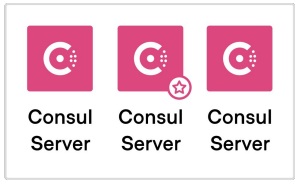
Admin Partition vs Namespaces

	Namespaces	Admin Partitions
Allow overlapping network between services (example Kubernetes pod IPs)	No	Yes
Support multiple proxy-default config entries per Consul datacenter	No	Yes, 1 per partition
Support multiple mesh config entries per Consul datacenter	No	Yes, 1 per partition
Datacenter scoped config entries between federated Consul datacenters	No	Yes*
Datacenter scoped ACLs between federated Consul datacenters	No	Yes*

How to Deploy Consul with Admin Partitions

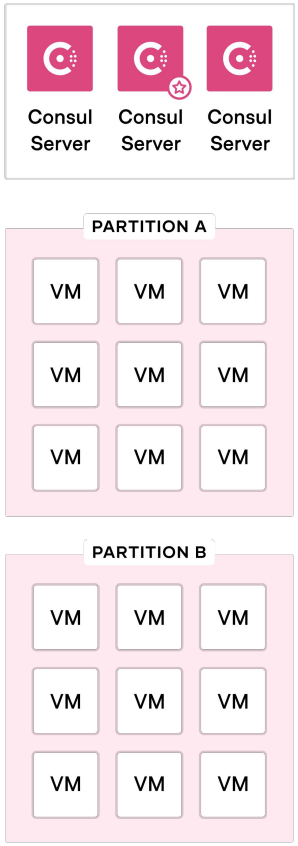


1 Deploy Consul servers (VM or Kubernetes):



2 Create partitions

```
$consul admin-partition create -name partition-A
$consul admin-partition create -name partition-B
```



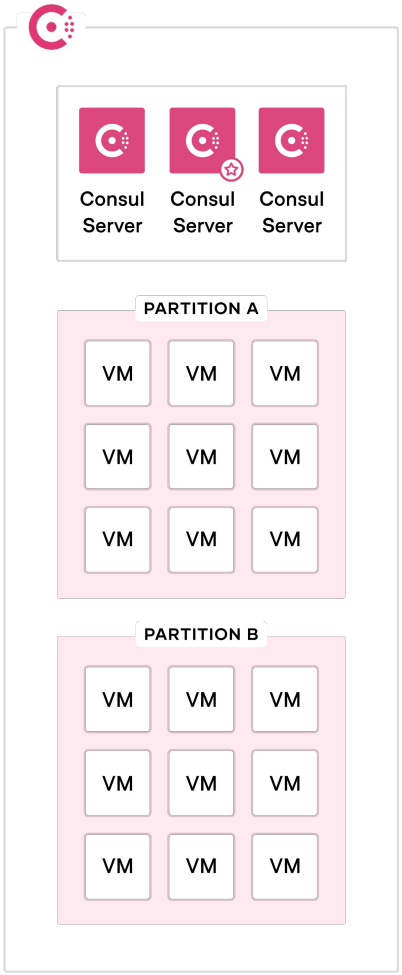
3 Add `partition` parameter to .hcl config file to all client VMs with desired partition name

```
{
  "node_name": "consul-client-X",
  "datacenter": "dc1",
  "partition": "partition-A",
  "data_dir": "/consul/data",
 ...
}
```

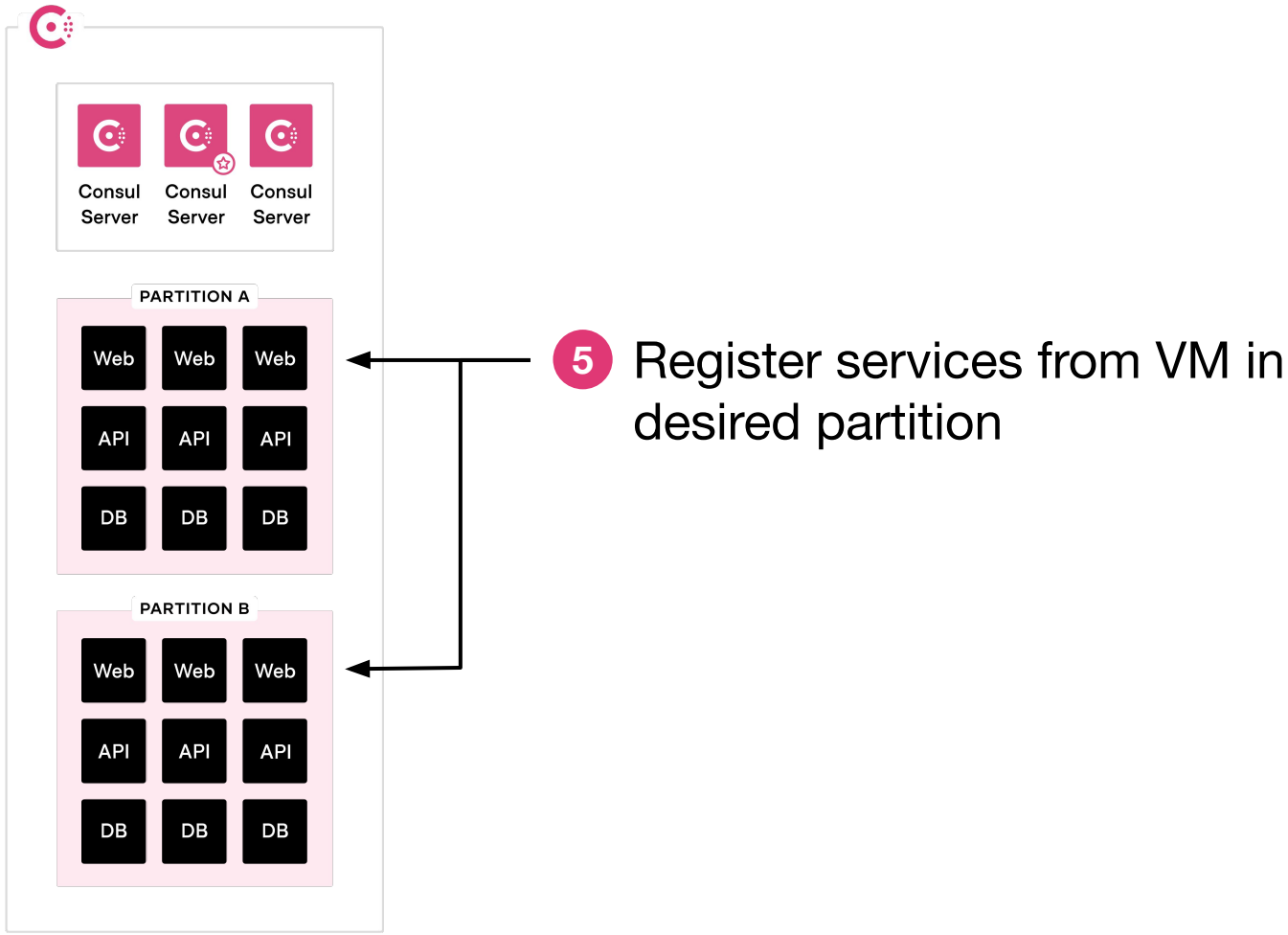
consul-A.hcl file

```
{
  "node_name": "consul-client-X",
  "datacenter": "dc1",
  "partition": "partition-B",
  "data_dir": "/consul/data",
 ...
}
```

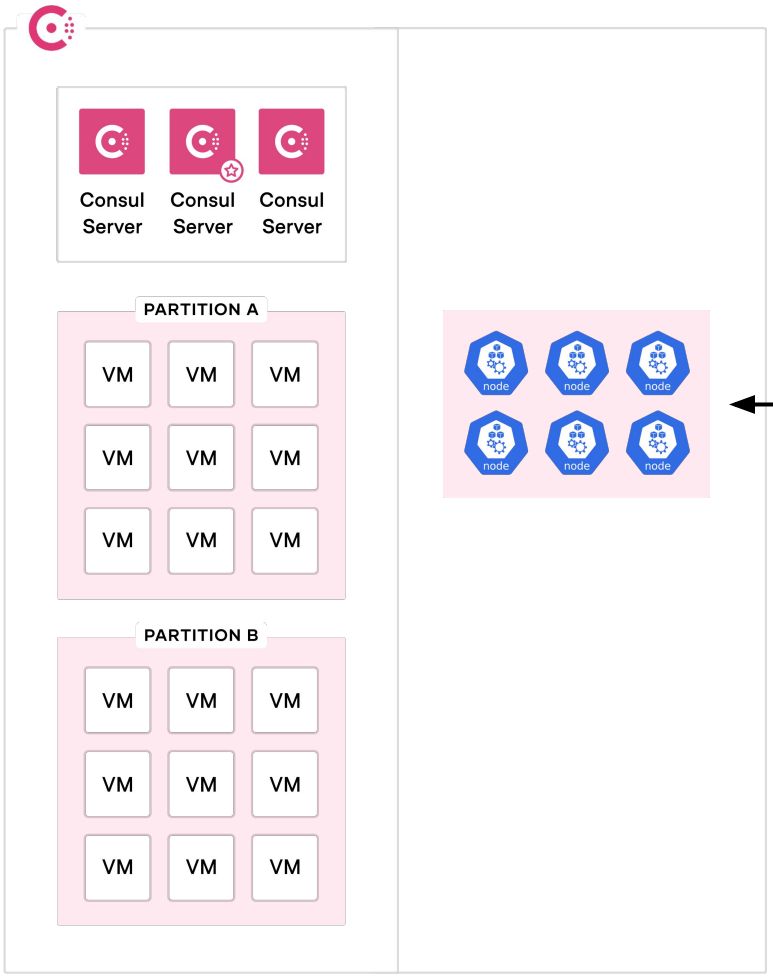
consul-B.hcl file



4 Join (or reload) Consul client to Consul datacenter



5 Register services from VM in desired partition




```
global:
  enabled: false
  enableConsulNamespaces: true
  image: hashicorp/consul-enterprise:1.11.0-ent-alpha
  adminPartitions:
    enabled: true
    name: "partition-C"
server:
  enterpriseLicense:
    secretName: license
    secretKey: key
externalServers:
  enabled: true
  hosts: ["<partition-service-IP-or-servercluster-IP>"]
  tlsServerName: server.dc1.consul
client:
  enabled: true
  exposeGossipPorts: true
  join: ["<partition-service-IP-or-servercluster-IP>"]
```

helm-C.hcl file

06

ACL Configuration

Consul ACLs

- ACLs authenticate requests and authorize access to resources
- Consul's ACL system is disabled by default
- ACLs control access to the Consul UI, API, and CLI
- ACLs authorize service-to-service and agent-to-agent communication
- Tokens are associated with policies, which define access permissions
- ACL System Components
    - Token: Bearer token used during the UI, CLI and API request
    - Policy: Grouping of rules that determine fine-grained rules to be applied to token
    - Roles: A collection of policies and/or service identities applied to token(s)
    - Service Identities / Node Identities: Shorthand terms for appropriate minimum ACL policy for a service or node

Bootstrapping Consul ACLs

1. Enable ACLs on agent config file and restart Consul

```
$ cat agent.hcl

acl = {
  enabled = true
  default_policy = "deny"
  enable_token_persistence = true
}
```

2. Create the initial bootstrap token

```
$ consul acl bootstrap
```

3. Apply individual tokens to agents
   a. Create the agent policy
   b. Create the token with the newly created agent policy
   c. Add the token to the agent

Consul ACL Setup Guide

Token Example

Tokens for Service Mesh

Example is a token for service mesh proxy with a stand-alone registration file

CODE EDITOR

```
namespace "<namespace name>" {
#Register the service within its namespace

 service "dashboard" {
   policy = "write"
 }
#Register the sidecar proxy within namespace
 service "dashboard-sidecar-proxy" {
   policy = "write"
 }
}

namespace_prefix "" {
#Be able to find upstream services
 service_prefix "" {
   policy = "read"
}

#And where those upstreams are running
 node_prefix "" {
   policy = "read"
 }
}
```

Token Example

Token for DNS

Example is policy that provides read privileges for all services, nodes, and prepared queries

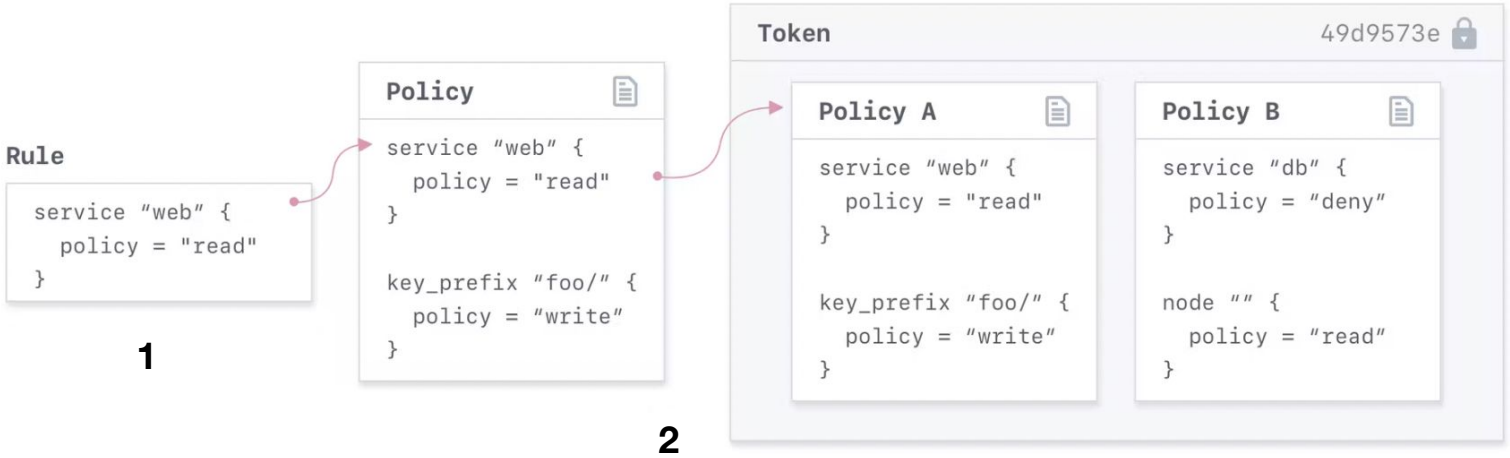
CODE EDITOR

```
$ cat dns-request-policy.hcl

namespace_prefix "" {
  policy = "write"
 node_prefix "" {
   policy = "read"
 }
service_prefix "" {
  policy = "read"
}
# only needed if using prepared queries
 query_prefix "" {
   policy = "read"
 }
}

$ consul acl policy create -name "dns-requests" -rules
@dns-request-policy.hcl

$ consul acl token create -description "Token for DNS
Requests" -policy-name dns-requests

$ consul acl set-agent-token default "<dns token>"
```

ACL Creation

1. Authentication rules are specified to define a policy

2. ACL Administrator generates and links a token to a policy



ACL Implementation

1. Authentication rules are specified to define a policy
2. ACL Administrator generates and links a token to a policy
3. Tokens are distributed and incorporated into services
4. Agents and services present tokens when making requests
5. Consul evaluates token for valid permissions

ACLs Best Practices

- Implement least privileges policy for every ACL token generated
- Using an exact match resource rule is the most secure, this grants least privileges necessary to accomplish a task
- Don’t reuse tokens for multiple services, generate a unique token per service
- Use allowlist (default deny) approach to force explicit anonymous access or token usage for all requests
- Reuse policies and roles for homogeneous environments or similar services
- Don't use bootstrap token or a global token for management
- Rotate tokens on a regular basis

Important Token Considerations

Global vs. Local Tokens for Federation

- Tokens can be created as local which results in them not being replicated globally
- Local tokens are not available or valid in federated datacenters
- Global tokens are the default type
- Certain Consul features require global token, such as Mesh Gateway service tokens and replication tokens

The anonymous token policy is NOT additive

- ACLs are not additive with the anonymous token policy
- If you grant read catalog permissions to the anonymous token, other tokens must also be explicitly granted read capabilities as well

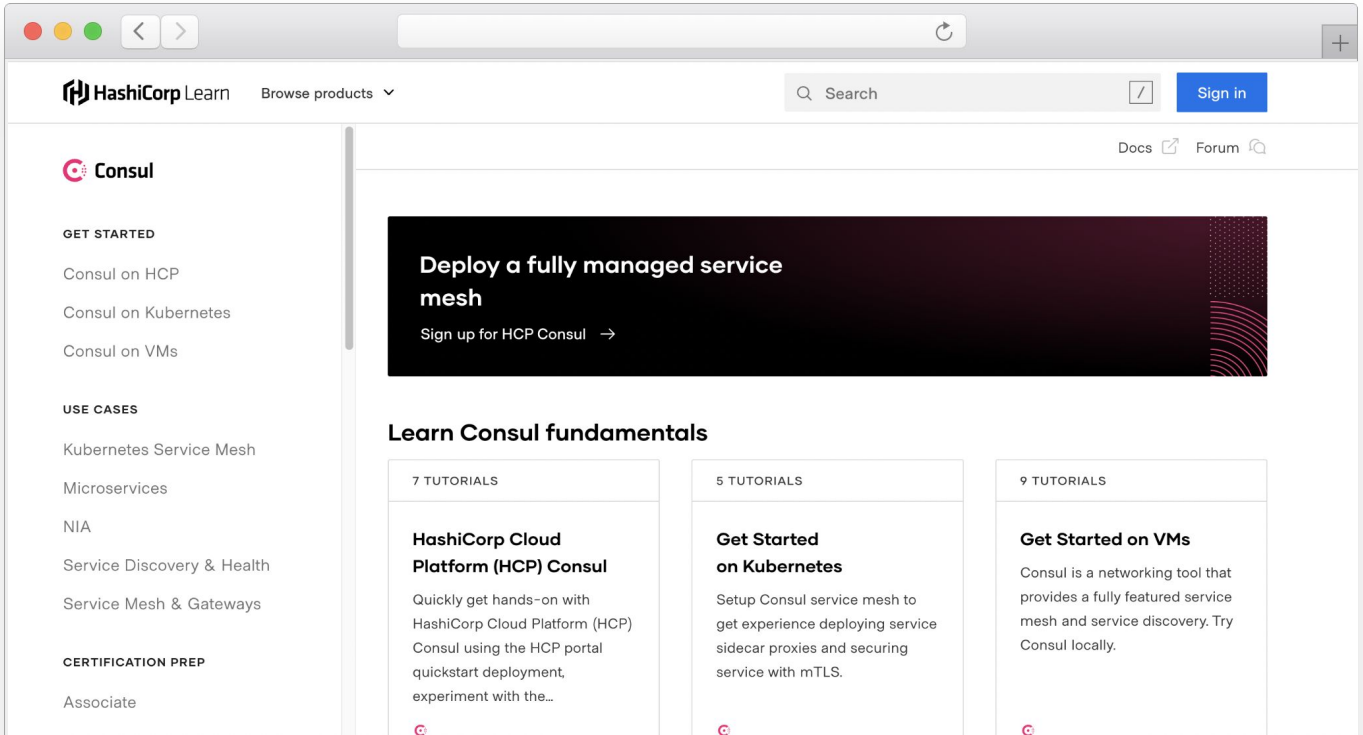
07

Next Steps

Learn

https://learn.hashicorp.com/consul

Step-by-step guides to accelerate deployment of Consul



Resources

- Automated Upgrades with Autopilot
- Upgrading Consul
- Federation between Kubernetes Clusters
- Federation between VM and Kubernetes Datacenters
- Namespace Setup Tutorial
- Admin Partitions Tutorial
- ACL Setup Guide
- Consul Snapshot Command
- Telemetry Reference
- Grafana Dashboard

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 Onboarding Webinars

Office Hour

An interactive open forum to discuss specific questions about your environment and Use Cases.

Please bring your questions!

Webinar:

Service Discovery and Health Monitoring

Topics include: implementing Consul service catalogue, health checks, and prepared queries for geo-failover

Office Hour

An interactive open forum to discuss specific questions about your environment and Use Cases.

Please bring your questions!

Q & A

Thank You

customer.success@hashicorp.com
www.hashicorp.com/customer-success