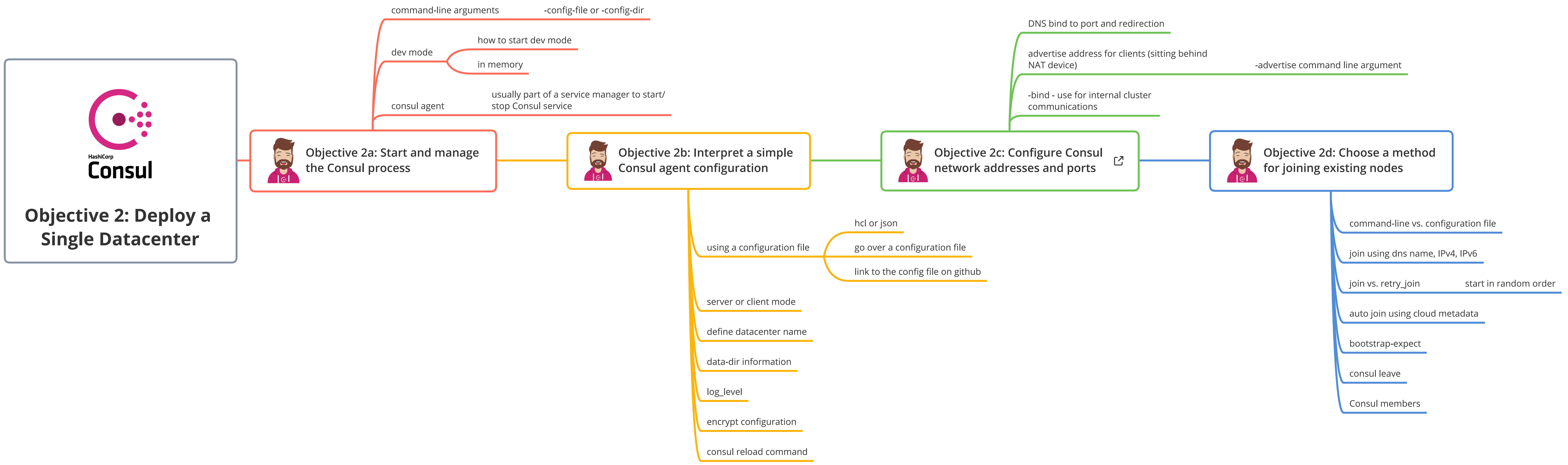
HashiCorp Consul
Objective 1: Explain Consul Architecture
Getting Started with HashiCorp Consul
Created by Bryan Krausen
Objective 1a: Identify the components of Consul datacenter, including agents and communication protocols
Consul Basics
supported architecture
Supported platforms
datacenter
multi-datacenter
multiple clusters
federation
WAN gossip pool
can be multiple locations
single datacenter
single cluster
single LAN gossip pool
single location
network traffic ports
api
LAN gossip
WAN gossip
RPC
DNS
sidecars
key protocols
Serf (gossip)
used by all cluster members
Raft (consensus)
only used by server nodes
Objective1b: Prepare Consul for high availability and performance
high availability
voting vs non-voting members
quorum size vs. fault tolerance
redudancy zones
autopilot
Objective 1c: Identify Consul's core functionality
service mesh
mTLS
intentions
service discovery
api
dns
networking automation
KV store
Objective 1d: Differentiate agent roles
Consul agent
Server
leadership elections
maintain log entries across server nodes
establish and maintain a quorum
definitions
peer set
quorum
logs
server node state
follower
leader
Client
gossip pools
LAN
WAN
dev mode

Getting Started with HashiCorp Consul

Created by Bryan Krausen


HashiCorp Consul
Objective 2: Deploy a Single Datacenter
Objective 2a: Start and manage the Consul process
command-line arguments
-config-file or -config-dir
dev mode
how to start dev mode
in memory
consul agent
usually part of a service manager to start/ stop Consul service
Objective 2b: Interpret a simple Consul agent configuration
using a configuration file
hcl or json
go over a configuration file
link to the config file on github
server or client mode
define datacenter name
data-dir information
log_level
encrypt configuration
consul reload command
Objective 2c: Configure Consul network addresses and ports
DNS bind to port and redirection
advertise address for clients (sitting behind NAT device)
-advertise command line argument
-bind - use for internal cluster communications
Objective 2d: Choose a method for joining existing nodes
command-line vs. configuration file
join using dns name, IPv4, IPv6
join vs. retry_join
start in random order
auto join using cloud metadata
bootstrap-expect
consul leave
Consul members

Getting Started with HashiCorp Consul

Created by Bryan Krausen

HashiCorp Consul
Objective 3: Register Services and Use Service Discovery
Objective 3a: Interpret a service registration
command: consul services register
works with local agent
default namespace for service
<name>.service.consul
default behaviour of registered services
when are they returned by Consul?
when they are not returned by Consul?
service health check failure
node health check failure
service was deregistered
determine what IP and port for service
determine required fields
determine defaults
Objective 3b: Differentiate ways to register a single service
registered via API
service definition as file
.hcl
.json
place inside -config-dir
create new definition
use consul reload
reloadable configurations
Objective 3c: Interpret a service configuration with health check
schedule the frequency of health checks
updating health checks
API
update config file & reload
differences between ID and NAME
health check port
types of health checks
types of checks
define health checks in configuration
define multiple health checks
script check
HTTP check
tcp check
TTL check
Docker check
gRPC check
alias check
system level (host) health check
application (service) health check
Objective 3d: Check the service catalog status from the output of the DNS/API interface or Consul UI
use DNS to determine healthy nodes
use the API to determine healthy nodes
using the Consul UI to determine healthy nodes
Objective 3e: Interpret a prepared query
create
using the API
purpose of prepared query
failover policies
static policy
dynamic policy
hybrid policy
tags
Objective 3f: Use a Prepared Query
default namespace for PQ
<name>.query.consul
actions taken based on prepared query results
order of results based on local and federated services

Objective 4: Access the Consul key/ value (KV)

- Objective 4a: Understand the capabilities and limitations of the KV store
  - what is the KV store
    - distributed architecture - replicated across all server nodes
    - used to store arbitrary data in Consul
    - not a database/datastore
    - often used to store runtime configs
    - limitations
      - no restrictions on type of object stored
      - limit of 512KB object size
    - accesible from any agent
    - always enabled in Consul
  - organizing data
    - common to use / to organize data
    - does not have a directory structure
    - / is treated like any other character
  - backups
    - consul snapshot save
    - consul snapshot agent
  - data security
    - data is NOT encrypted
    - store only non-sensitive data
      - use Vault instead
  - organization (structure)
- Objective 4b: Interact with the KV store using both the Consul CLI and UI
  - access via the CLI
    - commands to add data
    - commands to retrieve data
  - access via the API
    - base64 encoded
  - access via the UI
  - limit access to KV with ACLs
- Objective 4c: Monitor KV changes using watch
  - monitor values for updates
    - take action
    - alert
  - data updates cause a watch to trigger a handler
    - run an executable
    - call to an API endpoint
  - changes supported
    - key
    - keyprefix
    - nodes
    - etc
  - configure using the CLI or API
  - built-in to Consul
    - doesn't need additional binary
    - no additional configuration
  - data returned
    - returns updated data
    - returns any additional matching entries
  - consul watch CLI command
- Objective 4d: Monitor KV changes using envconsul and consul-template
  - separate binary for each
  - use cases
    - application integration
    - populate a templated file
      - populate with KV data
      - populate with data from a registered service
    - set values for environment variables

Getting Started with HashiCorp Consul

Created by Bryan Krausen

Getting Started with HashiCorp Consul

Created by Bryan Krausen

Objective 5: Back up and Restore

Objective 5a: Describe the contents of a snapshot

- what are snapshots?
  - atomic, point in time snapshots
  - includes
    - key/value entries
    - service catalog
    - prepared queries
    - sessions
    - ACLs
  - snapshots by leader vs. follower (stale)
    - consistent state = leader = normal cluster state
    - stale = if cluster has no leader
  - gzipped tar archives
    - includes raft metadata
    - includes a binary serialized version of Consul state

Objective 5b: Back up and restore the datacenter

- perform snapshots using the CLI
  - consul snapshot
    - agent(ENT)
    - inspect
    - restore
    - save
- perform snapshots using the API
- restore operations
  - not designed to handle server failures during restore
  - "all or nothing"
    - disruptive command
  - recover from DR
  - restoring to fresh set of Consul servers
- backup frequency
  - automated
  - manual
    - before upgrades
    - bootstrap a new datacenter w/ same name

Objective 5c: [Enterprise] Describe the benefits of snapshot agent features

- long-running daemon (automated snapshots)
  - customizable interval
- performs leader election
  - highly available
  - automatic failover
- registers itself with Consul
  - benefits of health checks
- snapshots are reported in agent log
- ID = unix timestamp with nanosecond resolution
- snapshot storage
  - local
  - remote
    - S3-compatible
    - GCP Cloud Storage
    - Azure Blob Storage

Getting Started with HashiCorp Consul

Created by Bryan Krausen

Objective 6: Use Consul Service Mesh

- **Objective 6a: Understand Consul Connect service mesh high level architecture**
  - Formerly called Connect (now Service Mesh)
  - what is Service Mesh?
    - service-to-service connection authorization and encryption
    - uses mTLS for auth and encryption
    - applications may or may not be aware of Service mesh
    - platform agnostic
      - physical networks
      - cloud
      - software-defined networks
      - cross-cloud
    - options to integrate apps with connect
      - applications use sidecar proxy architecture
        - most common is Envoy
        - Consul can deploy its own sidecar
      - native integration
        - consul (sdk) aware
  - App Security
    - uses intentions to permit/deny access between services
      - multi-datacenter aware (federation)
    - enables encryption in-transit between services that might not have been encrypted before
    - identity-based authorization
  - Layer 7 observability
    - proxies see all traffic
    - proxies can collect L7 metrics
    - exportable to monitoring tools
  - upstream vs downstream
    - upstream = the service that another service depends on is upstream from that dependent service
    - downstream = the service that depends on another service is downstream to that service
- **Objective 6b: Describe configuration for registering a service proxy**
  - registration does not start the sidecar proxy
  - configuration file
    - declare as connect-proxy (kind)
    - destiniation_service_name (the service the proxy represents)
    - port the proxy is running on so other services can connect to it
    - signify upstream services
      - can have multiple upstream services defined
      - upstream services may or may not live behind proxy
- **Objective 6c: Describe intentions for Consul Connect service mesh**
  - changing an intention doesn't affect existing connections
  - define access control for services in Consul
    - allow communication
    - deny communication
  - intentions can be managed using API, CLI, or UI
  - where are intentions enforced
    - inbound connections
    - proxy requests
    - within natively integrated app
  - controlling authorization
    - control via L4 or L7 depending on protocol used
    - L4 - identity based (TLS)
      - all or nothing accesscontrol
    - L7 - application--aware
      - can be based on L7 request attributes
  - only ONE intention controls authorization at any given time
  - default intention behavior is controlled by default ACL policy
    - allow all, then all connections are allowed by default
    - deny all, all connections are denied by default
  - precedence and match order
    - top-down approach
    - cannot be overridden
- **Objective 6b: Check intentions in both the Consul CLI and UI**
  - using the CLI
    - consul intention command
      - get
      - check
      - create
        - default = allow
      - delete
      - match
      - list
  - using the UI
    - using the "intentions" tab in the UI
  - API
    - /connect/intentions/exact
      - returns 'true' if created successful
    - /connect/intentions was deprecated in Consul 1.9.0

Getting Started with HashiCorp Consul

Created by Bryan Krausen

Objective 7 - Secure Agent Communication

Objective 7a: Understanding Consul security/threat model

- Gossip protocol (serf)
  - protected with symmetric key
  - shared secret
  - can use consul keygen to create key
- ACL system
  - optional
  - protect access to data and APIs
- Consul agent
  - supports encrypting all of its data
  - can use encryption to run Consul over untrusted networks
- HTTP/RPC
  - uses TLS to encrypt
    - forces TLS for all RPC and HTTPS connections
  - can verify authenticity of servers and clients
  - supports the ability to verify incoming/ outgoing communcations
- Consul TLS certificates
  - certificate types
    - server (consul tls cert create -sever)
    - clients (consul tls cert create -client)
    - cli (consul tls cert create -cli)

Objective 7b: Differentiate certificate types needed for TLS encryption

- certificates must be signed the same CA
  - can use Consul as a CA
    - enabled if connect is enabled without specifying a CA provider
    - consul can automatically distribute client certificates (you can do it manually as well)
  - can generate your own certs from another CA
    - must distribute certs manually, aka the ' operator' method
  - certificates must include server.< datacenter>.<domain> as a SAN
  - you can update to a new provider at any time
- setting the https port for the API
- certs required
  - API/RPC use TLS certs
  - connect uses mTLS

Objective 7c: Understand the different TLS encryption settings for a fully secure datacenter

- verify_server_hostname
  - verifies all outgoing TLS connections that the TLS cert from servers contains server.< datacenter>.<domain>
  - is false by default, so it only validates that cert is signed by a trusted CA
  - prevents compromised client from restarting agent as a server and access Consul state
- verify_incoming
  - requires all incoming connections use TLS with a cert signed by a CA included in the ca_ file or ca_path
  - valid for both RPC and HTTPS API
  - by default, this is false (must be enabled)
- verify_outgoing
  - requires all outgoing connections use TLS with a cert signed by a CA included in the ca_ file or ca_path
  - by default, this is false (must be enabled)
  - applies to both clients and servers

Objective 8 - Secure Services with ACLs

Getting Started with HashiCorp Consul

Created by Bryan Krausen

- Objective 8a: Set up and configure a basic ACL system
  - optional ACL system
    - control access to data and API
    - uses tokens associated with policies
    - must be enabled in the Agent config file
      - configuration includes default policy and other parameters
      - includes config on both servers and clients
  - policies
    - grouping of rules that can be used and associated with tokens
    - multiple policies can be created as needed
  - tokens
    - aka bearer token
    - includes an Accessor (name/id)
    - includes a Secret ID (actual token)
  - roles
    - grouping of a set of policies and service identities
    - can be applied to many tokens
  - service identities
    - used for Service Mesh
    - policy template to link a policy
    - used at authorized to allow a service and sidecar to access services and features in Consul
  - bootstrapping the ACL system
    - creates the bootstrap and anonymous toker
    - required before ACL system can be used
    - only done one time
      - there is a "reset" feature if bootstrap token is lost
    - default policy should be set to allow during bootstrapping process
      - all actions require a token after the default policy is set to deny
      - eventually you need to set default policy as deny after updating agent configurations with the proper token
- Objective 8b: Create policies
  - default policies
    - global management
    - namespace-management
  - define different resources available to create rules
    - differentiate between <resource> and <resource>_prefix
    - node identities in a policy
  - policies are attached to a token
    - multiple policies can be attached to a single token (combination of permissions)
  - consul acl policy create
  - creating a policy for the anonymous token
- Objective 8c: Manage token lifecycle: multiple policies, token revoking, ACL roles, service identities
  - consul acl token create
    - create token attached to policy
      - create a token with multiple policies
      - add a description
    - clone
    - delete
    - list
    - read
    - update
  - default tokens
    - bootstrap (aka master)
      - always ID 00000000-0000-0000-0000-000000000001
    - anonymous
      - always ID 00000000-0000-0000-0000-000000000002
- Objective 8d: Perform a CLI request using a token
  - set the CONSUL_HTTP_TOKEN environment variable
  - set the CONSUL_HTTP_TOKEN_FILE environment variable
  - reference token value stored in a file using the -token-file flag
  - using the -token flag when issuing a command
- Objective 8e: Perform an API request using a token
  - set the token using the X-Consul-Token header in the API request
  - set the token using the Authorization: Bearer header in the API request

Getting Started with HashiCorp Consul

Created by Bryan Krausen

Objective 9 - Use gossip encryption

- **Objective 9a: Understanding the Consul security/threat model**
  - review critical components of model
    - Consul TLS
    - HTTP/RPC security
    - Consul Agent
    - ACL System
    - Gossip Encryption
  - by default, Consul is not secure, including the Gossip communication
- **Objective 9b: Configure gossip encryption for the existing data center**
  - using the -encrypt parameter
    - encrypts the gossip communication
    - usually set on the agent configuration
    - can be set using the flag if running on the CLI
  - key
    - 32-byte, Base64 encoded key
    - can use consul keygen to create
    - all agents in the cluster must use the same key
  - you can modify an existing cluster to encrypt gossip communications without incurring downtime
    - uses agent configuration parameters
      - encrypt_verify_incoming
      - encrypt_verify_outgoing
      - on an existing cluster, set value to false initially
      - eventually these will be set to true
    - requires rolling update of agents but can do one at a time
- **Objective 9c: Manage the lifecycle of encryption keys**
  - use consul keyring to manage encryption keys
    - decrypting messages becomes more expensive when there is more than one key active, as multiple attempts to decrypt any given message are required
    - capabilities
      - distribute new keys
      - retiring old keys
      - changing the key used for encryption
      - listing existing keys
  - use consul keygen to create new keys
  - consul keyring CLI command
    - -install
    - -use
    - -list
    - -remove