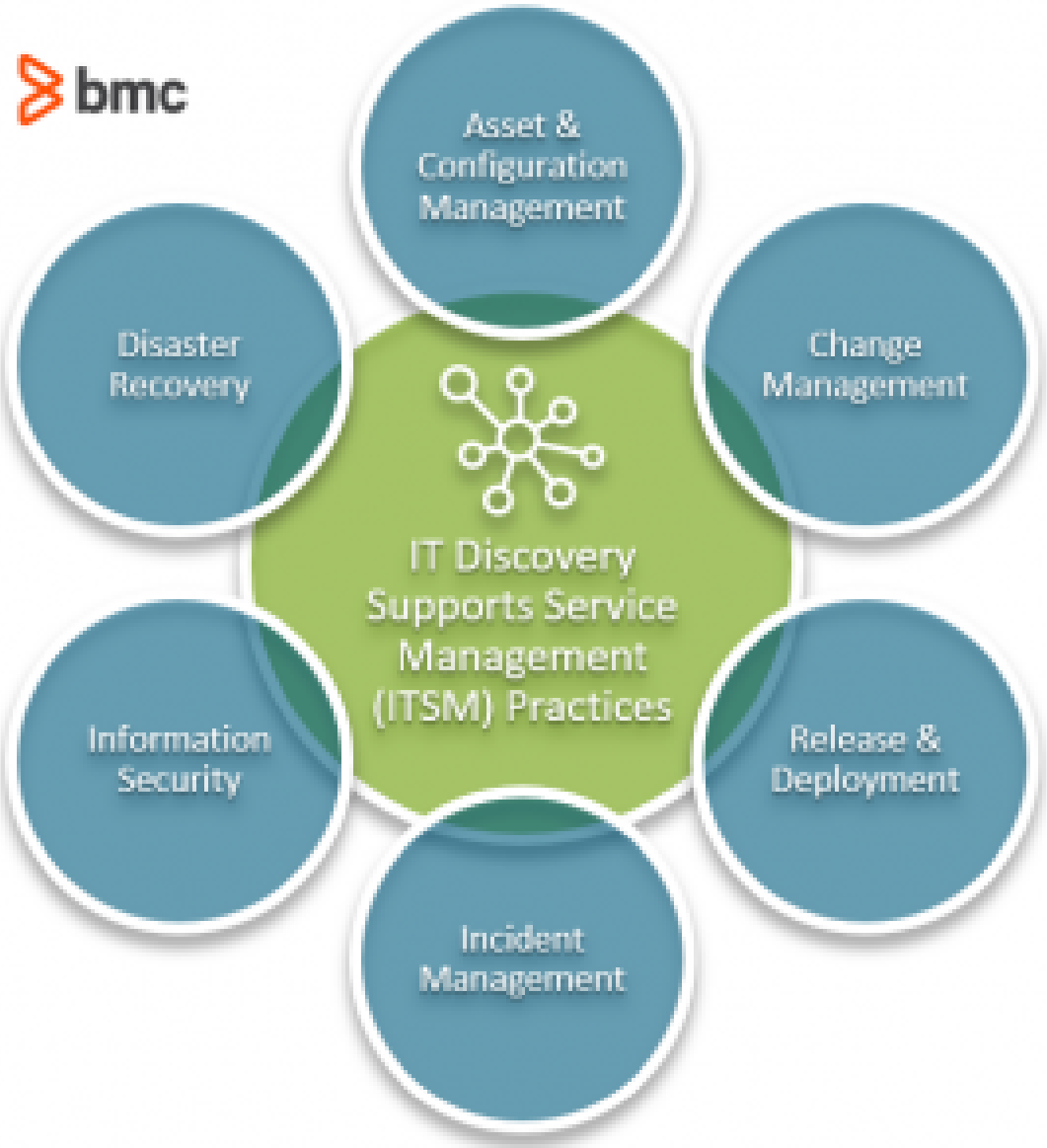
SERVICE DISCOVERY IN THE ENTERPRISE: A BEGINNER'S GUIDE

Modern software applications consist of highly dependent IT service components that operate on a complex IT infrastructure. A performance issue or outage facing an individual component can impact a vast global user base. Modern apps are also constantly updated with new features that follow rapid release cycles of a DevOps software development lifecycle (SDLC).

Consider the case of leading enterprises killing it at DevOps: AWS deploys code every 11.7 seconds, Etsy performs over 50 deployments per day, Netflix deploys code thousands of times per day.

The new deployments are tied to frequent configuration changes of the underlying IT services and resource provisioning that must be controlled carefully. The first step toward achieving this goal is to monitor and trace the services and their corresponding configurations, device information, and network data.

So, let's take a look at service discovery.



What is service discovery?

Service discovery is the process of automatically identifying services and applications on the network. It's considered a subset of general IT discovery.

Long ago, service discovery started as a manual process that involved all applications and services listed in a single global HOSTS.txt file. Address information, configuration details and device data were included in similar text files to keep track of services operating on an on premise IT network.

With the prevalence of connected datacenter systems communicating over the Web, and later on leading to highly distributed cloud computing infrastructure, service discovery evolved into an automated process.

Common service discovery implementations often use the following methods:

1. **DNS-based approach.** Using standard DNS libraries as clients. Each microservice is assigned an address that can be used to locate and connect with other microservices. Proxies such as NGINX can also be used to poll DNS addresses for discovering services over the network.
2. **Key/value store and sidecar.** A centralized data storage system is used to maintain the addresses of connecting services. A communication mechanism, such as sidecar, is used to

identify microservices that are already configured to interface with local proxies.
3. **Specialized library.** Developers directly call an API to discover connected service resources. Client libraries are available along with other services that are tracked and respond to API calls.

Now that we understand service discovery, let's look at the typical components of the process.

Service registry

A key part of the service discovery process is service registry, a database containing all details necessary for external API calls and DNS queries to locate a connected or provisioned service component. Some common Service Registry systems examples include:

- Netflix Eureka: A highly-available service registry system developed by Netflix. It uses the REST API to register and query service instances, as fast as every 30 seconds.
- etcd: A highly available and distributed key-value store that can be used for Service Discovery use cases. It's written in Go language and uses the Raft algorithm. The focus of this implementation is simplicity, security (TLS authentication) and speed (10,000 writes per second).
- Consul: A simple and automated solution to discover microservice instances on the network.
- Apache Zookeeper: One of the most popular high-performance service discovery tools for distributed applications.

Service discovery patterns

The implemented service discovery agents can take various patterns in terms of discovering relationships between the client and server side of the service interface. Let's discuss the two main types of Service Discovery patterns:

Client-side discovery

The client queries the Service Registry to identify the location of a service instance and the corresponding load balancing requests. The address of services is already registered at the Service Registry when it is first provisioned, and removed when the service is terminated.

An example is the Netflix OSS service that uses the Netflix Eureka service registry with REST API calls for service querying and discovery. A load balancing algorithm is used to select a service and make the necessary discovery request.

The mechanism requires fewer network hops as compared to server-side discovery. However, the client is always coupled with the service registry and a discovery logic must be programmed for each framework used by the application.

Server-side discovery

With the Server-Side discovery pattern, the client uses a load balancer to:

1. Query the service register
2. Route the request to the appropriate service instance

The load balancer works on behalf of a client, as previously noted in the Client-Side Discovery

pattern. An example is the AWS Elastic Load Balancer (ELB) used as the server-side discovery router. The ELB functions as the load balancer to internal or external Web traffic between all EC2 instances and also as a Service Registry.

Server-Side Discovery does not require developers to include discovery calls within the client code since services such as AWS ELB already provide the service. A router system supporting the necessary protocols must be installed and managed. Most cloud services such as AWS already include the service, which makes it a convenient off-the-shelf solution for service discovery.

Of course, there are drawbacks. Server side discovery takes several additional network hops, which adds to the overall latency in network communication.

Related reading

- BMC Service Management Blog
- Application Mapping: Concepts & Best Practices for the Enterprise
- IT Asset Management: 10 Best Practices for Successful ITAM
- Introduction To Application Portfolio Management
- The State of Service Management Today
- What Is Microservice Architecture? Microservices Explained