Microservice Catalogs Explored

A definition and comparison

"Software systems have become so complex that it's getting too difficult for humans to reason about the systems they design. The complexity spans the team (even small projects with 20 dependencies have a broader committer graph of 10K engineers), the tooling, the architecture, and the users.

Service Governance will be the creation of service catalogues, metadata stores about *how* the software was created, and intelligence systems that inform stakeholders on insights for the system."

Tyler Jewel, The Developer Led Landscape 2022

What is a Microservice Catalog?

A microservice catalog simplifies cloud-native architecture by displaying all services in one place. A microservice catalog facilitates collaboration across organizational siloes, saves money by encouraging reuse, and tames the difficulty common to a shared service architecture. The unified catalog manages all critical information needed to sustain a microservice architecture, including:

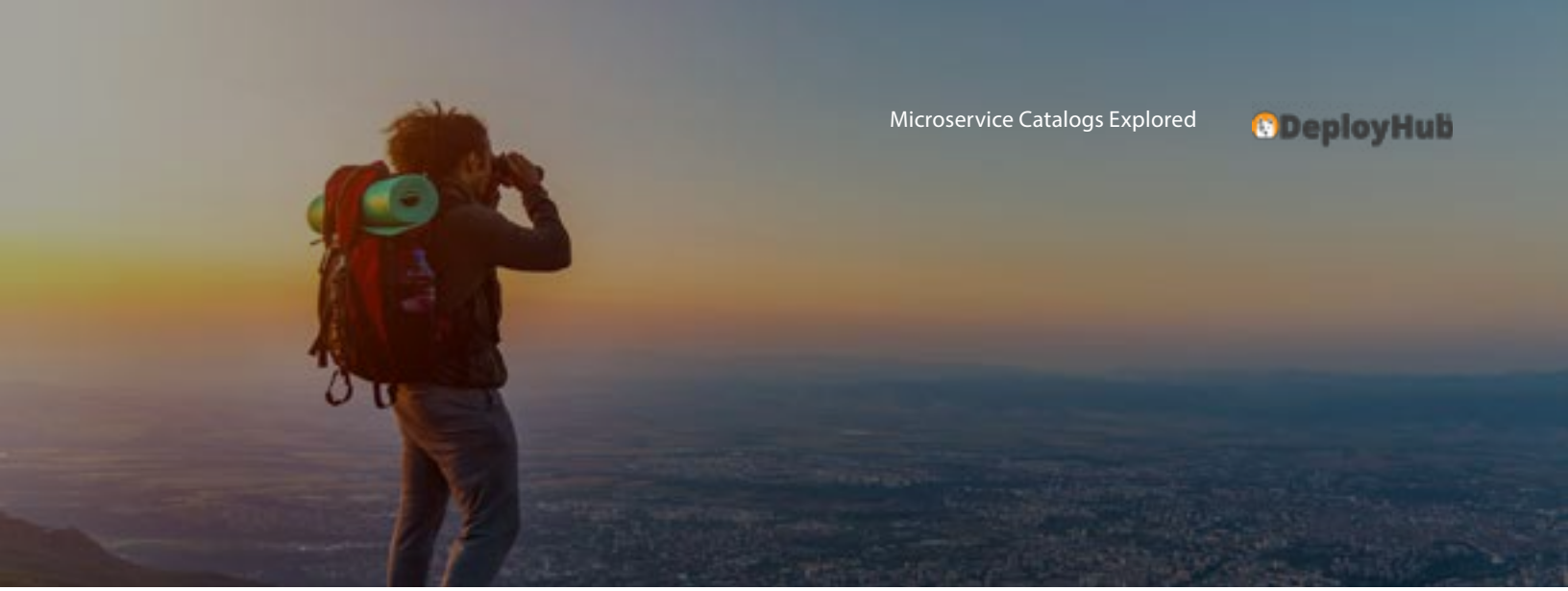
- domain
- ownership
- versions
- deployment requirements
- CD Pipelines
- key value pairs
- inventory
- usage
- SLOs
- dependencies
- logical applications with SBOM

The use of a microservice catalog simplifies a cloud-native implementation by providing a central location to find and share thousands of services consistently across all clusters and teams.

Users of Microservice Catalogs

> A microservice architecture disrupts the way we develop software. The loss of the monolithic application changes the way we manage the creation and delivery of software to end-users. This modern shared microservice architecture has created new roles and responsibilities from the producers to the consumers of services.

Producers - The API or Microservice Developer	Familiar with microservices is domain-driven approach where 'Domain' teams develop small functions to address bounded context problem spaces. These teams are the 'producers' of microservices. They are not delivering software to end-users. They instead deliver functions to 'application' or 'project' teams.
Consumers - Application or Project Teams	Application developers are the consumers of microservices. They create software solutions by writing their unique services and reusing available ones. A collection of microservices is a 'logical' application. Application teams use a microservice catalog to find and reuse available services and package their logical applications.
DevOps Teams	We have all gotten very good at using a continuous delivery process to push monolithic changes across the dev to prod lifecycle. DevOps teams are responsible for keeping a lifecycle process automated even in a microservices architecture. The job of the DevOps Team is to manage changes, report differences, and understand the impact of a single update. Microservices make their job particularly difficult. DevOps Teams use a microservice catalog to track the impact of services and the applications that use them.
SREs & Support Teams	Imagine trying to support a microservice you know nothing about. Solving this puzzle is the job of the Site Reliability Engineer and Support Team. Maintaining SLOs in a microservice architecture is their challenge. These individuals use a microservice catalog to determine who to contact when a service fails.
CISO and Application Security	Governance of the microservice supply chain is essential, however difficult. Tracking what microservices are being consumed by a particular application, and creating a full Software Bill of Material (SBOM) for that application is near impossible in a decoupled microservice architecture. The microservice catalog is used by Security Professionals to govern the supply chain of microservices and create application-level Software Bill of Material reports.



Evaluating Catalogs

Over the last few years, the market has begun to offer several microservice catalogs from which to choose. While they all centralize data on microservices, they all approach the problem differently. The difference often depends on who they are serving. For example, some catalogs track mainly production information and are focused on helping the support teams maintain SLOs. Others cater to the microservice developer and automate the onboarding of services. Others track supply chain intelligence on services to support DevOps teams across the lifecycle. Below is a list of standard features of microservice catalogs:

1

Project Templates

When you create a new microservice, certain DevOps standards are needed for each microservice. Project templates allow the microservice developer to select a template when registering their service. The template prevents developers from scripting a new process for every microservice.

2

Microservice Organization

Structures the catalog in terms of groups that own the services. Domains, teams, or tiered service levels are the basis for the organization. By organizing microservices, you make them easy to find, which reduces redundant coding.

3

Ownership

Assigns developer contact information to the microservice, including slack channel, email, phone number, pager duty. Ownership information is essential for SREs and support teams to maintain incident response times.

4

Service to Service Dependencies

Service to service dependencies shows you the relationships between the services, with a list of the API endpoints that a microservice produces and the API endpoints that it consumes.

5

Application Supply Chain Analysis and SBOMs

A software solution uses microservices as raw materials. A collection of microservices creates a 'logical' application delivered to the end-user. Tracking the services that each application consumes shows who is using the microservices, the service's risk level, and provides the application level 'software bill of material.'

6

Supply Chain Versioning

While it is essential to know the supply chain of each 'logical' application, it is more important to understand what has changed in the supply chain. Tracking changes is the essence of versioning. Each time a microservice is updated, the catalog creates a new version of the service and shows its changes, including the SHA, Key-Value Pairs, CVEs, and Swagger details. A new microservice version also creates a new 'logical' version of any consuming applications. Finally, versioning allows for creating 'difference reports,' an essential tool in debugging and determining root cause analysis.

7

Vulnerability Reporting

Common Vulnerabilities and Exposures (CVE) is a system that provides a reference for publicly known cybersecurity issues. Each microservice is scanned for known problems. The catalog displays the vulnerability report for each microservice.

8

Inventory by Cluster

'Drift' is a typical complexity with microservices. Drift occurs when multiple versions of a single microservice runs across multiple clusters. The catalog tracks the version of the microservice across all clusters.

9

Deployment Metadata

Microservices deploy independently. The catalog contains the information needed to deploy the microservice, including the deployment engine, the deployment logic, and key value pairs.

Microservice Catalogs improve Software Security

> Microservice catalogs are a global representation of your entire microservice inventory. Catalog features critical for achieving security standards and compliance include:

- service to service dependencies
- application supply chain
- vulnerability reporting
- cluster inventory with deployment metadata

Data unified in these core areas create application-level Software Bill of Material (SBOM) reports, tracks vulnerabilities, expose microservice impact, and facilitate the quick update of patched services across all affected clusters.

In other words, if you need to know that the microservices your application is using are up-to-date and patched against a known security vulnerability, you need to know what services your application is consuming and which versions. The centralization of this level of data is a core feature of the microservice catalog.

SBOMs & Microservices

Software Bill of Material Reports (SBOM) at the application level are extremely difficult to track in a microservice architecture. An SBOM is generated at the time a new container is created. However, applications are a collection of microservices. Your application SBOM is an aggregate of all the microservice SBOMs. Each time a new service version is updated, a new SBOM is generated. With each application consuming hundreds of microservices, SBOMS become highly dynamic.

Your microservice catalog can quickly produce an application SBOM if it tracks service and application dependencies. Without a central hub of service to service and application to service data, application-level SBOMs become impossible to track.

Vulnerability and CVEs

Like SBOMs, Common Vulnerability and Exposure (CVE) scanning is done at the microservice level. If you are the microservice developer who learns a service needs a patch, how do you know who is using your service and where it is running?

Catalog tools that track service to service dependencies and the applications supply chain can provide the 'blast radius' information needed to quickly notify all teams that a vulnerability exists and push the patch out to all endpoints confidently.

Cluster Inventory – Where the service is running

To efficiently address a vulnerability, it is essential that you know the version of the impacted service and where it is running across all clusters and stages of the pipeline. The catalog can provide this level of information by tracking the microservice version and where it is running. When the catalog also stores the deployment metadata, it can facilitate a complete update of the patched service to all endpoint locations.

Recognizing Drift

'Drift' occurs when application teams consume different versions of the same microservice. The catalog can show which teams have services lagging behind the latest version. The catalog can identify which application teams need to upgrade to the latest and safest version of a common service.

Microservice Catalog Comparison

Below is a list of the available microservice catalogs with a summary of their primary features.

	Project Templating	Microservice Organization	Microservice Ownership	Service to Service Dependencies	Application Suply Chain Management	Supply Chain versioning	Track SLOs	Vulnerability Reporting	Inventory by Cluster	Deployment Meta Data
DeployHub/Ortelius		X	X	X	X	X		X	X	X
OpsLevel			X				X	X		
Backstage/Roadie	X		X	X				X		X
LeanIX		X	X	X	X		X	X	X	
Tilias			X	X						
Cortex	X	X	X	X			X			

Note: If there is a something we have missed, please let us know. Email us at: request-info@deployhub.com

Conclusion

As you mature in your microservice journey, the need to manage microservices from a central location will become clear. Part of the complexity of a microservice implementation is understanding who uses them, where they are running, and their impact. Critical in this new architecture is to have a clear view of your microservice supply chain. This view includes knowing what microservices are available, what is missing, what versions are running, and their changes over time. The core purpose of a microservice catalog is to provide everyone, from the API developers to the production support teams, essential information needed to succeed in a shared services architecture.

About DeployHub:

> DeployHub's mission is to empower organizations to achieve business agility through a managed approach to the microservice supply chain.

Unique to the DeployHub catalog is its ability to version services along with their consuming applications providing the visibility of service impact before a deployment. DeployHub provides a clear view of your microservices supply chain and how it changes over time.

The members of the DeployHub Team are recognized experts in life cycle and configuration management and have applied that knowledge to a cloud-native architecture.

Get Started with DeployHub

» FREE DEPLOYHUB TEAM SAAS SIGN-UP

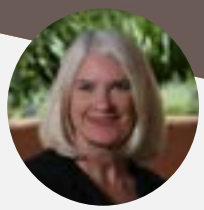
DeployHub Team is based on the open-source Ortelius project incubating at the Continuous Delivery Foundation (Linux Foundation).

This version can be used to track and configure unlimited microservices and applications with unlimited end users & endpoints.

https://www.deployhub.com/microservice-dashboard/

Get Involved in the OS Project

Help us create the best, open source microservice catalog available at ortelius.io. We believe everyone has something to offer in solving the microservice management puzzle. We would love to have you on board.



About the Author:

Tracy Ragan is CEO of DeployHub and has served on the Continuous Delivery Foundation Board.

She is a microservice evangelist with expertise in software configuration management, builds and release. Tracy was a consultant to Wall Street firms on build and release management for 7 years prior to co-founding OpenMake Software in 1995. She was a founding member of the Eclipse organization and served on the board for 5 years. She is a recognized leader and has been published in multiple industry publications as well as presenting to audiences at industry conferences. Tracy co-founded DeployHub in 2018 to serve the microservice development community.

Visit us at DeployHub.com