vFunction User Guide

Introduction

vFunction is a cloud-native modernization platform that combines dynamic and static code analysis, data science, and automation to automatically identify and extract services from existing applications. vFunction is the only platform purpose-built for modernization of Java applications.

The vFunction modernization process starts by learning the running monolithic application, and surfacing the interdependencies within it. Using its algorithms, the platform analyzes and identifies services that can be separated from the application. This decomposition can present a range of micro, mini, or even macro services, depending on your application environment, each being an independently deployable and scalable application component.

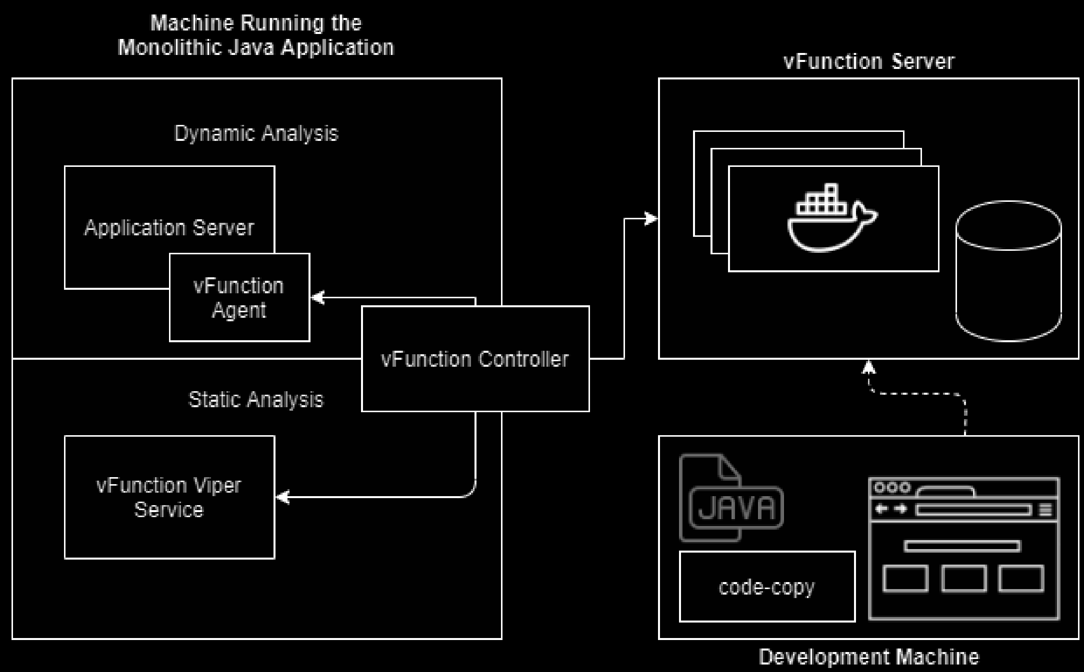
vFunction automates the extraction of these services, enabling you to modernize your monolith, quickly and easily.

The vFunction Platform

Review the following diagram to familiarize yourself with the various components of the vFunction platform. The platform consists of 3 basic components; the server, the controller package, and a tools package. The server runs on a Linux machine, with Docker installed. The controller package is installed on the machine that runs the monolithic application which can be either a Linux or a Windows machine, and the tools are run on a development machine, with access to the code of the monolithic application.

The controller package consists of three elements: the vFunction agent, that collects data during the dynamic analysis phase; the vFunction Viper application, that performs static analysis on the binaries of the application; and the vFunction controller that handles all the communication between the agent, Viper, and the vFunction server.

The vFunction agent is a mix of a Java and native agent, and needs to run on the JVM that is currently running your application. Refer to the vFunction Support Matrix document for a list of supported application servers and JVMs.

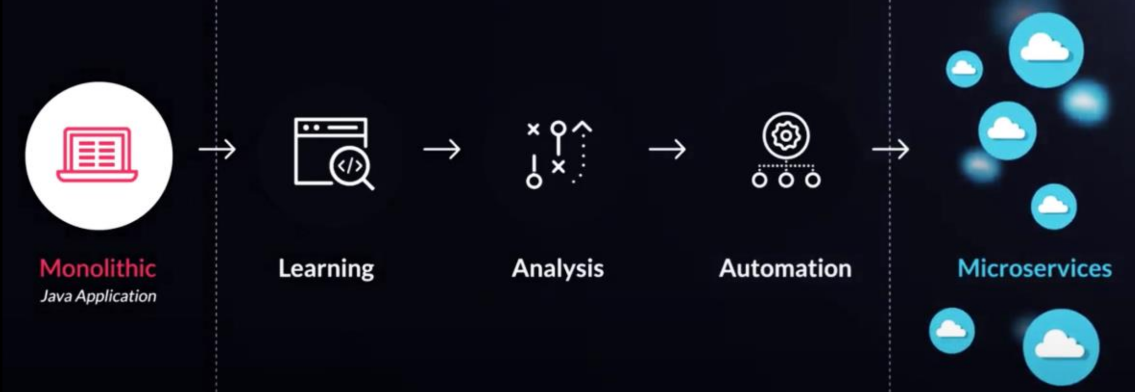


© vFunction Inc. 2021 www.vFunction.com To learn more about vFunction Inc. click here.

Getting started

There are four main steps to take you from your monolith to extracted, independent services:

1	INSTALL	• Install the vFunction server • Create your application in vFunction • Install the vFunction controller package • Add the vFunction agent to the JVM arguments
2	LEARN	• Start a new measurement • Create a new measurement with a baseline • Import a previous measurement
3	ANALYZE	• Pick a service • Explore the service tree • Analyze the service classes & infra classes • Analyze the service resources • Remove dependencies
4	CREATE	• Download the service specification file (i.e., "service spec") • Run the service creation tool to create the service



© vFunction Inc. 2021 www.vFunction.com To learn more about vFunction Inc. click here.

Install

This section takes you through the installation and configuration of vFunction.

Before you begin

- For download, make sure you have access to our portal.
- Have the password provided to you by vFunction handy.

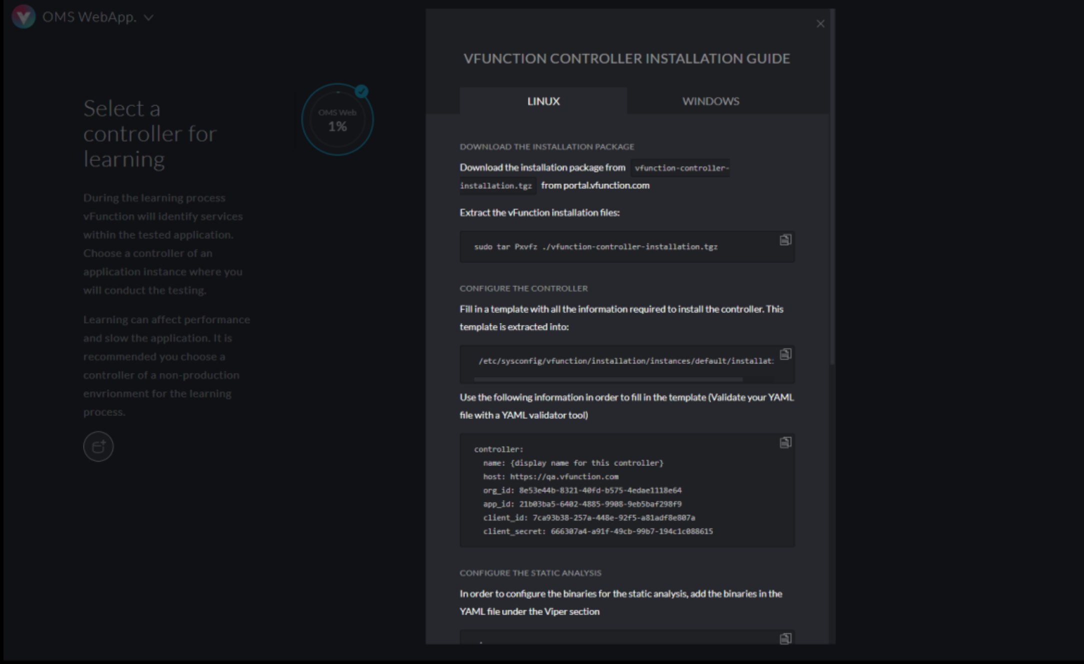
Install the server

Install the server according to the instructions in the installation guide:

- Install using sudo - follow if you have access to root privileges on the server.
- Install using a local user - follow if you do not have sudo privileges on the server.

Create your application

1. Sign in to vFunction in your browser.
2. In the vFunction menu, click **NEW APPLICATION**, and give your application a name.
3. Click **CONFIGURE PARAMETERS**, and in the **Classes to include** box, enter the application's core package prefixes, e.g., **com.vfunction**.
4. Click **Create**.
   The controller installation guide appears.
5. Select **LINUX** or **WINDOWS** and follow the instructions to install the controller.



Install the controller and agent

Install the controller and agent according to the instructions in the installation guide relevant to the OS your application is running on:

- Linux installation with root privileges
- Linux installation with no root privileges
- Windows Installation

Add the vFunction Agent to the JVM arguments

The location of the JVM arguments depends on the environment your organization is using. This task should be performed by the application server admin.

1. Add the agent to the JVM arguments of your application.
   Follow the sample JVM arguments provided in the following location:
   - LINUX: `/etc/sysconfig/vfunction/agent/instances/default/vmargs-examples/`
   - WINDOWS: **C:\vfunction\agent\vmargs-examples**
2. Restart your Java application and make sure the vFunction Agent is running in the arguments and that the application itself is running as expected.
3. To check that the agent is installed and running correctly, look for the following string in the application log file:

```
vFunction native agent version {version number} successfully loaded
```

Learn

You're now ready to get vFunction to learn about your monolithic Java application. During the learning phase, vFunction reviews your application and identifies all the elements that make up its services, and the dependencies and connections between them.

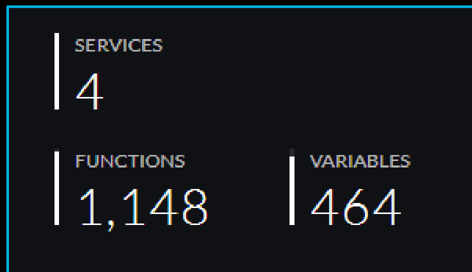
The first time you use vFunction, you'll create a new measurement. A measurement is an instance of Learning. To allow the platform to learn your application, run tests that emulate normal use operations on your application while running the measurement.
In case you need to stop the measurement, you can always go back to re-starting the same measurement, and perform more tests to allow the platform to discover more information. The more tests are performed, and the more information gathered, the platform will be able to provide a more detailed picture of all the connections and dependencies in your monolith.

We recommend you create a new measurement whenever you've updated the code of your application, and need to reassess connections and dependencies.

Create a new measurement

1. Sign in to vFunction and choose the application you want to test.
2. Click **NEW MEASUREMENT +**.
3. Choose the application controller where you want to conduct the learning measurement.
4. Click **START** and ask your QA team to run thorough tests that emulate normal application load for the breadth of the monolithic application.

   While learning runs, identified services and other scanned metrics, such as functions and classes, start to appear in the vFunction console:



5. When you finish testing, or when no additional metrics are identified by vFunction, click **STOP**.
   The **STOP** button changes to **STOPPING** while the process finishes gathering all information in the queue. The process is stopped completely when the **START** button appears.

Create a new measurement with a baseline

If you've made changes to your code, starting a new learning measurement enables you to understand the impact of your changes.
However, while starting a new measurement, it is possible to base the measurement on an existing "baseline", which means that an analysis that was performed on an already existing measurement will be taken into account when analyzing the new information from the new measurement.

After creating a new measurement, and before you click **START**, click **SET BASELINE**, and choose the measurement to set as the baseline for your architectural settings.

Import a previously collected measurement

At any point, you can download a collected measurement, save it, and upload it later to create a new measurement from its data.
To import a measurement click **IMPORT** and upload a measurement zip file to use the measurements collected in a previous learning phase.

© vFunction Inc. 2021 www.vFunction.com To learn more about vFunction Inc. click here.

Analyze

After the learning phase is complete, use vFunction to analyze the services and their dependencies in your current monolithic application. During the analysis phase, review what vFunction learned about your application, and use the information to begin to isolate application services.

Explore the call tree of each service to understand the boundaries of the service, and analyze the service classes in order to understand interdependencies. You will find that you move between these two tasks in order to know your service and plan its extraction.

Afterward, you will analyze the service resources and remove dependencies to build your target architecture.

Exclusivity

Exclusivity is a key concept of the analysis phase. During the analysis phase, a class is considered exclusive if it is used within a single service. Similarly, service exclusivity is defined as the percentage of classes that are exclusive to it. This means that the higher the service exclusivity is for a given service, the more functionality is extracted from the monolithic application once the service itself is extracted. More specifically, every class that was exclusive to the extracted service is no longer needed throughout the monolithic application and can be removed from there.

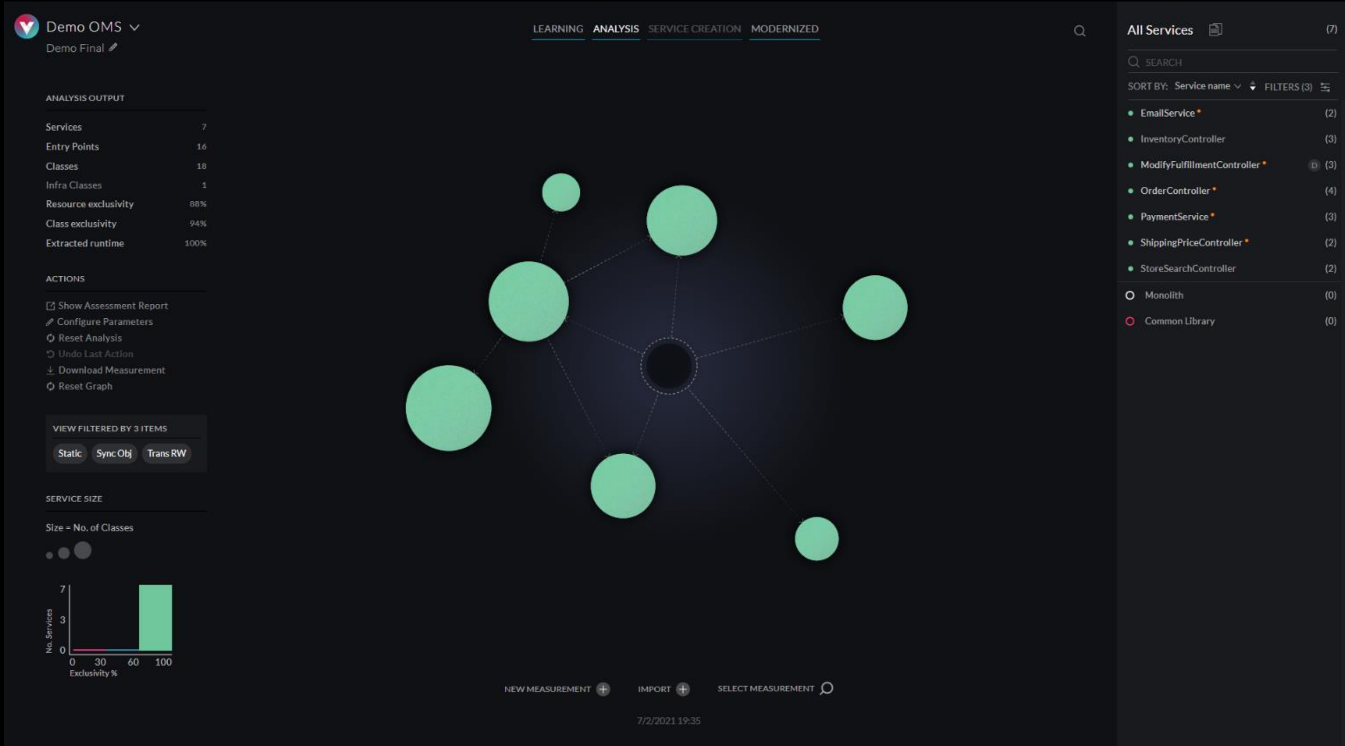
Your goal

During the analysis phase, **your goal is to increase the exclusivity of each service** as much as possible - this is the key to defining good services. The higher the exclusivity, the better defined the service is, and once extracted, the less functionality resides in the monolith.

© vFunction Inc. 2021 www.vFunction.com To learn more about vFunction Inc. click here.

What you can see in the Analysis tab

Access the **ANALYSIS** tab to review the services that were automatically identified by vFunction. The tab is divided into three panes.



Left pane

The number of identified services and entry points (the functions/methods that call on the identified services) are shown in the **ANALYSIS OUTPUT** list.

Click next to the Analysis output to open the analysis parameters. See Analysis Parameters for more information.

Center pane

Each sphere presented on the center pane represents a service. A service is a group of classes and context that can be potentially separated from the monolith to modernize the application:

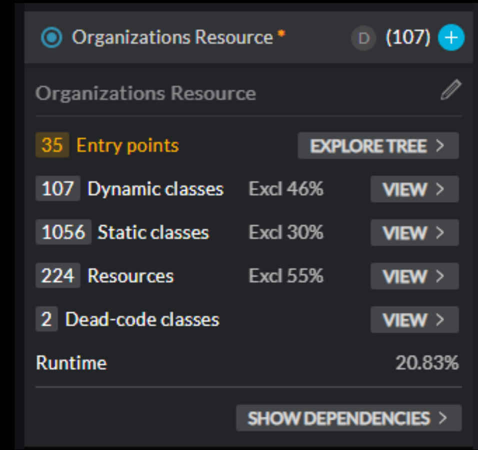
- The sphere size represents the size of the service, measured in the number of runtime-identified classes.
- The sphere color represents the exclusivity of the service, that is, the degree to which classes are accessed exclusively from that service.

  - A **green** sphere shows a service with high exclusivity (61%-100%), often a good starting point for extracting the service.
  - A **blue** sphere indicates the service has medium exclusivity (31%-60%).
  - A **red** sphere indicates that many of the service elements are dependent on other services (low exclusivity, 0%-30%)
- The dotted lines between the spheres represent calls between services. The arrow represents the call direction. A dotted line from the black sphere in the middle is a call from flows that were not extracted from the monolithic application or from existing end-points of the system.
- When selecting a service, solid lines between services may appear. These lines between the spheres represent interdependencies between the services (i.e., non-exclusive resources) based on the parameters filter chosen in the top-right pane - these need to be examined. For more information, see Analyzing the service resources below.

Right pane

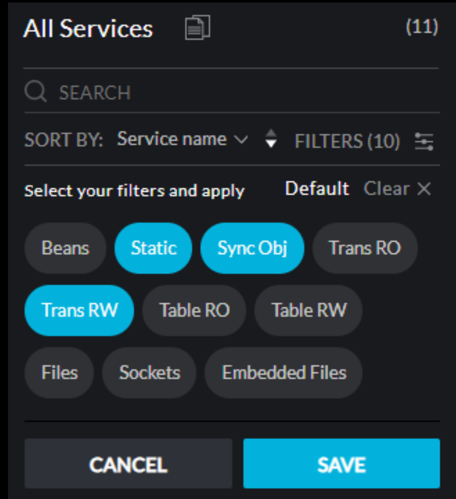
Details of the identified services are shown in the right pane:

- The list of services
- The number of entry points
- The number of dynamic and static classes and their exclusivity
- The number of resources and their exclusivity
- The number of classes that were identified as dead-code
- An estimate of the amount of time spent in that service during testing



The interdependencies filter

The interdependencies filter allows you to choose which interdependencies will be represented by the solid lines between the services in the middle pane when a service is chosen.



Analyze a service

Use the three panes in the Analysis tab together to:

- Understand the number of potential services that can be extracted
- Understand the size and name (i.e., the functionality) of the potential services
- Understand the complexity of separating each service by evaluating its exclusivity
- Assess the amount of work needed to extract these services by understanding the number of interdependencies across classes and resources
- Prioritize which services to separate first

Pick a service for modernizing

Choose one service at a time to analyze and modify. To access your chosen service, click the sphere in the center pane, or click the service name in the right pane.

Explore the tree

1. Click **EXPLORE TREE** and begin to familiarize yourself with the service and its functions.
   In this call-tree, you can see the classes or functions that make up the service. 3rd party classes and classes that appear in a large number of services ("Infra classes") appear in different colors to allow you to focus on the core functionality of the service.
2. The entry-points of the service are methods that once invoked trigger the service. A call to an entry point of a different service signifies a service-to-service call in the presented decomposition of the application.
3. Once you have familiarized yourself with the services and after having reviewed the exclusivity of the classes, the call tree allows you to explore where a class is accessed from. The call-tree then allows you to define new entry-points to any service to better define the service boundaries, and improve the exclusivity of the service.
4. From exploring the entry-points of a service, you also can merge a service entirely into another service, create a new service from a single entry-point, move entry-points to a different or to completely remove a single entry point, all to further optimize the service boundaries and continue to increase the exclusivity of the services.

© vFunction Inc. 2021 www.vFunction.com To learn more about vFunction Inc. click here.

Analyze the service classes

A class can either be **exclusive** to a service, **non-exclusive**, meaning it appears in a small number of services, or **infrastructure** ("**infra**") if it is a dependency for many services.

Infra classes can either be extracted with the service itself and duplicated between services or packaged in a library that you'll subsequently make common to many services in an extracted "common-library".

Reviewing the list of classes in your monolith and understanding whether they should be exclusive, non-exclusive, or infra is critical.

1. Click **VIEW DYNAMIC** to view classes that were identified in the learning process using dynamic analysis.
2. Click **VIEW STATIC** to view the compile-time dependencies of the dynamic classes that were found using static analysis.
3. The right pane is divided into tabs for classes that are exclusive (**EXCL**), non-exclusive (**NON**), infrastructure elements (**INFRA**), and all (**ALL**).
4. Click **EXPLORE** to drill into each class to understand how the service can be isolated.
5. Change the class as appropriate.
   For example, select a class and click **MARK CLASS AS INFRA** to change it to an infrastructure element. This element will be "ignored" when isolating the service, and ultimately will be included in a shared library used by multiple services.
   On the other hand, if you find a class identified by vFunction as **infra**, but you think it should be part of a particular service, mark the class **exclusive** or **non-exclusive**, as applicable.
6. In case the infra class is from a jar that all its classes can be classified as "infra", select the class and click **MARK JAR AS INFRA.** All classes from this jar will be removed from the analysis. To remove a jar from the list of "infra jars" review the Analysis Parameters.

© vFunction Inc. 2021 www.vFunction.com To learn more about vFunction Inc. click here.

Analyze the service resources

In this step, you'll review the service resources, and assess the amount of work needed to separate the service by understanding the interdependencies with other services, and constraints to separating the service.

Resources

Resources are defined as any object that may add a constraint to separating the system into services or can hint to the domain of the service.
The platform tracks seven types of resources:

- **Files**
  When analyzing the resources, review the files accessed by the service, and determine whether they can be made available to more than one service.
- **Synchronization objects**
  Synchronization objects such as locks or atomic resources may be used to protect areas in the code that run in multi-threaded environments. Review all non-exclusive sync objects to make sure the multi-threaded aspects of the service are handled before extracting the service.
- **Static** resource
  A static resource has a single copy in memory shared by all instances of the class. Review all non-exclusive static-variable to make sure they do not constitute a "shared state" in the monolith.
- **Database Transactions**
  A database transaction cannot be shared across service boundaries. Splitting out the service naively may break these complex transactions. Review all the non-exclusive read-write database transactions that were found in the service.
- **Java Beans**
  Spring beans or EJBs are objects that may contain state, and may hint to certain application domains. It is highly preferable to keep certain beans exclusive to one service, as it helps keep the domain unique to the service.
- **Network Sockets**
  Network sockets inform us of external resources that the application requires. If net sockets are not exclusive, a developer will need to review them to make sure the service can be extracted.
- **Database Tables**
  Access to database tables, similar to beans, hints to the application domain in which the service operates. Similar to beans, it is highly preferable that table access (and especially read-write

access) is exclusive to a single service. When exclusive, it's easier to split the data layer down the road.

Reviewing the resources

1. Next to the resources in the service you are analyzing, click **VIEW**.
   Filter the resources list based on the type of the resource or by it's exclusivity (exclusive, non-exclusive or infra).
2. Review the non-exclusive resources to understand how they are accessed from other services and whether they pose a constraint to extracting the service.

Remove dependencies

As you go, isolate the service elements (entry points, non-exclusive classes, and resources) that should be exclusive, and remove their dependencies.
For each element:

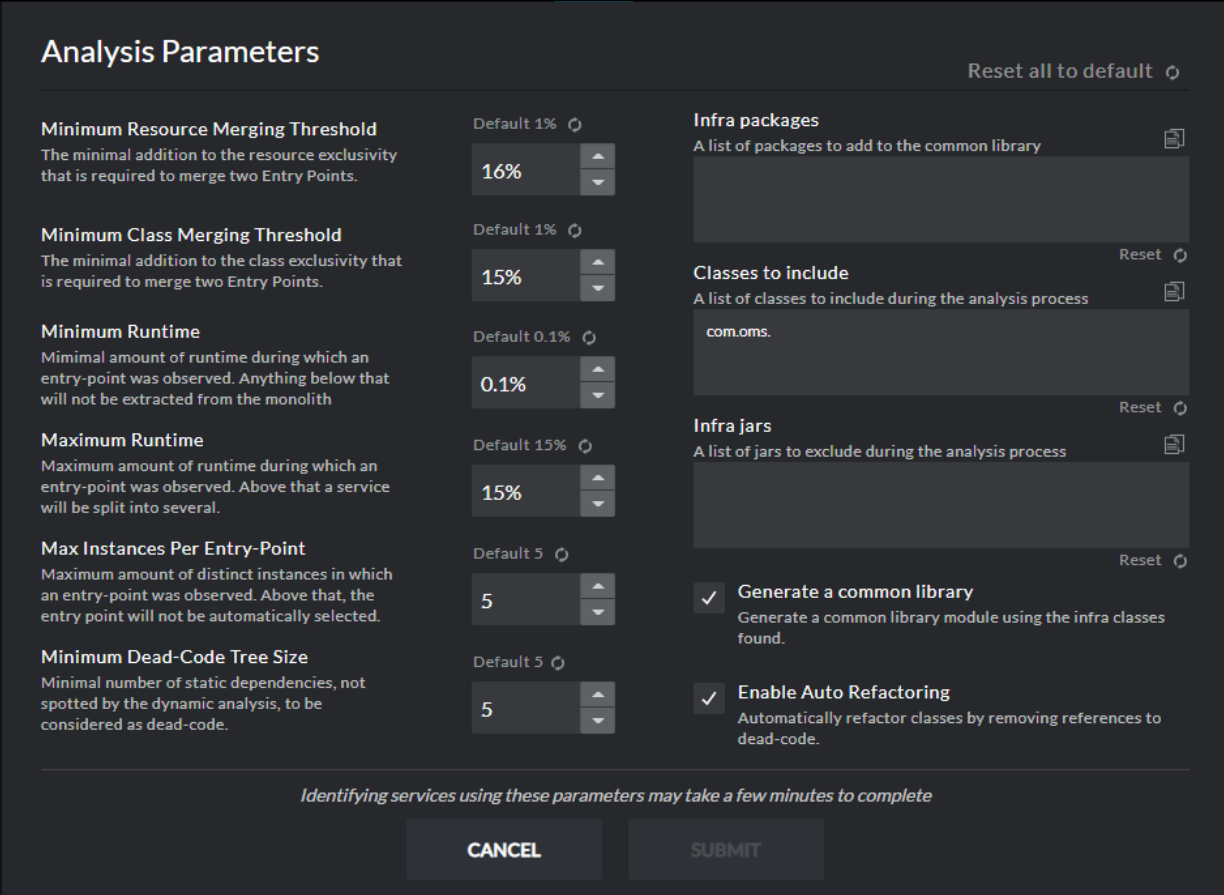
1. Click the element.
2. Click EXPLORE to see where else it is called from.
3. Decide which service logically should be calling the element.
4. Review the call tree from where the class is accessed.
5. Find the right method to set as an entry point to the chosen service.

Following these steps, the vFunction platform will perform the analysis based on the new configuration, and the specific dependency will be removed from the service.

Once your target architecture is ready, you can extract the service.

Analysis configuration

Click next to the Analysis output in the left pane to access the Analysis Parameters screen.



The analysis parameters determine certain behaviors of the analysis and may affect the result of the analysis considerably. However, any modification to the parameters can be reversed, and it will not change any entry points that were previously defined by the user.

Understanding the analysis parameters:

Minimum Variable Merging Threshold & Minimum Class Merging Threshold

The analysis will merge services together if merging the two services will increase the overall variable or class exclusivity of the system by this threshold amount. If the result of the analysis yields too many services, try to reduce the thresholds to allow the platform to merge more services together. If the result of the analysis seems to merge too many services together, increase the threshold in order to let the analysis merge a service only in case the addition to the overall exclusivity is considerable.

Minimum Runtime & Maximum Runtime

In order to allow the user to focus on the important flows in the application, the analysis filters out classes with very little or too much time spent in them. Decrease the minimum runtime in case too few flows are captured by the analysis. Increase it if too many services are presented. Decrease the maximum runtime if a service is found to encapsulate too many of the application flows, increase it if it encapsulates too few flows.

Infra packages, Classes to include, & Infra jars

In order for the analysis to know how to construct the service specification, the user needs to let the analysis know which of the application's classes should be extracted with the service, which should be provided as 3rd party dependencies, and which should be provided as 1st party dependencies, or as application libraries. Add all the package prefixes in your application to the "Classes to include", e.g. com.vfunction., org.vfunction. and add all the packages in your application that you don't want to include in the service to the "Infra packages". If there are jars that contain classes that match the "Classes to include" but you would like that jar to be set as a dependency to the service, add that jar to the "Infra jars" list.

Generate a common library

If you want the analysis to create a library from all the "infra classes" that were found, mark this configuration. Otherwise, if you want the infra classes to be extracted in each service, same as the non-exclusive classes, leave this configuration unmarked.

Enable Auto Refactoring

With this configuration enabled the system will automatically remove methods that were not seen in the dynamic analysis and that are dependent on classes that were marked as dead code.

Extract services

With the target architecture ready, you can start to decompose the application by business capabilities or by application sub-domains.

Download service spec

1. Click on the service you want to extract.
2. Click **Save** , and then click Service Creation State
3. Click on **CONFIGURE**.
   Set the group ID, target platform, dependency repository type (The type of dependency management repository used by the original project. If you do not have one, you must select **LOCAL**.) and generate endpoints. Click on **SUBMIT** once it is configured.
4. Click on **DOWNLOAD**
   The system generates a specification file containing all the information required to automatically create the service.

Run the service creation tool

Before you begin:

Make sure you install the tools on a machine (Linux or Windows) that has access to the monolith's code. Download the vFunction tools from: vFunction-tools and unzip and extract the code-copy executable from either the Linux or Windows folder, depending on your OS.

Run the service creation tool:

Given the service spec file, the tool copies the provided source files according to their packaged structure.
It then adds the dependencies and test files and generates the required configuration files.
Note: Re-running the tool will cause it to add additional sources without overriding existing work.

The following parameters are required to run the code-copy tool:

- **spec**: The path to the service spec file
- **source:** The path to a directory containing the source code of the monolithic application
- **dest**: The path to the destination folder where the new service will be created

Note: Run 'code-copy --help' to see more details on the possible parameters to running the tool.

Example: `./code-copy --spec /home/user/MyService.json --source /home/user/application/ --dest /home/user/new/myService`
Once the code-copy tool is done, the service can be compiled, tested, containerized & deployed.

Glossary

LEARNING

The phase in which the vFunction agent collects data while the application is being tested in pre-production.

ANALYSIS

The phase in which the data collected from the learning phase is analyzed to identify services, resources, functions, classes, and their interdependencies.

SERVICE CREATION

The phase in which the analyzed services will be extracted to compilable, working micro-services, with their classes, dependencies, and resources.

MODERNIZED

This phase marks that the services have been extracted and the application has been modernized.

SERVICES (I.e. Proposed Services)
A group of classes, methods, and resources that represent a potential service.

CALL TREE (A.K.A CALL GRAPH)

A control flow graph that represents calling relationships between functions within a service.

ENTRY POINT

A method at the root of the service's call tree.

EXCLUSIVITY

The % of resources used within the service that are accessed exclusively from within that service.

EXTRACTED RUNTIME

The % of CPU time spent in functions that are called from within the services. The remaining time is spent in functions that were not assigned to any service.

CLASS EXCLUSIVITY

The % of classes found during learning or during static analysis that are used exclusively from within a single service.

EXCLUSIVE CLASSES

These classes are used exclusively from a single service.

NON-EXCLUSIVE CLASSES

These classes are called from several services.

RESOURCES

Objects that potentially pose constraints to extracting a service, or hint to a certain domain of a service (e.g. locks, database transactions, Java Beans, etc.)

RESOURCE EXCLUSIVITY

The % of resources found during learning that are accessed exclusively from within a single service.

EXCLUSIVE RESOURCES

These resources are accessed exclusively from a single service.

NON-EXCLUSIVE RESOURCES

These resources are accessed from several services.