IBM Agile Lifecycle Manager
Version 2.2.0

# *Installation, Administration and User Guide*
*June 26, 2020*

SC28-3145-00

**Note**

Before using this information and the product it supports, read the information in "Notices" on page 103.

This edition applies to Version 2.2.0. of IBM® Agile Lifecycle Manager (product number 5737-E91) and to all subsequent releases and modifications until otherwise indicated in new editions.

# Contents

# Preface

This PDF document contains topics from the Knowledge Center in a printable format.

## Version 2.2 release notes

**26 June 2020:** The IBM Agile Lifecycle Manager Installation Guide has been updated.

For a summary of new features and enhancement, see Chapter 1, “New features overview,” on page 1.

**Updated planning documents**

Updated hardware requirements:
“Hardware requirements” on page 17

Updated software requirements:
“Software prerequisites” on page 17

# Chapter 1. New features overview

IBM Agile Lifecycle Manager Version 2.2.0 is available on Red Hat OpenShift Container Platform 4.3 or later running IBM Common Services 3.4 or later.

A number of new features and enhancements have been introduced in IBM Agile Lifecycle Manager Version 2.2.0.

- **Support for Red Hat OpenShift Container Platform 4.4**
- **Support for Red Hat OpenShift Container Platform on Amazon Web Services**
- **Dynamic storage provisioning**

  A storage class can be specified during the deployment. This enables StatefulSets to use dynamically provisioned persistent storage of any type supported by Red Hat OpenShift Container Platform.
- **Custom certificates and keys**

  Custom certificates and keys can be provided to the deployment.
- **Run with the restricted Security Context Constraint**

  Microservices run as an arbitrary user on the Red Hat Openshift Container Platform restricted Security Context Constraint.
- **Service Account**

  A Service Account is created to be used by IBM Agile Lifecycle Manager pods.
- **Ability to create clones of Assembly Designs**

  From the IBM Agile Lifecycle Manager User Interface Assembly Designer it is now possible to clone an existing Assembly Design giving it a new Name/version. The Assembly Design is cloned in its entirety including the set of Doki scripts if applicable.
- **Improved Design visibility/navigation on canvas for large designs**

  In addition to the persisting of view location and zoom levels added in the previous release a new "mini-map" showing the location of the view on the overall design canvas and the location of each of the assembly components on the canvas so that the user can more easily navigate/zoom to the desired portion of the design.
- **Enhanced Resource Manager Driver API supporting full end to end lifecycle execution**

  The Resource Manager Driver API has been enhanced to allow a driver to support both the VIM and xNF/Resource tasks
- **Ability to suspend users via the "Suspended" LDAP Group**

  It is possible to suspend users by adding them to an LDAP group called "Suspended". If a user is added to this group, attempts to access resources will result in an authorisation failure and inability to login/ refresh an access token.
- **Extension of the Resource Descriptor allows Resource Manager Driver selection at the lifecycle transition and operation levels**

  Prior to IBM Agile Lifecycle Manager v2.2.0, a Resource Manager Driver was selected by the Resource Manager based on the infrastructure type. This has been enhanced to allow Resource Descriptors to identify the preferred driver for individual Lifecycle Transitions. This extends the flexibility of the system from the perspective of the xNF/Resource designer who can leverage different Resource Manager Drivers to perform Lifecycle Transitions for a given xNF/Resource.

## Benefits

Using IBM Agile Lifecycle Manager you can design and integrate external resources into virtual production environments and then automate the management of end-to-end lifecycle processes. This approach is known as 'network function virtualization'. This section elaborates on the benefits of this approach and the key functionality offered by IBM Agile Lifecycle Manager, and also provides you with case study material.

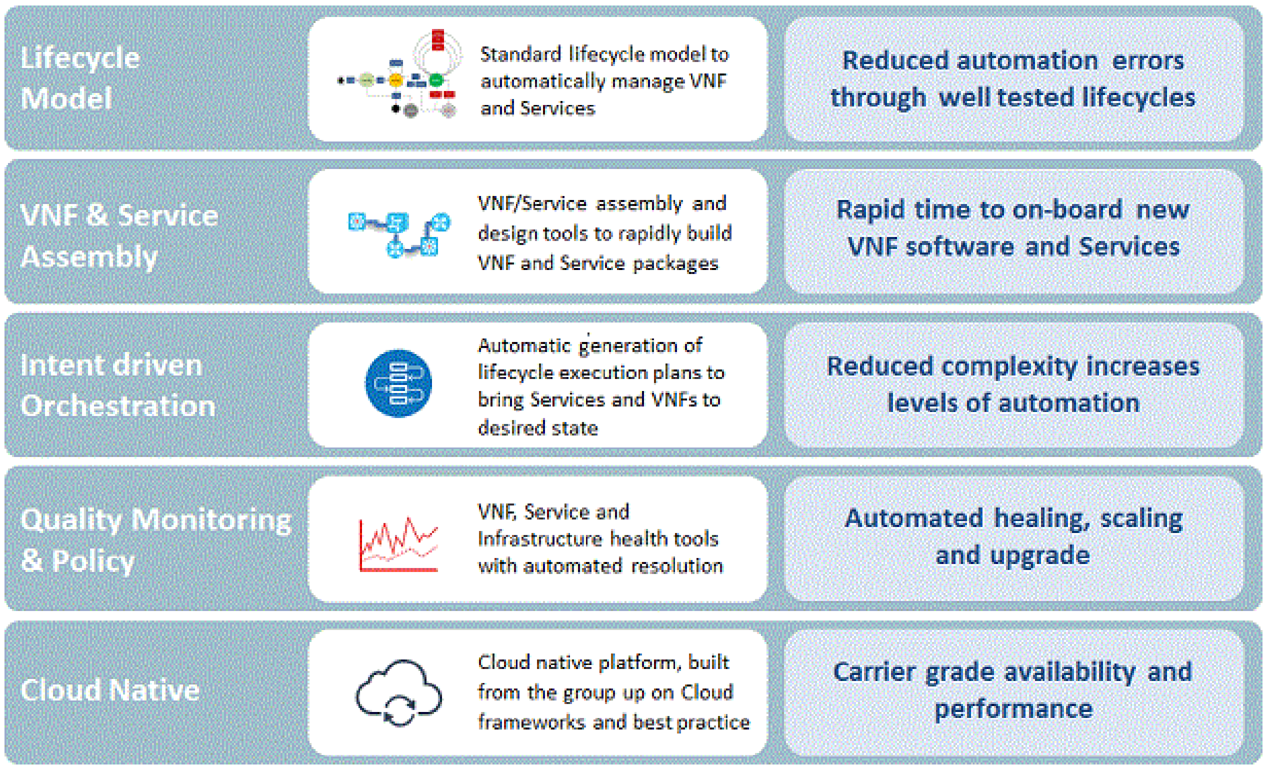
## Benefits of network function virtualization (NFV)

NFV brings a significant operational paradigm shift for service providers. Today's physical network appliances require highly manual processes to manage their end-to-end lifecycle. Testing, installation, configuration, and problem management of network appliances all revolve around manual activities that often require a physical truck roll or a human to run each lifecycle process.

NFV's software paradigm promises fully automated lifecycle processes for bringing network services into production and maintaining them thereafter. Virtual network functions (VNF) allow a much simpler set of lifecycle tasks enabling near full automation of the creation and healing of virtual services, far more than is possible with their physical counterparts.

There are vast business opportunities associated with NFV transformation including new revenue streams, improved customer experience and reductions in both operational and capital expenditure. However, a fully automated lifecycle solution for NFV comes with additional complexities.

IBM Agile Lifecycle Manager is a comprehensive services design, testing and automated deployment platform addressing the challenges and complexities of the NFV paradigm. It delivers an end-to-end automated service lifecycle solution from initial design to production, as depicted in the following figure:

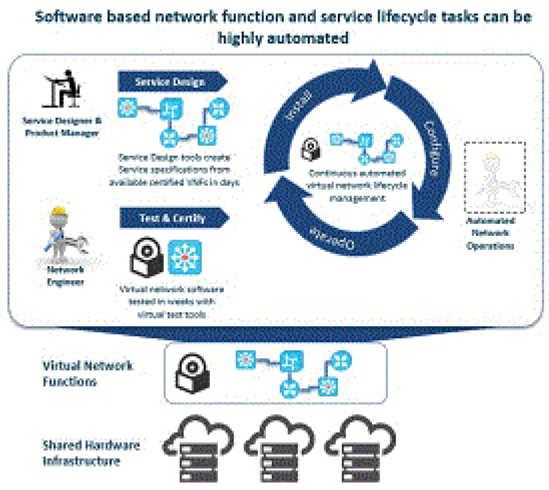


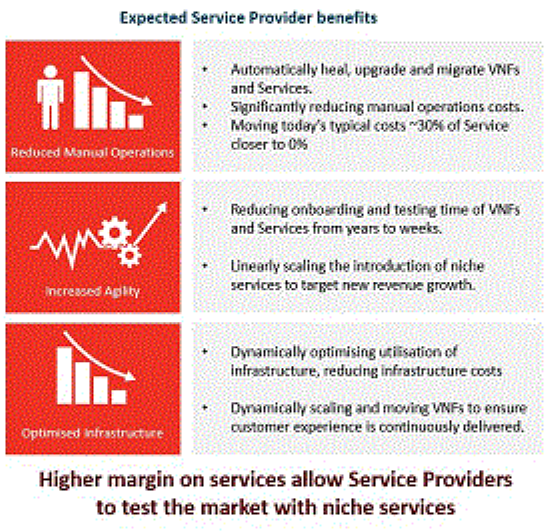
| | | |
|---|---|---|
| Lifecycle Model | Standard lifecycle model to automatically manage VNF and Services | Reduced automation errors through well tested lifecycles |
| VNF & Service Assembly | VNF/Service assembly and design tools to rapidly build VNF and Service packages | Rapid time to on-board new VNF software and Services |
| Intent driven Orchestration | Automatic generation of lifecycle execution plans to bring Services and VNFs to desired state | Reduced complexity increases levels of automation |
| Quality Monitoring & Policy | VNF, Service and Infrastructure health tools with automated resolution | Automated healing, scaling and upgrade |
| Cloud Native | Cloud native platform, built from the group up on Cloud frameworks and best practice | Carrier grade availability and performance |

The key differentiating features of IBM Agile Lifecycle Manager depicted in this figure include:

- A common way of handling resources through a unified lifecycle model
- Support for quick resources and assembly on-boarding
- Intent-driven lifecycle management
- Quality management and policy modules
- Cloud-native solution

The benefits of using IBM Agile Lifecycle Manager to deliver NFV are illustrated in the following figure:


Software based network function and service lifecycle tasks can be highly automated
Service Designer & Product Manager
Service Design
Service Design tools create Service specifications from available certified VNFs in days
Network Engineer
Test & Certify
Virtual network software tested in weeks with virtual test tools
Install
Configure
Operate
Continuous automated virtual network lifecycle management
Automated Network Operations
Virtual Network Functions
Shared Hardware Infrastructure


Expected Service Provider benefits
Reduced Manual Operations
Increased Agility
Optimised Infrastructure

- Automatically heal, upgrade and migrate VNFs and Services.
- Significantly reducing manual operations costs.
- Moving today's typical costs ~30% of Service closer to 0%

- Reducing onboarding and testing time of VNFs and Services from years to weeks.
- Linearly scaling the introduction of niche services to target new revenue growth.

- Dynamically optimising utilisation of infrastructure, reducing infrastructure costs
- Dynamically scaling and moving VNFs to ensure customer experience is continuously delivered.

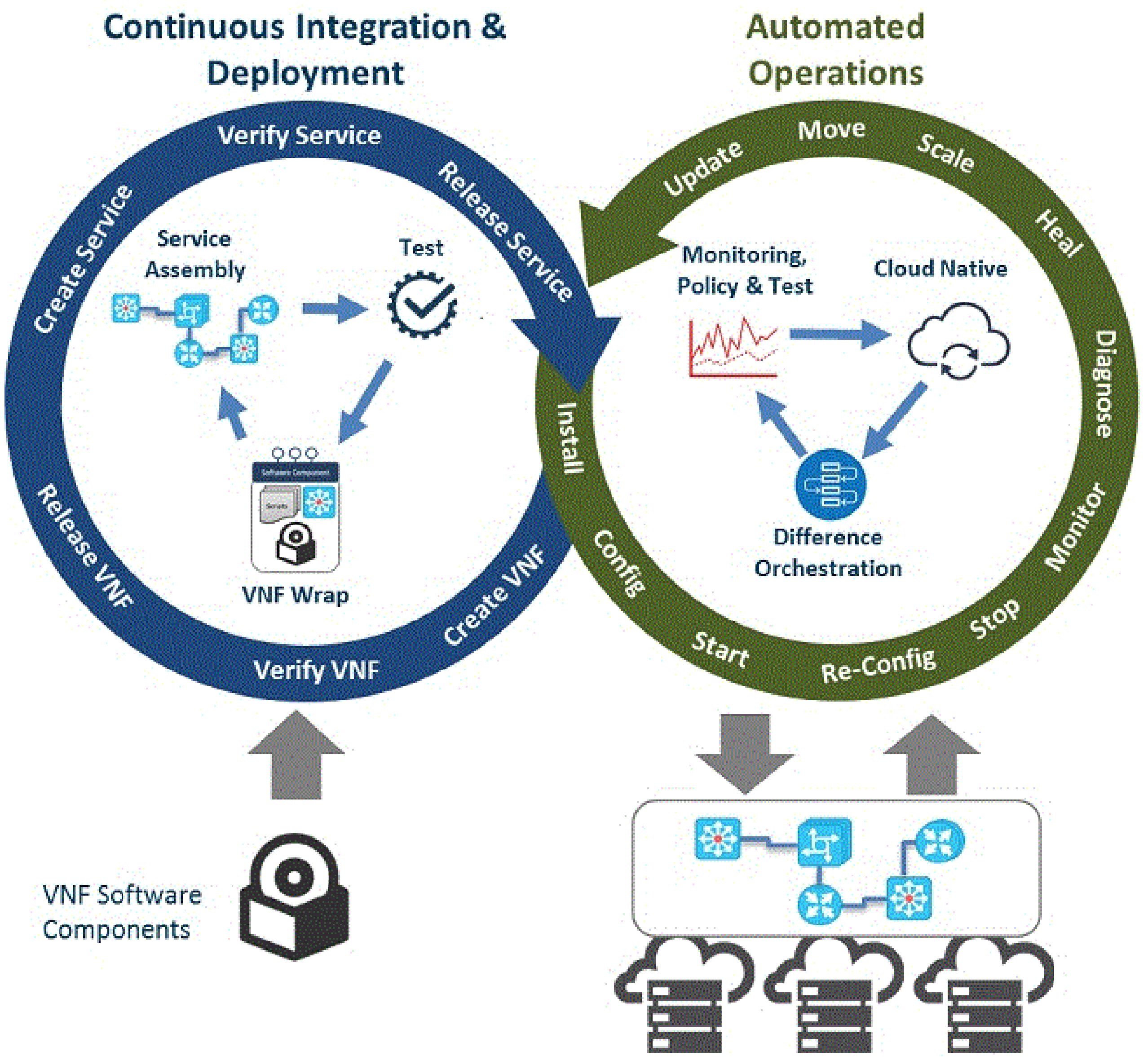
Higher margin on services allow Service Providers to test the market with niche services

## Lifecycle management

To achieve NFV's promised levels of automation, IBM Agile Lifecycle Manager provides a complete DevOps toolchain that manages the end-to-end lifecycle of virtual network services, from release management of VNF software packages to the continuous orchestration and running of VNFs and service instances.

Third party VNF software must be wrapped in a well-tested standard lifecycle interface, and service bundles of multi-vendor VNFs must be tested for interoperability and performance to ensure that there are no errors. This ensures that, once in production, services and VNFs can be constantly created, configured, updated, scaled, healed, and migrated without manual intervention.

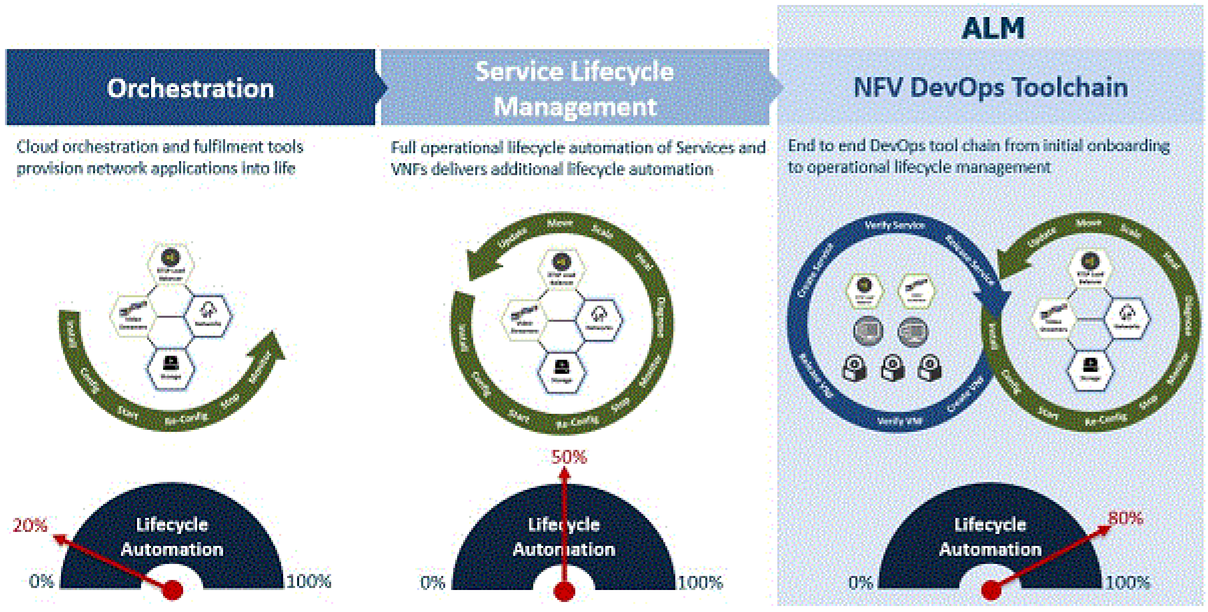
The complete lifecycle management is shown in the following figure:



VNF and services onboarding requires a comprehensive release management strategy, a suite of tools and a lifecycle integration framework to accommodate the variety of third party VNF software package formats. Continuous in-life orchestration also requires a new approach to modelling and managing the complexity of in-life network function lifecycle management. To achieve the levels of automation required, a much simpler and standardized approach is required to implement all foreseen lifecycle transitions.

### Advantages of an end-to-end DevOps deployment model

The advantages of adopting an end-to-end DevOps deployment model are illustrated in the following figure:


Orchestration
Service Lifecycle Management
ALM
NFV DevOps Toolchain
Cloud orchestration and fulfilment tools provision network applications into life
Full operational lifecycle automation of Services and VNFs delivers additional lifecycle automation
End to end DevOps tool chain from initial onboarding to operational lifecycle management
Lifecycle Automation
20%
50%
80%
0%
100%

## Case study: Heal operation using IBM Agile Lifecycle Manager

In the following 'closed loop heal' case study, a server no longer receives IP packets. An alarm is raised, the event is evaluated, and a Heal event is triggered and executed by IBM Agile Lifecycle Manager. The solution addressing this case study is comprised of the following components (some of which are IBM Telco Network Cloud Manager - Assurance):

- IBM Tivoli Netcool/OMNIbus and Web GUI
- IBM Tivoli Netcool/Impact
- IBM Netcool Agile Service Manager
- IBM Agile Lifecycle Manager

### Heal solution process, step-by-step

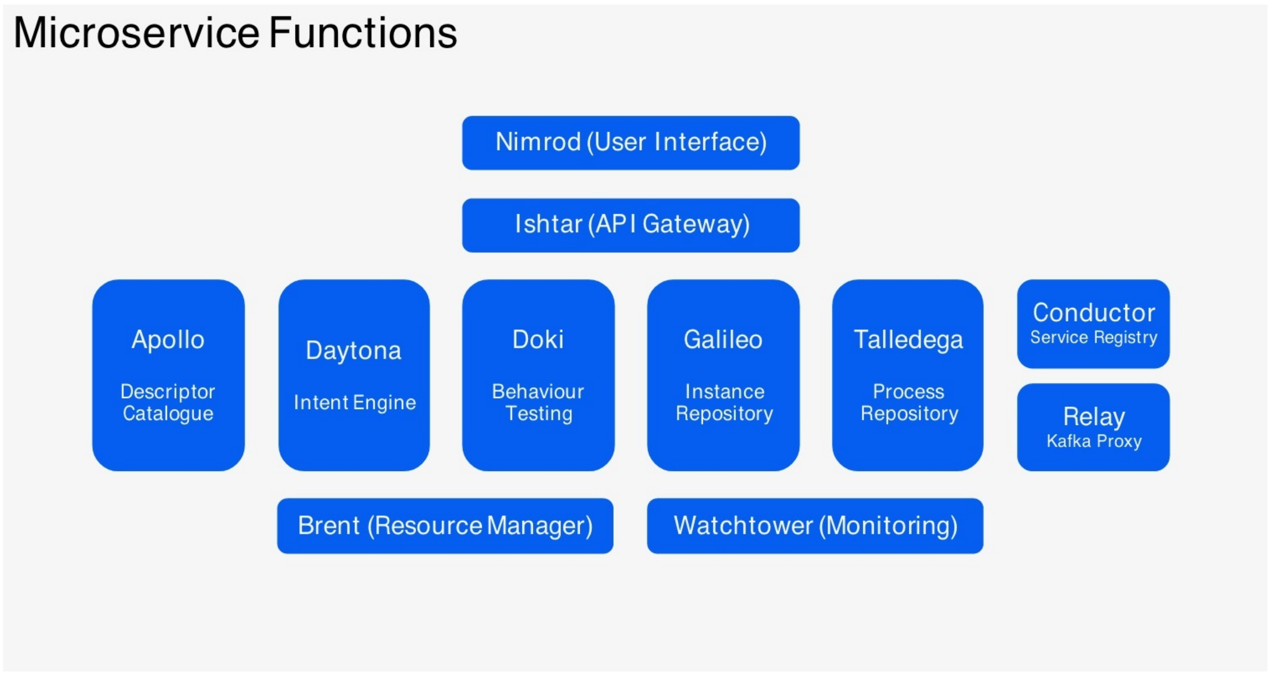
1. Tivoli Netcool/OMNIbus (acting as **external OSS**) receives an alarm that a server has stopped receiving IP packets.
2. This alarm then triggers a Tivoli Netcool/Impact policy.
3. As part of the service design and assembly onboarding, Netcool Agile Service Manager reconciles the IBM Agile Lifecycle Manager assemblies with OpenStack virtual machines.
4. Tivoli Netcool/Impact (acting as **operations**) now has enough knowledge to trigger an IBM Agile Lifecycle Manager Heal request.
5. IBM Agile Lifecycle Manager transitions the 'broken' server through the following lifecycles:
   a. STOP
   b. START
   c. INTEGRITY
6. Netcool Agile Service Manager gets notified by IBM Agile Lifecycle Manager that a lifecycle event has occurred on the 'broken' server.
7. This triggers the Netcool Agile Service Manager observer that raised the alarm to rerun.
8. Rerunning the observer clears the Tivoli Netcool/OMNIbus alarm, confirming that the server is receiving packets again, and 'healing' was successful.

## Architecture

This topic provides an overview of the IBM Agile Lifecycle Manager architecture.

### Basic architecture

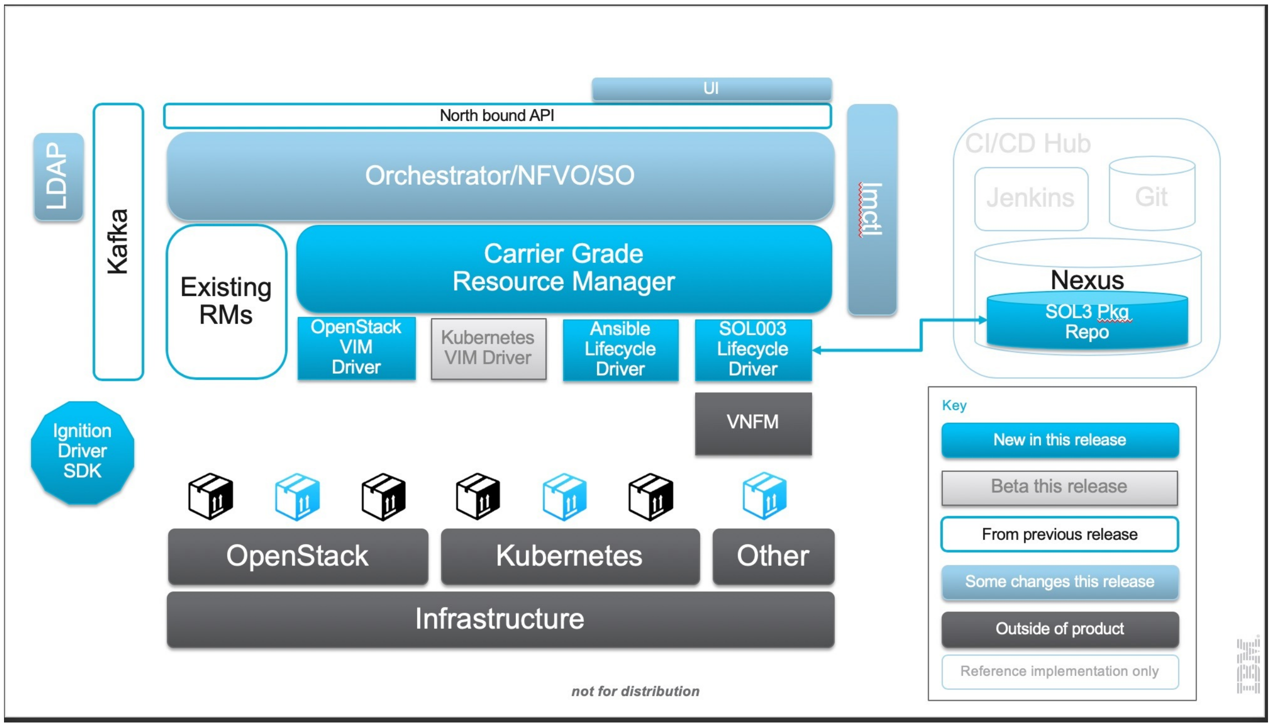
The basic IBM Agile Lifecycle Manager architecture is depicted in the following figure:



### Data flow

IBM Agile Lifecycle Manager receives requests through its north bound API to put an assembly instance representing a VNF or service into an intended state, such as 'active'. IBM Agile Lifecycle Manager in turn orchestrates all external resource managers through their northbound APIs to configure their managed resource instances accordingly. Throughout this orchestration period IBM Agile Lifecycle Manager and each resource manager publish orchestration status events to a Kafka topic for each assembly and resource instance state change.

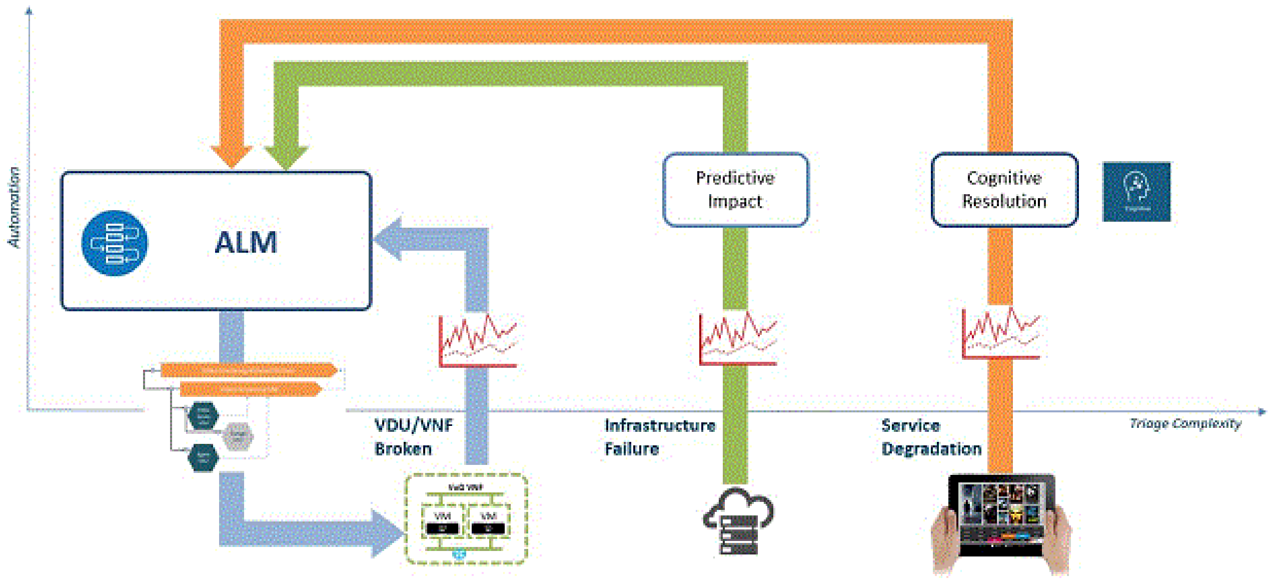
This process is depicted in the following figure:



Resource managers are responsible for orchestrating virtual infrastructure managers (VIM) to control cloud infrastructure compute, storage and network resources in support of their resource instances' standard lifecycles.

## Extending IBM Agile Lifecycle Manager

In addition to published orchestration events, performance and quality metrics may be published to a dedicated monitoring Kafka topic. These are consumed by automatic scaling and healing policies configured in IBM Agile Lifecycle Manager. IBM Agile Lifecycle Manager can be integrated with various external systems responsible for monitoring different aspects of an end-to-end service. These external systems, such as SQM or other assurance and analytics systems, can thus be extended to perform the following types of healing using the IBM Agile Lifecycle Manager northbound API:



**Broken virtual resources**
: Resource instances that are not performing within acceptable levels are identified as broken and healed by IBM Agile Lifecycle Manager.

**Infrastructure failure**
The impact on resource instances by a physical infrastructure is reported and appropriate healing or migration is performed by the IBM Agile Lifecycle Manager.

**Service degradation**
Reduced customer experience or service quality can trigger scaling events.

## Functionality

IBM Agile Lifecycle Manager provides continuous integration and deployment of resources, intent-driven operations to automate lifecycle processes, and an open framework.

This section introduces key concepts that are useful to understand how IBM Agile Lifecycle Manager works and the key drivers behind the design.

### Unified lifecycle model

The standard lifecycle model is a key enabling factor for intent-based automation. To drive automation, IBM Agile Lifecycle Manager uses a standardized lifecycle model which requires all the underlying artifacts, network services and virtualized network functions (VNFs), to look the same to IBM Agile Lifecycle Manager. Each artifact of a network service will be expected to go through the same lifecycle with defined lifecycle transitions.

Each artifact in a network service is optimally stepped through this lifecycle concurrently with others accounting for the declaratively identified dependencies. For example, on installation each artifact will be moved to the *Installed* state together before they are moved to the *Inactive* state. Only the lowest level artifacts (VNF components) have transition steps defined that will be performed as the network service progresses through the lifecycle.

### Intent-based automation

Intent-based automation is comparable to a satellite navigation system. In order to go from A to B, you only need to enter the destination and the satnav works out the best way to get there. You do not need to manually program each turn. If there is a roadblock or a traffic jam, the engine works out the best alternative route without you having to manually reprogram the route.

IBM Agile Lifecycle Manager works in a similar way. Rather than programming all individual lifecycles for all possible lifecycle scenarios (such as upgrade and migration) the engine automatically generates and executes all lifecycles for VNFs and network services.

The intent engine (one of the core micro-services) in the heart of IBM Agile Lifecycle Manager has the responsibility for driving network service instances through their lifecycles according to the intent requests it receives. With support from other IBM Agile Lifecycle Manager services, like catalog and topology, the intent engine resolves the necessary network service level actions, breaks them down to VNF and eventually to VNF component level lifecycle transitions, and works out the correct order to execute the steps. The exact sequence of execution steps depends on the model of the network service, the current state of the service, and the requested target state and shape of the service. Finally, the intent engine pushes the corresponding requests to the responsible resource managers to complete through IBM Agile Lifecycle Manager's southbound APIs.

The detailed execution plans are stored by IBM Agile Lifecycle Manager to enable real-time monitoring of the progress of currently active intents as well as browsing the intent history for a specific network service instance.

Intents can be requested through a set of API endpoints on the IBM Agile Lifecycle Manager northbound API. Requests can be triggered by an external northbound system, by IBM Agile Lifecycle Manager itself for policy-driven intents (such as auto-healing or scaling for example) or they can be manually requested through the IBM Agile Lifecycle Manager user interface which then calls the corresponding API end point.

### Behavior-driven testing

IBM Agile Lifecycle Manager has in-built behavior-driven testing capability functioning as an automated test framework for VNF and network service testing. While intent-driven automation automates the production lifecycle of your network services and associated resources, behavior-driven testing, together with CI/CD Hub, helps you to automate your service development processes. Automated behavior-driven testing reduces manual effort and errors in production.

In development and preproduction phase, IBM Agile Lifecycle Manager’s behavior-driven testing can automatically spin up test environments, network service under test, and additional test VNFs such as probes and traffic generators, and run through full test scenario of all lifecycle states.

While services are deployed in production environment, ready-for-service testing can be invoked automatically so that the network service is first tested in full before being released to the customer.

Once designed, the test scenarios will be part of the VNF or network services package. Test scenarios can be either run directly from IBM Agile Lifecycle Manager’s user interface or triggered through its APIs enabling full automation and integration to CI/CD Hub pipelines.

### CI/CD Hub

Cloud DevOps best practices and principles are at the heart of the IBM Agile Lifecycle Manager solution. To scale any cloud-based networking program, a unified operations and engineering model should be combined with a set of automation tools that can simplify and automate the complexities of an end-to-end VNF or network service lifecycle.

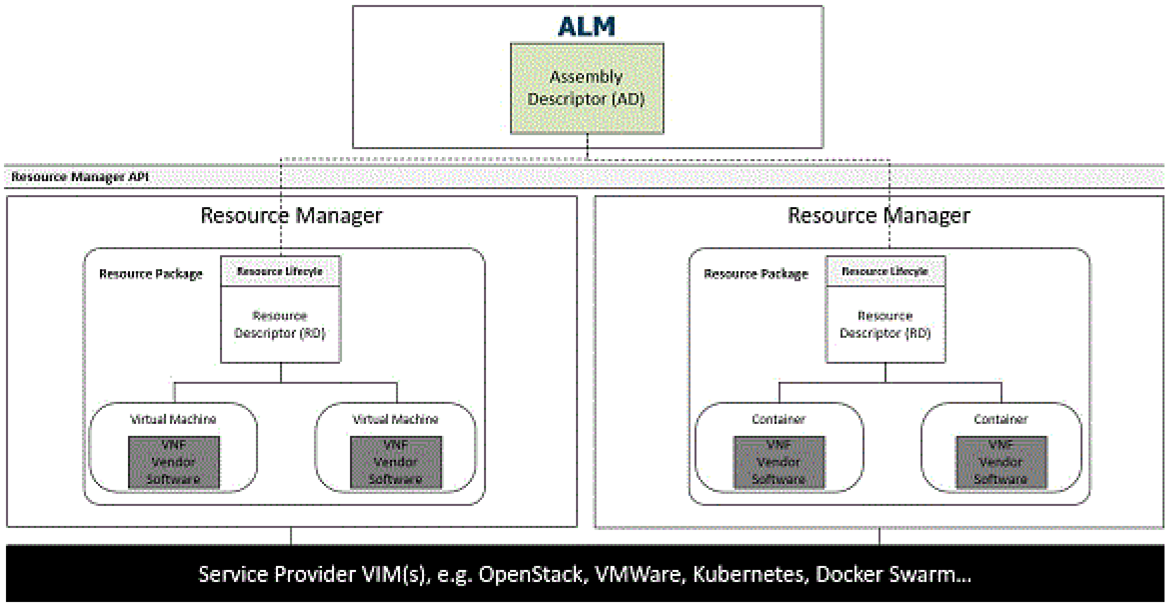
The IBM Agile Lifecycle Manager CI/CD tools and processes are designed to simplify and automate the following DevOps tasks:

- **Onboard VNFs and design network services**: Quickly integrate and package the lifecycle actions required to operate a VNF or any virtual or physical network appliance.
- **Behavior-driven testing in preproduction**: Deploy VNFs to preproduction environments and easily script complex operational behavior tests to ensure the onboarded VNF behaves as expected in all Day-1 or Day-2 lifecycle tasks.
- **Deploy to production**: Once fully tested, auto-deploy to production environments.
- **Monitor and Change**: Monitor and sense environmental or VNF state and auto scale, heal or move components of the network service.
- **Report and Resolve issues**: Errors in lifecycle actions or VNF software found in production are reported and trigger an upgrade process that rebuilds a new version of the VNF.

To learn more about CI/CD Hub and applying DevOps processes on network service and VNF lifecycle management see: “Using the CI/CD Hub” on page 96.

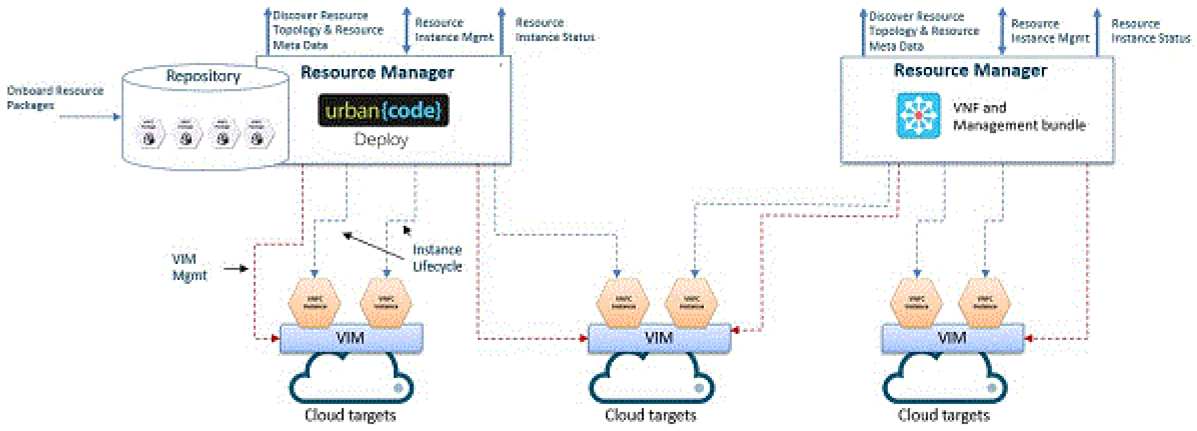
### Resource Manager framework

The Resource Manager framework provides an open set of tools to allow VNF vendors to wrap their software in a standardized lifecycle that can be manipulated by IBM Agile Lifecycle Manager. Proprietary VNF managers or general-purpose software managers, such as Canonical's JuJu Charms, IBM's Urbancode, or RedHat's Ansible, can be integrated to allow the virtual or physical resources they manage to be discovered and manipulated by IBM Agile Lifecycle Manager.



Resource managers adhere to an API that allows IBM Agile Lifecycle Manager and other external systems to discover the data center topology supported by the resource manager instance and the resource types it supports. The API also provides the ability to manipulate and monitor the state and health of each resource instance.

The following figure depicts IBM Agile Lifecycle Manager's ability to manipulate resources from many resource managers. Each resource manager manages the lifecycle of several virtual or physical devices and the underlying virtual or physical infrastructure.



Resource managers must abide by the following use cases and requirements:

- Multiple Resource Managers can manage resource instances on the same VIM(s)
- Resource Managers can manage one or more VIMs
- Component instances are managed by a single Resource Manager
- VNF types are registered with Resource Manager types
- Resource Manager instances are registered to work with one or more VIMs
- Instances of VNF types can be deployed to multiple VIMs by the registered Resource Manager VIMs
- Multiple Resource Managers can manage VNF types on a single VIM

The Resource Manager framework includes the following artifacts to support the above use cases and requirements:

**Swagger API definition and specification**
Rest and Kafka API semantics and messages

**Resource descriptor specification**
    Descriptor specification

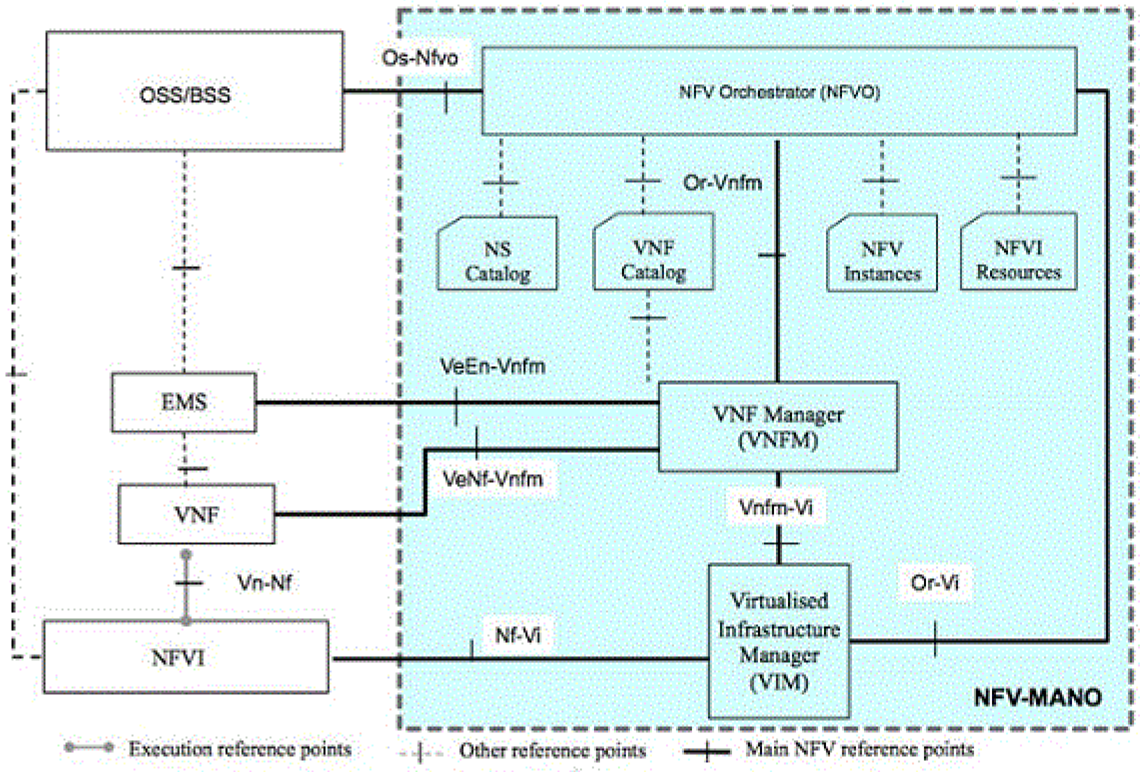
**Resource archive format**
    Standard and portable packaging format for bundling software, lifecycle and operation scripts, descriptors and resource manager configuration

**API drivers**
    Integrations for popular software management systems

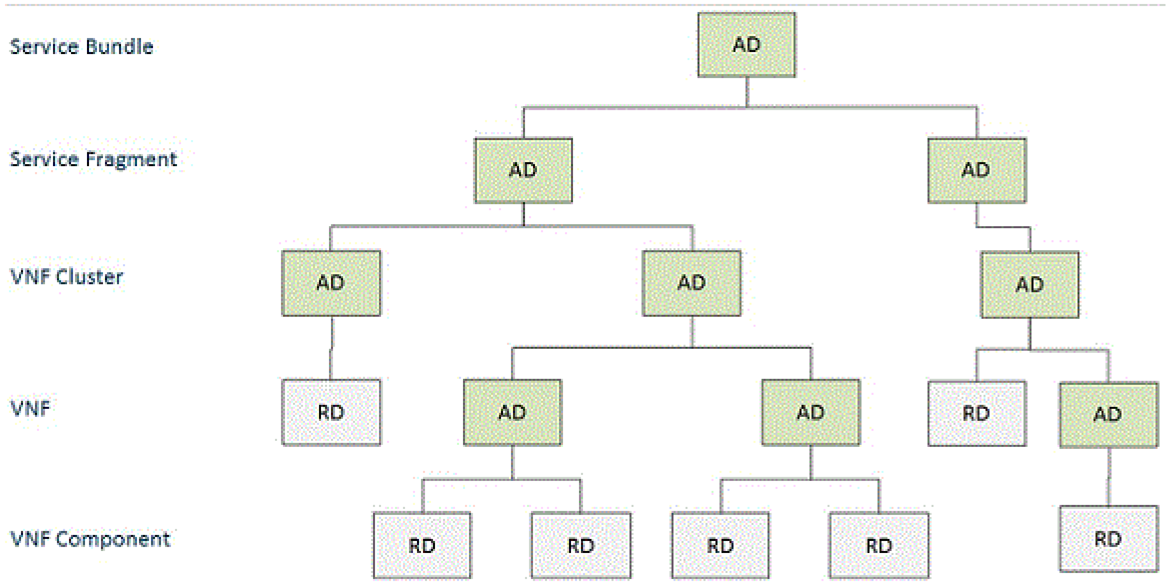
**Resource Manager API:** The Resource Manager API is responsible for defining the interactions between a lifecycle manager and the resource managers used to manage resources within virtual (or physical) infrastructures.

## Mapping to industry standards

In recent years the industry has been actively generating standards for NFV. NFV specifications are published on a regular basis by various industry forums. Those include ATIS, Broadband Forum, ETSI NFV ISG, IETF, ONF and others. In parallel, open source projects have been established to accelerate NFV adoption. The most recent and notable initiative within the orchestration area is the Open Network Automation Platform (ONAP) that is joining two projects: The Enhanced Control, Orchestration, Management and Policy (ECOMP) and the Open Orchestrator (Open-O). IBM closely monitors relevant forums and partners with key industry contributing members.



ETSI has introduced a number of key concepts that provide a language for describing an NFV environment. The following figure shows a mapping of IBM Agile Lifecyle Manager concepts to ETSI definitions. ETSI terminology/concepts are shown on the left of the figure mapped to IBM's core concepts, that is, Assembly Descriptor (AD) and Resource Descriptor (RD).



How a VNF vendor has engineered their software will determine how many resource descriptors it presents to IBM Agile Lifecyle Manager. For example, a native Cloud style VNF implementation could provide many Resource Descriptors representing micro-services that are assembled into VNF components, which are in turn assembled into VNFs.

Conversely, a VNF vendor may provide a single resource representing a complete VNF function. These resource descriptors in any case are re-assembled into an architecture specific to the service providers' environment and service design, once again by layering assembly descriptors and relationships.

## Glossary

Refer to the following list of terms and definitions to learn about important IBM Agile Lifecycle Manager concepts.

### IBM Agile Lifecycle Manager terminology

**assembly**
: An assembly is a definition of a service and may comprise of one "resource (or component)" on page 14, (or more than one resource), and/or other assemblies.

  It is defined in an assembly descriptor and can be instantiated as an assembly instance.

**assembly descriptor (or descriptor)**
: An assembly descriptor is a computer-readable definition of an assembly implemented as a YAML file.

**assembly designer (or service designer)**
: An actor or end user role designing services using IBM Agile Lifecycle Manager. An assembly designer takes informal service design artifacts defined by service designers and translates them to a set of formal computer-readable descriptors that model the target service.

**assembly instance**
: Instantiation of an assembly descriptor and all the composed resources or assemblies.

**capability**
: Capabilities is a section of an assembly descriptor or a resource descriptor defining what functions the resources or assemblies are implementing.

**catalog (or assembly catalog)**
: The repository within IBM Agile Lifecycle Manager storing published assembly descriptors and resources descriptors.

**cloud**
The cloud is a common term referring to accessing computer, information technology (IT), and software applications through a network connection, often by accessing data centers using wide area networking (WAN) or internet connectivity.

**CSAR (or archive)**
Cloud service archive (CSAR) describes a format used for describing resource packages.

CSAR specification is a part of OASIS TOSCA.

**deployment location**
Deployment location is a facility where resources can be deployed while they are instantiated.

In various contexts deployment locations are referred to as data centre, project (OpenStack), or availability zone (OpenStack).

**descriptor**
See "assembly descriptor (or descriptor) " on page 12

**forwarding graph**
See service chain.

**intent engine (or engine)**
The entity responsible for generating the assembly deployment plan and instructing, step by step, resource managers to execute the plan.

**Kafka**
Apache Kafka is a distributed streaming platform.

**Tip:** Streams of records are stored in Kafka in categories called 'topics'.

**lifecycle event (or event)**
IBM Agile Lifecycle Manager published intent and status change event onto a Kafka topic.

**lifecycle state (or state)**
A lifecycle state defines the state of a specific resource instance or assembly instance.

Examples of lifecycle states include: Installed, Inactive, Active, Broken, and Failed.

Changes from one lifecycle state to another are lifecycle transitions.

**lifecycle transition (or transition)**
A lifecycle transition is a process aiming to change the lifecycle state of an assembly or resource.

Lifecycle transitions are initiated through the IBM Agile Lifecycle Manager API, orchestrated by IBM Agile Lifecycle Manager and executed by the underlying resource managers.

Examples of lifecycle transitions include: install, configure, start, stop, integrity, and uninstall.

**microservice**
Microservices are a variant of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services.

The benefit of decomposing an application into different smaller services is that it improves modularity and makes the application easier to understand, develop and test.

It also parallelizes development by enabling small autonomous teams to develop, deploy and scale their respective services independently.

**migration**
Migration is one of the opinionated patterns aiming to migrate a deployed NVF from a location to another.

**monitoring metrics**
Performance or health metrics published by resource managers and/or resources onto a Kafka topic.

**network**
A type of resource.

**network function virtualization (NFV)**
The design and integration of external resources into virtual production environments, which can then automate the management of end-to-end lifecycle processes.

Also see virtual network functions.

**OASIS**
OASIS is a non-profit consortium that drives the development, convergence and adoption of open standards for the global information society.

**onboarding**
Onboarding is the act of adding a resource manager to IBM Agile Lifecycle Manager. It lets IBM Agile Lifecycle Manager know that the resource manager exists, and it imports the descriptors of all the resource types managed by the resource manager. It also gathers the information about the deployment locations that the resource manager uses.

**operations**
'Operations' is a section of an assembly descriptor or a resource descriptor that defines sets of operations, which can be called to enable relationships to be created between resources and/or assemblies.

**opinionated patterns**
The group of lifecycle transitions to achieve a particular task.

Examples of tasks include: heal, reconfigure, and upgrade.

**policies**
'Policies' is a section of an assembly descriptor or a resource descriptor containing the set of policies that are used to manage the assembly or resource instances.

**property**
'Properties' is a section of an assembly descriptor or a resource descriptor containing the properties that belong to the resource or assembly descriptors.

These include the full set of properties that are required to orchestrate them through to the active state.

These can be understood as the 'context' for the management of the item during its lifecycle.

**quality monitoring**
Quality Monitoring is a process to monitor the health of deployed resources and NFV infrastructure and to test, monitor and evaluate the end-to-end service performance.

**reference**
'Reference' is a section of assembly descriptor or resource descriptor.

When the IBM Agile Lifecycle Manager has already instantiated an assembly it is possible for another assembly to share the instance by referencing it within the references section.

The references section can also refer to existing objects that may have been created outside the IBM Agile Lifecycle Manager.

**relationship**
'Relationships' is a section of assembly descriptor or resource descriptor.

Relationships define how the descriptors link requirements to capabilities.

A relationship has source-capabilities and target-requirements as parts of its description.

**requirement**
'Requirements' is a section of an assembly descriptor or a resource descriptor explaining what functions the resources or assemblies need before they can work successfully.

**resource (or component)**
A piece of software that can be automatically deployed in a virtual environment, and that supports key lifecycle states including install, configure, start, stop, and uninstall.

**resource descriptor**
The list of resource attributes and properties written in YAML.

**resource health (or component health)**
Resource health is a microservice within IBM Agile Lifecycle Manager responsible for monitoring health-related messages and initiating recovering actions related to deployed resources.

For example, the resource health may send a 'heal' message to the intent engine if a certain event indicating health issues is detected.

**resource instance**
A resource instance represents the logical grouping of infrastructure being managed by an external resource manager.

**resource manager**
The entity instructing resources.

For example, IBM UrbanCode.

**resource manager record**

IBM Agile Lifecycle Manager maintains a record of each resource manager it can use to create and manage resources. When a new resource manager record is created, the resource types managed by that resource manager instance are read into IBM Agile Lifecycle Manager using the resource manager's API.

**resource package**
Resource package is described as a CSAR archive.

This is the bundle of everything needed for a resource that is loaded into a resource manager.

**scale**
Scale is one of the opinionated patterns aiming to increase or decrease the amount of deployed resources of a specific type.

**service chain**
Instantiated as relationships in assembly descriptors.

**TOSCA**
Topology and orchestration specification for cloud applications (TOSCA) is a standard defined by OASIS.

**topology (or instance inventory)**
The repository storing key state information related to assembly and resource instances and topology of the deployment locations.

**virtual infrastructure manager**
The entity controlling the cloud infrastructure compute, storage and network resources.

For example, OpenStack.

**virtual network functions (VNF)**
Virtual network functions (VNF) allow a much simpler set of lifecycle tasks enabling near full automation of the creation and healing of virtual services, far more than is possible with their physical counterparts.

Also see network function virtualization.

# Chapter 2. Planning

This section helps you to plan your installation and use of IBM Agile Lifecycle Manager by listing the minimum software and hardware requirements for the Red Hat OpenShift Container Platform version of IBM Agile Lifecycle Manager.

## Hardware requirements

This section lists the minimum hardware requirements for a deployment of IBM Agile Lifecycle Manager.

Your minimum hardware requirements are determined by the needs of the components of your specific solution. The requirements listed here focus on what you need to deploy each IBM Agile Lifecycle Manager instance.

On Red Hat OpenShift Container Platform, IBM Agile Lifecycle Manager requires a minimum of three worker nodes with a combined specification as shown in the following table.

*Table 1. IBM Agile Lifecycle Manager hardware requirements (spread across a minimum of three worker nodes)*

| Requirement | Setting |
|---|---|
| CPU | 72 cores |
| Memory | 96 GB of RAM |
| Disk space | 1.5 TB |

In addition to this, IBM Common Services requires the following resources on Red Hat OpenShift Container Platform compute or worker nodes.

*Table 2. IBM Common Services hardware requirements*

| Requirement | Setting |
|---|---|
| CPU | 8 cores |
| Memory | 32 GB of RAM |
| Disk space | 100 GB |

**Note:**

The hardware requirements listed for IBM Common Services satisfy only the requirements for the minimum Operands necessary for IBM Agile Lifecycle Manager. If additional IBM Common Services Operands are required (for use by other products), please refer to the sizing guidance for IBM Common Services, see the following Knowledge Center page: https://www.ibm.com/support/knowledgecenter/SSHKN6/installer/3.x.x/hardware_sizing_reqs.html

## Software prerequisites

This section lists the software prerequisites for a deployment of IBM Agile Lifecycle Manager.

IBM Agile Lifecycle Manager has the following software prerequisites.

*Table 3. IBM Agile Lifecycle Manager software prerequisites*

| **Prerequisites** |
|---|
| Red Hat OpenShift Container Platform 4.3 or later |
| Docker for Red Hat Enterprise Linux Version 1.12 or later<br>(Required only on the boot node of Red Hat OpenShift Container Platform where the install is carried out from.) |
| IBM Common Services 3.4 or later with the following Operands installed:<br>`ibm-platform-api-operator`<br>`ibm-helm-api-operator`<br>`ibm-helm-repo-operator`<br>`ibm-catalog-ui-operator` |
| openssl-1.0.2k or later |

**Note:** The IBM Agile Lifecycle Manager User Interface is only supported on the Google Chrome browser.

# Chapter 3. Installing and configuring

IBM Agile Lifecycle Manager is distributed as a self-contained package delivered as a Helm chart. To install IBM Agile Lifecycle Manager, you complete the required pre-installation tasks, then perform the installation.

## Before you install

Before installing IBM Agile Lifecycle Manager on Red Hat OpenShift Container Platform, you must install IBM Common Services.

For details about installing IBM Common Services, see Installing IBM Cloud Platform Common Services in your Red Hat OpenShift Container Platform cluster.

Helm and cloudctl should be installed. Refer to the CLI tools guide section of the IBM Cloud Platform Common Services documentation for more details.

## Installing IBM Agile Lifecycle Manager on Red Hat OpenShift Container Platform

This topic describes how to deploy IBM Agile Lifecycle Manager on Red Hat OpenShift Container Platform. It is required to perform the deployment on a non-default namespace. After the deployment, in order to be able to access the IBM Agile Lifecycle Manager User Interface, the IBM Agile Lifecycle Manager instance has to be integrated with an external Directory Service.

### Before you begin

Before you install IBM Agile Lifecycle Manager make sure you have met the following prerequisites:

- You must have downloaded the eAssembly.
- Make sure that clocks in each node of the cluster are synchronized. Refer to Red Hat OpenShift Container Platform documentation for details on how to synchronize the nodes.
- Ensure you have completed all prerequisite tasks.
- Ensure that the boot node from which you plan to deploy IBM Agile Lifecycle Manager has internet access.
- You must enable IPsec in the cluster to ensure all data in motion is encrypted.

You must install IBM Agile Lifecycle Manager into a **non-default namespace**, ensure that the user you are deploying with has adequate privileges to perform the installation.

### About this task

You install IBM Agile Lifecycle Manager 2.2.0 as cluster administrator.

**StatefulSet component storage**

StatefulSet component storage can be configured to use static provisioning for local storage or to use dynamic provisioning for any type of storage. The default storage configuration uses static provisioning of local storage, which requires local Persistent Volumes and Persistent Volume Claims to be created. Configuration scripts are provided to automatically set up the local Persistent Volumes and Persistent Volume Claims. Alternatively, if Dynamic Provisioning is to be used, this is set as part of the helm install command. The storage class to be used for Dynamic Provisioning is also set as part of the helm install command.

### Procedure

**Prepare the IBM Agile Lifecycle Manager environment**

**Restriction:** You **must** run the prerequisite scripts as cluster administrator.

1. Create a local folder on a boot node for the IBM Agile Lifecycle Manager images.

   ```
   mkdir ./<alm_archive_folder>
   cd ./<alm_archive_folder>
   ```

   Where `<alm_archive_folder>` is the local folder used for the IBM Agile Lifecycle Manager archive.

   The IBM Agile Lifecycle Manager archive file can be downloaded from the IBM Passport Advantage® website.

**Prepare and load installation assets**

2. Log in to the Red Hat OpenShift Container Platform cluster:

   ```
   oc login -s <openshift_server_url> -u <cluster_admin> -p <cluster_admin_pwd>
   ```

   Refer to Red Hat Openshift Container Platform documentation for more information regarding the login procedure.

3. If required, create a new namespace for IBM Agile Lifecycle Manager:

   ```
   oc create namespace <alm_namespace>
   ```

   Where `<alm_namespace>` is the namespace to be used for the IBM Agile Lifecycle Manager installation.

   **Note:** IBM Agile Lifecycle Manager can be installed in a pre existing non-default namespace.

4. Expose the Image registry on Red Hat OpenShift Container Platform:

   ```
   oc patch configs.imageregistry.operator.openshift.io/cluster --patch '{"spec":
   {"defaultRoute":true}}' --type=merge
   ```

   Check that the Image registry has been exposed with the following command:

   ```
   oc get route -n openshift-image-registry default-route -o jsonpath='{.spec.host}'
   ```

   This value will be used in the subsequent command where `<ocp_registry>` is referenced.

5. Trust the image registry certificate:

   ```
   mkdir -p /etc/docker/certs.d/<ocp_registry>
   openssl s_client -showcerts -servername <ocp_registry> -connect <ocp_registry>:443 2>/dev/
   null | openssl x509 -inform pem > /etc/docker/certs.d/<ocp_registry>/ca.crt
   ```

6. Add the Red Hat OpenShift Container Platform internal Helm repository to the Helm CLI:

   Obtain the route to the IBM Common Services host which can be displayed by the following command:

   ```
   oc get route -n <cs_namespace> cp-console -o jsonpath='{.spec.host}'
   ```

   Where `<cs_namespace>` is the namespace where IBM Common Services is installed.

   This output will be referred to as `<cs_host>` in subsequent commands.

   Add the helm repo with the following command:

   ```
   helm repo add <chart_repo_name> https://<cs_host>:443/helm-repo/charts --ca-file ~/.helm/
   ca.pem
   ```

   Where `<chart_repo_name>` is the repository chosen to load the IBM Agile Lifecycle Manager archive.

7. Log in to the Red Hat OpenShift Container Platform docker registry:

Obtain a token using the command:

```
oc whoami --show-token=true
```

Use this value to login to docker:

```
docker login -u <cluster_admin> -p <token> <ocp_registry>
```

8. Login in to cloudctl:

   Obtain the administrator password using the command

   ```
   oc -n <cs_namespace> get secret platform-auth-idp-credentials -o
   jsonpath='{.data.admin_password}' | base64 -d
   ```

   Where `<cs_namespace>` is the namespace where IBM Common Services is installed.

   Login to cloudctl using the password obtained in the previous command.

   ```
   cloudctl login -a <cs_host> -u admin -p <password> -n default
   ```

   Where `<cs_host>` is the route to the IBM Common Services host obtained in step 6.
9. Load the IBM Agile Lifecycle Manager archive:

   ```
   cloudctl catalog load-archive --archive <alm_archive_file> --registry <ocp_registry>/
   <alm_namespace> --repo local-charts
   ```

   Where `<alm_archive_file>` is the archive file containing the IBM Agile Lifecycle Manager package.

   The IBM Agile Lifecycle Manager Helm charts will be uploaded to the Helm Red Hat OpenShift Container Platform local-charts repository, and the Docker images to the Red Hat OpenShift Container Platform Docker registry. You can view the chart repositories on Red Hat OpenShift Container Platform:

   ```
   cloudctl catalog repos
   ```
10. Update the local Helm repositories information:

    ```
    helm repo update
    ```

    Verify that the Helm charts are now visible from the Helm CLI:

    ```
    helm search alm
    ```

    **Run the prerequisite script**

    If static provisioning for local storage is to be used for StatefulSet component storage execute steps 11 to 13. If Dynamic Provisioning is to be used, steps 11 to 13 can be skipped.
11. Extract the scripts from the archive.

    Use the following command:

    ```
    tar -xvf <alm_archive_file> pak_extensions
    ```
12. Add configuration data to the `storageConfig.env` file.

    A minimum of three workers must be configured. The recommended values are shown in the following example.

    ```
    cd pak_extensions/prereqs/
    vi storageConfig.env
    ```

    For example:

```
WORKER1=worker1.an.os.example.abc.com
WORKER2=worker2.an.os.example.abc.com
WORKER3=worker3.an.os.example.abc.com
FS_ROOT=/opt/ibm/alm
CAPACITY_CASSANDRA=130
CAPACITY_KAFKA=295
CAPACITY_ELASTICSEARCH=50
CAPACITY_ZOOKEEPER=25
```

**Note:** For the `WORKER1` to `WORKER3` values, you must input the full worker name as found from the following command:

```
oc get nodes
```

**Note:** IBM Agile Lifecycle Manager PersistentVolumeClaims only support PersistentVolumes with an access mode of `ReadWriteOnce`. For further information on access modes in relation to PersistentVolumes, see the following topic on the Kubernetes website: Access Modes

13. Execute the following script:

```
./createStorageVolumes.sh <alm_namespace> <alm_release>
```

Where `<alm_release>` is the name you chose for the release.

Create the required folders described in the output of the createStorageVolumes script.

**Install IBM Agile Lifecycle Manager**

You can install IBM Agile Lifecycle Manager either from the Helm CLI, or the Red Hat OpenShift Container Platform console. It is possible to deploy using local storage or Dynamic Provisioning. Both install commands are detailed below.

14. To install IBM Agile Lifecycle Manager from the Helm CLI using **local storage**, use the following command:

```
helm install --name <alm_release> <chart_repo_name>/ibm-alm-prod --namespace
<alm_namespace> --set license=accept,global.image.repository=image-registry.openshift-image-
registry.svc:5000/<alm_namespace> --tls --tiller-namespace <cs_namespace>
```

Where the parameter `<chart_repo_name>` is the repository chosen to load the archive in Step 6.

To install IBM Agile Lifecycle Manager from the Helm CLI using **Dynamic Provisioning**, use the following commands:

Verify the availability of the required storage class and note the value in the `NAME` column of the selected storage class in the command output:

```
oc get storageclass
```

Execute the helm install command setting custom values related to `global.persistence.useDynamicProvisioning` and `global.persistence.storageClassName`:

```
helm install --name <alm_release> <chart-repo-name>/ibm-alm-prod --namespace
<alm_namespace> --set license=accept,global.image.repository=image-registry.openshift-image-
registry.svc:5000/<alm_namespace> --tls --set
global.persistence.useDynamicProvisioning=true --set
global.persistence.storageClassName=<storage_class_name> --tiller-namespace <cs_namespace>
```

Where `<storage_class_name>` is the name of the chosen storage class.

## Verifying a successful installation of IBM Agile Lifecycle Manager

This topic describes how to verify a successful installation by checking the status of the IBM Agile Lifecycle Manager services and displaying the IBM Agile Lifecycle Manager user interface login page.

### Before you begin

Before you perform this task, see "Installing IBM Agile Lifecycle Manager on Red Hat OpenShift Container Platform" on page 19.

### Procedure

To verify a deployment of IBM Agile Lifecycle Manager, use the following steps:

1. Execute the following helm test command:

   ```
   helm test <alm_release> --cleanup --tls --tiller-namespace <cs_namespace>
   ```

   This command checks the status of the IBM Agile Lifecycle Manager services.

   `PASSED` will be returned for all tests if the install was successful.
2. Execute the following helm status command for the installation.

   ```
   helm status <alm_release> --tls --tiller-namespace <cs_namespace>
   ```

   This command returns the following details about routes, pod status and deployment status.

   **`v1/Route` section**

   Routes related to `ishtar` and `nimrod` should be present.

   **`v1/Pod(related)` section**

   All pods listed should display `STATUS Running`, with the exception of `brent-onboarding` which should have `STATUS Completed`.

   **`v1/Deployment` section**

   The value of the `DESIRED`, `CURRENT`, `UP-TO-DATE` and `AVAILABLE` number of pods listed for each deployment should be equal. The number of pods `AVAILABLE` should match the `DESIRED` value for each deployment.

   **`v1/StatefulSet` section**

   For the stateful applications listed, the `CURRENT` number of pods should match the `DESIRED` value.

### What to do next

See "Integrating IBM Agile Lifecycle Manager with an existing external directory service" on page 23.

## Integrating IBM Agile Lifecycle Manager with an existing external directory service

IBM Agile Lifecycle Manager requires an LDAP directory service to authenticate and authorize users. IBM Agile Lifecycle Manager can be configured to use groups and users from a directory service and associate those to roles and privileges.

### IBM Agile Lifecycle Manager configuration for LDAP Directory Service

The configuration used by IBM Agile Lifecycle Manager to connect to an external Directory Service is specified in a generic Kubernetes secret in the form of literal key-value pairs.

*Table 4. Supported configuration parameters*

| Parameter | Description | Required or Optional |
|---|---|---|
| `LDAP_URL` | Specifies the LDAP connection URL to the external Directory Service. | Required. |
| `LDAP_BIND_DN` | Specifies the user DN that is used to authenticate to the LDAP Directory Server. If not specified, an anonymous bind is attempted. | Optional. |
| `LDAP_ADMIN_PASSWORD` | Specifies the bind user's password. This is required if anonymous bind not available.<br>No default values are set. | Optional. |
| `LDAP_BASE_DN` | Specifies the DN of the search base.<br>The default is `'dc=lm,dc=com'`. | Optional. |
| `LDAP_USER_SEARCH_BASE` | Specifies the search base for the user. If the parameter `LDAP_BASE_DN` is specified, the user search base must be relative to it.<br>The default is `'ou=people'`. | Optional. |
| `LDAP_USER_SEARCH_FILTER` | Specifies the filter used in the user search.<br>The default is `'uid={0}'`. | Optional. |
| `LDAP_GROUP_SEARCH_BASE` | Specifies the search base for groups of which the user is a member. If the parameter `LDAP_BASE_DN` is specified, the group search base must be relative to it.<br>The default is `'ou=groups'`. | Optional. |
| `LDAP_GROUP_SEARCH_FILTER` | Specifies the filter used in the group search<br>The default is `'member={0}'`. | Optional. |
| `LDAP_PASSWORD_ATTR` | Specifies the name of the attribute in the Directory Service that contain the user password. | Optional. |

*Table 4. Supported configuration parameters (continued)*

| Parameter | Description | Required or Optional |
|---|---|---|
| LDAP_PASSWORD_ENC | Specifies the encryption algorythm used to encrypt the user password. Possible values are BCRYPT and PLAIN. If not specified, the default value BCRYPT is used. | Optional. |
| AUTH_PROVIDER | Specifies the type of bind to be performed when the IBM Agile Lifecycle Manager client connect to the LDAP Directory Server. Possible values are ldapSimple and ldapBind. In the ldapSimple type, the bind user is used to perform LDAP activities. In the ldapBind type, the bind user is used to search the IBM Agile Lifecycle Manager user and to retrieve its password. A subsequent bind is performed using the user credentials. Any subsequent LDAP operation is carried out as the user. If not specified, the default value ldapSimple is used. | Optional. |

**Note:** The properties LDAP_SEARCH_BASE and LDAP_SEARCH_FILTER available in previous versions of IBM Agile Lifecycle Manager have been renamed to LDAP_USER_SEARCH_BASE and LDAP_USER_SEARCH_FILTER. If you are using a secret definition from a previous version, make sure to update the name of these properties.

### Configuring IBM Agile Lifecycle Manager to use an LDAP directory service

To configure IBM Agile Lifecycle Manager to use an LDAP directory service, you need to create a kubernetes secret containing the LDAP parameters.

The following code shows an example of a command to create an LDAP secret.

```
oc create secret generic <alm_ldap_config_secret> \
--from-literal=LDAP_BASE_DN=<base-dn> \
--from-literal=LDAP_BIND_DN=<bind-dn> \
--from-literal=LDAP_ADMIN_PASSWORD=<bind-dn-password> \
--from-literal=LDAP_PASSWORD_ATTR=<user-password-attribute> \
--from-literal=LDAP_URL=ldap://<ldap-server-host>:<ldap-server-port> \
--from-literal=LDAP_USER_SEARCH_BASE=<user-search-base> \
--from-literal=LDAP_USER_SEARCH_FILTER=<user-search-filter> \
-n <alm_namespace>
```

Then you need to update IBM Agile Lifecycle Manager to use the LDAP configuration by running the following command:

```
helm upgrade <alm_release> --set global.security.ldapConfig.secretName=<alm_ldap_config_secret>
--install <chart_repo_name>/ibm-alm-prod --reuse-values --tls --tiller-namespace <cs_namespace>
```

Where the parameter `<alm_release>` is the name chosen for the release and `<chart_repo_name>` is the repository chosen to load the archive during the IBM Agile Lifecycle Manager installation procedure.

### IBM Agile Lifecycle Manager roles and privileges

Roles in IBM Agile Lifecycle Manager are defined by the privileges to which they are associated. IBM Agile Lifecycle Manager uses privileges to authorize users. IBM Agile Lifecycle Manager comes with a set a predefined roles, `SLMAdmin`, `Portal`, `ReadOnly` and `RootSecAdmin`. These predefined roles are associated by default to LDAP groups with the predefined group names: `SLMAdmin`, `Portal`, `ReadOnly` and `RootSecAdmin`.

To use predefined roles, you can create groups in the LDAP directory service named after the predefined roles and add members to them. Alternatively, you can associate custom LDAP directory service groups to predefined roles. For details see Associate custom LDAP Directory Service groups to pre-defined roles.

### Creating new roles in IBM Agile Lifecycle Manager

New roles can be defined in a kubernetes `ConfigMap` YAML file where the data section related to the `alm-role` is in JSON format.

The following code shows an example of a `ConfigMap` which defines a role called `alm-operator`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: alm-operator
  namespace: alm-namespace
data:
  alm-operator.json: |-
    {
       "alm" : {
          "roles" : {
           "alm-operator" : {
              "privileges" : {
                 "NsinstsMgt" : "read,execute",
                 "nsDesMgt" : "read",
                 "DeployLocMgt" : "read",
                 "VnfInstsMgt" : "read",
                 "BehvrScenDes" : "read",
                 "VduInstsMgt" : "read",
                 "IntentReqslMgt" : "read,execute",
                 "VduGrpMgt" : "read",
                 "BehvrScenExec" : "read,execute",
                 "VduDesMgt" : "read",
                 "RmDrvr" : "read"
              },
              "ldapGroups" : [
                 "Operators"
              ]
           }
          }
       }
    }
```

The `alm-operator` role is associated to an LDAP group called `Operators`. Users who are members of the LDAP group `Operators` will inherit the privileges associated to the `alm-operator` role.

Create the ConfigMap by running the following command:

```
oc create -f <ConfigMap-yaml-file>
```

After creating the `ConfigMap`, update IBM Agile Lifecycle Manager passing the `ConfigMap` in the `global.security.almRoles.configMapName` property to allow IBM Agile Lifecycle Manager to use the new role:

```
helm upgrade <alm_release> --set global.security.almRoles.configMapName=alm-operator --install
<chart_repo_name>/ibm-alm-prod --reuse-values --tls --tiller-namespace <cs_namespace>
```

Where the parameter `<alm_release>` is the name chosen for the release and `<chart_repo_name>` is the repository chosen to load the archive during the IBM Agile Lifecycle Manager installation procedure.

**Associate custom LDAP Directory Service groups to pre-defined roles**

You may want to use existing LDAP directory service groups with predefined IBM Agile Lifecycle Manager roles. To do that, you need to create a `ConfigMap` that associates your groups to the predefined roles. For example, the following `ConfigMap` associates a group called `my_custom_group` to the predefined role `Portal`.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: alm-portal-group
  namespace: default
data:
  alm-portal-group.json: |-
    {
       "alm" : {
          "roles" : {
           "Portal" : {
              "ldapGroups" : [
                 "my_custom_group"
              ]
           }
          }
       }
    }
```

After creating the `ConfigMap`, update IBM Agile Lifecycle Manager passing the `ConfigMap` in the `global.security.almRoles.configMapName`:

```
helm upgrade <alm_release> --set global.security.almRoles.configMapName=alm-portal-group --
install <chart_repo_name>/ibm-alm-prod --reuse-values --tls --tiller-namespace <cs_namespace>
```

Where the parameter `<alm_release>` is the name chosen for the release and `<chart_repo_name>` is the repository chosen to load the archive during the IBM Agile Lifecycle Manager installation procedure.

**Note:** Only one `ConfigMap` can be passed to IBM Agile Lifecycle Manager, so all the configuration to create new roles and associate groups to roles have to be specified in one `ConfigMap`. It is not possible to modify predefined roles by adding or removing privileges.

For details of five examples that explain how to set up an LDAP directory service and plug in the same with the IBM Agile Lifecycle Manager deployment, see "LDAP directory service examples" on page 89.

## Accessing the IBM Agile Lifecycle Manager UI

The IBM Agile Lifecycle Manager User interface is available at http://`<nimrod_route_hostname>/ui/login`

Where `<nimrod_route_hostname>` is the host on which IBM Agile Lifecycle Manager is installed.

**Note:** You can obtain the name of `<nimrod_route_hostname>` by using the following command:

```
oc get routes alm-nimrod -o jsonpath='{.spec.host}' --namespace=<alm_namespace>
```

The login page is available, but you will not be able to log in until you have configured an LDAP server.

## Invoking REST API

This section describes how to invoke REST API.

To access the REST API, you need to retrieve a token. This is done as follows:

1. Fetch the `<client_secret>` which is the key for 'clientId: Admin' using the base64 decode

   ```
   oc get secrets <alm_credentials_secret> -o jsonpath='{.data.adminClientSecret}' | base64 -d
   ```

   Where:

   `<alm_credentials_secret>` is the secret holding almCredentials

`<alm_namespace>` is the namespace in which IBM Agile Lifecycle Manager is deployed.

2. Encode the header for basic authorisation

```
echo -n "Admin:<client_secret>" | base64
```

3. Fetch the `<ishtar_route>`

```
oc get routes alm-ishtar -o jsonpath='{.spec.host}' -n <alm_namespace>
```

4. Generate the <access_token> required for Oauth authentication

```
curl -sk -X POST "https://<ishtar_route>/oauth/token" -H "Authorization: Basic
<basic_auth_encoded>" -d 'grant_type=client_credentials' | python -c 'import sys, json;
print json.load(sys.stdin)["access_token"]'
```

Where:

`<basic_auth_encoded>` is the header generated in Step 2.

`<ishtar_route>` is the route obtained in Step 3.

5. To test access to the API, run a curl command to retrieve assemblies

```
curl --insecure -H 'Accept: */*' -X GET -H "Authorization: Bearer <access_token>" https://
<ishtar_route>:443/api/topology/assemblies
where :
```

If you have created any assemblies, they should be displayed, or else an empty set will be returned. You should not receive an error.

All available REST commands are discussed in detail in Chapter 5, “Getting started using the APIs,” on page 63.

## Using custom keys and certificates for deploying IBM Agile Lifecycle Manager

The resources of IBM Agile Lifecycle Manager are protected by an authentication layer in place on the gateway i.e Ishtar. In order to gain access to any of these protected resources, an authorisation process takes place. As a minimum, this requires providing some valid client credentials (client Id and secret) which are normally deemed as sufficient security.

- The Nimrod(User Interface) and Ishtar(API Gateway) components have a security layer enabled which protects their APIs.
- Ishtar acts as an Oauth2 provider and can provide tokens (JWT format) which enable authorisation against the protected APIs. To authenticate, it is necessary to provide a secret in a token request to Ishtar for the Ishtar clientId (Admin).
- JWT access tokens given out by Ishtar contain a list of roles that the user has. The token is signed using the jwtSigningKey to avoid tampering, which is a private key.
- Nimrod requires its own client (NimrodClient) in order to communicate with backend APIs through Ishtar. When the user logs in through the UI, the username and password(from LDAP) are combined with Nimrod's client credentials to make a token request against Ishtar. A token is then supplied, which the UI can use to make backend calls until its expiry.
- Doki also requires its own client (DokiClient). This is so that it can provide access control against the backend calls which are made while executing behaviour scenarios.
- Generally the Admin clientId is used and it enables access to APIs in a protected way.
- "client-credentials-bootstrap.yml" is a yaml file holding details of each of the above clientCredentials is made available to Ishtar through a Kubernetes secret and passed at install time.
- All other services communicate directly (i.e. not via Ishtar) and so can avoid the need to authenticate.
- Conductor, Ishtar, Brent and Vault have their own self-signed certificates generated.

These specific keys and certificates are provided in the form of kubernetes secrets and made available to the system. These secrets are automatically generated during the deployment.

Alternatively, you can pass your own keys and certificates to the IBM Agile Lifecycle Manager deployment in the form of self-created kubernetes secrets at the time of install.

A sample script which implements and creates keys, certificates and custom secrets to be passed to the deployment is included with the package and is available at pak_extensions/prereqs/createSecrets.sh. This example script uses openssl and keytool to generate the certificates. Customers are free to use any other tool also.

### Using custom keys and certificates

To use custom keys and certificates, use the following the steps:

1. IBM Agile Lifecycle Manager expects below secrets to be pre-created and passed at the time of deployment:
   - `global.security.almCredentials.secretName`: Name of a pre-created secret containing the IBM Agile Lifecycle Manager credentials.
   - `global.security.almCerts.secretName`: Name of a pre-created secret containing the IBM Agile Lifecycle Manager certificates.
   - `global.security.vaultCerts.secretName`: Name of a pre-created secret containing the Vault certificates.

   The above three secrets have to be created in the namespace in which IBM Agile Lifecycle Manager is deployed. The name of each of these secrets is customizable, which means you can choose whatever name you want.

   a. **Create secret to hold almCredentials (for example, <custom_credentials_secret>).**

      Each of the keys `<nimrod_client_key>`, `<admin_client_key>`, `<doki_client_key>`, `<jwt_signing_key>` and certificates for Conductor, Ishtar and Brent are to be created using any keys and certificates generator tool (for example openssl or keytool)

      Create `<custom_credentials_secret>` by using existing set of keys for nimrod, doki and admin clients.

      Create the `client-credentials-bootstrap.yml file` as shown, by setting clientSecret values to `<nimrod_client_key>`, `<admin_client_key>`, `<doki_client_key>` for the respective clientIds, in addition to values for attributes such as grantTypes, roles, accessTokenValidity and refreshTokenValidity.

```
clientCredentials:
  - clientId: Admin
    clientSecret: <admin_client_key>
    grantTypes: client_credentials
    roles: SLMAdmin
  - clientId: DokiClient
    clientSecret: <doki_client_key>
    accessTokenValidity: "1200" #20 minutes
    grantTypes: client_credentials
    roles: BehaviourScenarioExecute
  - clientId: NimrodClient
    clientSecret: <nimrod_client_key>
    accessTokenValidity: "1200"    #20 minutes
    refreshTokenValidity: "30600" #8.5 hours
    grantTypes: password,refresh_token,client_credentials
```

      Pass the above created file while creating the <custom_credentials_secret>, as shown below :

```
oc create secret generic <custom_credentials_secret> \
--from-literal=adminClientSecret=<admin_client_key> \
--from-literal=dokiClientSecret=<doki_client_key> \
--from-literal=nimrodClientSecret=<nimrod_client_key> \
--from-file client-credentials-bootstrap.yml \
--from-literal=jwtSigningKey=<jwt_signing_key> \
-n <alm_namespace>
```

Where:

`<custom_credentials_secret>` is the name of the secret to hold IBM Agile Lifecycle Manager credentials,

`<admin_client_key>` is the key for clientId: Admin,

`<doki_client_key>` is the key for clientId: DokiClient,

`<nimrod_client_key>` is the key for clientId: NimrodClient,

`<jwt_signing_key>` is the key for securing the JWT token generated by Ishtar,

`<alm_namespace>` is the namespace in which IBM Agile Lifecycle Manager is deployed.

b. **Create secret to hold almCerts (for example <custom_security_secret> )**.

Pass the certificate files for Conductor, Ishtar and Brent and create <custom_security_secret>.

```
oc create secret generic <custom_security_secret> \
--from-file <conductor_certificate> \
--from-file <ishtar_certificate> \
--from-file <brent_certificate> \
--from-file <keystore_name> \
--from-literal=keystorePassword=<keystore_password> \
-n <alm_namespace>
```

Where:

`<custom_security_secret>` is the name of the secret to hold IBM Agile Lifecycle Manager certificates,

`<conductor_certificate>` is the certificate for conductor component,

`<ishtar_certificate>` is the certificate for ishtar component,

`<brent_certificate>` is the certificate for brent component,

`<keystore_name>` is the file named .keystore that is set by the user based on the type of storetype,

`<keystore_password>` is the passphrase used for securing the .keystore file,

`<alm_namespace>` is the namespace in which IBM Agile Lifecycle Manager is deployed.

c. **Create secret to hold vaultCerts (for example <custom_vault_secret> )**

Pass the Vault key and certificate files to create <custom_vault_secret>.

```
oc create secret generic <custom_vault_secret> \
--from-file <vault_key> \
--from-file <vault_certificate> \
-n <alm_namespace>
```

Where:

`<custom_vault_secret>` is the name of the secret to hold Vault certificates,

`<vault_key>` is the key for vault component,

`<vault_certificate>` is the certificate for vault component,

`<alm_namespace>` is the namespace in which IBM Agile Lifecycle Manager is deployed.

2. Trigger a deployment by passing the created secrets.

   Use with `helm install`.

```
helm install --name <alm_release> <chart_repo_name>/ibm-alm-prod --namespace <alm_namespace>
--set license=accept,global.image.repository=image-registry.openshift-imageregistry.svc:5000/
<alm_namespace> --set global.security.almCerts.secretName=<custom_security_secret> --set
global.security.almCredentials.secretName=<custom_credentials_secret> --set
global.security.vaultCerts.secretName=<custom_vault_secret> --tiller-namespace
<cs_namespace> --tls
```

The parameters `<chart_repo_name>` is the repository chosen to load the archive during the IBM Agile Lifecycle Manager installation procedure, `<alm_namespace>` is the namespace in which IBM Agile Lifecycle Manager is deployed and `<cs_namespace>` is the namespace in which IBM Common Services is installed.

## Upgrading IBM Agile Lifecycle Manager

You cannot upgrade an existing deployment of IBM Agile Lifecycle Manager on Red Hat OpenShift Container Platform.

Instead, you must uninstall it as described in "Uninstalling IBM Agile Lifecycle Manager" on page 31, and then perform a fresh installation.

## Uninstalling IBM Agile Lifecycle Manager

Uninstall IBM Agile Lifecycle Manager from Red Hat OpenShift Container Platform by performing the following steps.

### About this task

This procedure uninstalls the deployed version of IBM Agile Lifecycle Manager, but does not remove the load images or charts.

**Note:** When you uninstall a release, any unused docker images that were used as part of the installation remain on the master node and on a local repository on one or more worker nodes. From time to time, Kubernetes removes unused images as part of its garbage collection process. For more information, see Configuring kubelet Garbage Collection.

### Procedure

1. Delete the release to be removed by running the following Helm command:

   ```
   helm delete --purge --tls <alm_release>  --tiller-namespace <cs_namespace>
   ```

   Where `<alm_release>` is the name the release to be uninstalled.
2. Remove Persistent Volumes and Persistent Volume Claims.

   If you created the Persistent Volumes and Persistent Volume Claims using the `createStorageVolumes.sh` then run the `deleteStorageVolumes.sh` script:

   ```
   ./deleteStorageVolumes.sh <alm_release>
   ```

   Then manually delete the folders described in the output of the script.

   If using dynamically provisioned storage, delete the Persistent Volume Claims:

   ```
   oc delete pvc -l release=<alm_release> -n <alm_namespace>
   ```

   If the reclaim policy of the used storage class is set to Delete, the Persistent Volumes are automatically deleted.

   You can verify that the Persistent Volumes have been automatically deleted by using the command:

   ```
   oc get pv | grep "<alm_namespace>/data-alm-"
   ```

   If the Persistent Volumes are not automatically deleted, you may want to ask your storage provisioner to clean them. Notice that dynamically provisioned Persistent Volumes left from previous deployments of IBM Agile Lifecycle Manager won't prevent new deployments to succeed, as new Perstistent Volumes will be created and associated to the Persistent Volume Claims.
3. Clean up remaining cron job objects and their related pods.

After IBM Agile Lifecycle Manager has been uninstalled, orphaned job objects and related pods may remain on the system. Remove these as in the following example:

```
oc delete job -l release=<alm_release> --namespace
<alm_namespace> --cascade
```

And execute the following command:

```
oc delete secret alm-vault-keys
```

# Chapter 4. Using IBM Agile Lifecycle Manager

This section describes the IBM Agile Lifecycle Manager User Interface, and the tasks it allows you to perform.

## Deploying IBM Agile Lifecycle Manager

To develop, deploy and operate IBM Agile Lifecycle Manager within an operational environment requires a wide range of expertise, including network domain knowledge, expertise in third-party technology, and experience of IBM Agile Lifecycle Manager.

The process of deploying IBM Agile Lifecycle Manager can be understood as focusing first on the network lifecycle to be modeled, then translating that understanding into a model deployed into IBM Agile Lifecycle Manager, before using IBM Agile Lifecycle Manager to manage the network.

**Developing virtual network functions (VNF)**

In a typical use case, a subject matter expert with both domain knowledge and expertise in third party VNF software is responsible for the creation or onboarding of third party VNF software into an operational package suitable for management by IBM Agile Lifecycle Manager.

This also includes creating VNF functional tests and behavior scenarios, with a typical end-to-end workflow as follows:

1. Create a new VNF project.
2. Develop the VNF components for the project.
3. Onboard the VNF components to IBM Agile Lifecycle Manager and a Resource Manager (RM) being managed by IBM Agile Lifecycle Manager.
4. Design and test the VNFs by assembling the required components and connecting them together as required.

**Creating network service designs**

Following the development and testing of VNFs, these are translated into a Network Service design, which is then implemented by connecting Network Service and VNF packages to a location-based network configuration.

Service packages and their third-party resources are tested and their expected behavior verified before the packages is deployed into production.

Such a deployment scenario is typically be undertaken by someone with advanced IBM Agile Lifecycle Manager expertise, and follows a workflow much like the following:

1. Create a new Service Design in IBM Agile Lifecycle Manager.
2. Define the behavior-driven test scenarios to validate the Service Design.
3. Perform service testing in IBM Agile Lifecycle Manager using the behavior-driven test scenarios.
4. Export the Service Design into an operational environment.

**Managing the lifecycle of services within production environments**

Once deployed, someone with operational IBM Agile Lifecycle Manager expertise manages the life of service instances within production environments.

Such an operational expert executes new or planned changes to service instances, and is responsible for diagnosing and resolving reported errors or anomalies.

Operational tasks encompass the following:

1. Deploy a new service into production.
2. Monitor and maintain the service.
3. Decommission the service.

## IBM Agile Lifecycle Manager tools and user interfaces

IBM Agile Lifecycle Manager itself can be accessed and used through its native Graphical User Interface or commanded through APIs.

Besides core IBM Agile Lifecycle Manager, there are also a set of supporting tools around it.

- Lifecycle Manager Controller is a command line tool utilizing IBM Agile Lifecycle Manager's APIs and helping to make onboarding VNF components and VNFs easier and supporting in exporting VNF and Network Service packages from IBM Agile Lifecycle Manager.
- CI/CD hub to give a framework for managing various artifacts though the whole DevOps process from development phase to production deployments.

  **Note:** This is an example framework and not delivered as a part of the product. CD/CD Hub is documented here as an example of how to manager the DevOps process.

**Lifecycle Manager Controller (lmctl)**

Lifecycle Manager Controller (lmctl) command line tools provide a set of tools to automate the management of lifecycle manager environments and the tasks required to deploy VNFs and Network Services.

The lmctl tool supports the following use cases:

- Push and pull VNF/Network Service projects to/from lifecycle manager environments
- Run behavior tests
- Perform administrative tasks on lifecycle manager environments

**CI/CD Hub**

The CI/CD Hub (HUB) is a collection of open source tools that can be set up to perform example continuous integration and deployment processes with the IBM Agile Lifecycle Manager. The HUB provides a way to manage the artifacts used by IBM Agile Lifecycle Manager through a development and production cycle.

These include:

- Gogs git server – a repository to manage the files used by IBM Agile Lifecycle Manager.
- Nexus repository – a repository for managing the built artifacts produced by the process. This includes a docker registry, a python registry, and a raw repository for storing archives, including a package of IBM Agile Lifecycle Manager All-in-one single server instance of IBM Agile Lifecycle Manager.
- Jenkins server – a platform to perform automated tasks such as running tests, packaging built objects.
- IBM Agile Lifecycle Manager – an instance of the IBM Agile Lifecycle Manager to be used by the Jenkins server

Once installed the hub comes with some artifacts already installed. These include a version of IBM Agile Lifecycle Manager packaged to work with the HUB and a set of VNF and a Network Service to show how the HUB can be used.

**Managing the lifecycle of services within production environments**

Once deployed, someone with operational IBM Agile Lifecycle Manager expertise manages the life of service instances within production environments.

Such an operational expert executes new or planned changes to service instances, and is responsible for diagnosing and resolving reported errors or anomalies.

Operational tasks encompass the following:

1. Deploy a new service into production.
2. Monitor and maintain the service.
3. Decommission the service.

## IBM Agile Lifecycle Manager GUI functionality

You use the GUI to create, operate and delete service instances managed by IBM Agile Lifecycle Manager. You also use it to design new services and their components, define behavior test scenarios for various testing, verification and diagnostics purposes, as well as perform a number of administration tasks.

### Logging on

**Restriction:** The IBM Agile Lifecycle Manager GUI is only supported in the latest version of Google Chrome.

After successfully logging into IBM Agile Lifecycle Manager, the default **Recent Assembly Instances** page is opened showing the recently created or modified service instances.

The main elements on all pages within IBM Agile Lifecycle Manager user interface has two main parts:

- Navigation Panel on the left side of the page.
- The main application view to the right of the Navigation Panel. The contents of this view will vary depending on the active application page.

The left-hand Navigation Panel provides access to all main applications within IBM Agile Lifecycle Manager including the following Main applications. Availability of these applications to individual users is controlled by the roles the user is associated with. Different IBM Agile Lifecycle Manager user roles allow access to different set of applications so all the above options may not be available to all users.

**Operations**

Operations applications provide capabilities to create, operate, and delete service instances managed by IBM Agile Lifecycle Manager. Operations capabilities can be accessed through two different pages depending of the preferred method to find service instances. Creation of a new service instance can be performed equally through both pages.

- Recent Assembly Instances: Recent Assembly Instances lists 20 assemblies based on last executed intents.
- Assembly Instance Search: Assembly Instance Search allows the user to search specific service instances by name.

**Designer**

Designer is a toolset for service developers to design new services and their components as well as define behaviour test scenarios for various testing, verification and diagnostics purposes.

- Assembly Designer: Assembly Designer provides tools for developers to define new service descriptors either to define new VNFs or new network services or change the existing descriptors. Designer has also integrated Behaviour Testing tool to define and run testing scenarios on instantiated services

**System**

System section contains tools for systems administration to make IBM Agile Lifecycle Manager aware of underlying available deployment locations where service components can be instantiated and resource managers managing specific VNFCs on specific infrastructure technologies.

- A resource manager called Brent is installed automatically when installing IBM Agile Lifecycle Manager. This resource manager will appear in the system section and can be seen running as a pod on the Red Hat OpenShift Container Platform cluster.
- Deployment Locations: Deployment Locations page allows users to define new deployment locations to instantiate resources.

## Operations

Operations applications provide capabilities to create, operate, and delete service instances managed by IBM Agile Lifecycle Manager. Operations capabilities can be accessed through two different pages depending of the preferred method to find and view existing service instances. Creation of a new service instance can be performed equally through both pages.

### Recent Assembly Instances

The Recent Assembly Instances page, functioning also as the default page after login, provides direct access to the latest assemblies, i.e. service instances, and their status. The Last 20 assemblies having had intents executed on them are shown in the list.

Each Assembly Instance is listed on a new line within the list, showing:

- Name
- Current State
- Assembly Descriptor it is built upon
- Last intent that was executed
- Status of the last intent executed
- Date & time of the last intent execution

For each listed service instance there are options available to trigger a new lifecycle intent (Make Inactive, Upgrade or Uninstall) by clicking the “New Intent” button. Triggering any intent will activate the execution view, described later in this section, to see the timeline of steps executed to complete the intent.

Navigation to the topology of the service instance is available by clicking the “Open” Button.

Additionally, there are navigation items available through the “more items” menu represented as an ellipsis icon (“...”) on the far right-hand end of each Assembly Instance row. Available options include:

- “Open Execution” to see the details of the last intent and its execution steps.
- “Open Design” to open designer and view or modify the service design the service instance is based on.
- “Execution History” to see the lifecycle history of the service instance.

Creation of a completely new service instance can be done by clicking the “Create” button in the top right corner of the page.

### Topology

The Topology view of the service (displayed by using the “Open” button on the Recent Assemblies list) shows the service structure and all the elements instantiated as part of the service.

The view is made up of the following:

- Top header bar – This provides basic information about the topology being viewed and an indicator as to whether any new intents have been executed.
- Right-hand properties and relationships panel – This has individual sections that can be expanded/ collapsed and a button that allows a new intent to be executed.
- Bottom tab bar – This allows the user to easily switch between the Topology view and the Process Execution View

The Top-level service is indicated as “Box” symbol on a dark background on top of the main view. Service elements have their own symbols depending on their type: an individual resource or sub assembly is indicated as a single hexagon-symbol. A cluster is shown as a stacked hexagon, and finally an external reference with a hexagon with dotted outline.

Details of a selected element can be made visible in the right-hand properties and relationships panel by clicking the symbol of the element.

A new lifecycle intent (Make Inactive, Upgrade or Uninstall) can be triggered by clicking the “Change Intent” button.

Toggling between the “Topology” and “Execution” pages can be done by clicking the tabs on the bottom of the page.

### Execution

The Execution page (also known as the Process Execution View) shows the details of the last intent executed on the selected service instance. On the left side of the execution page the service topology is shown as a hierarchical list of assemblies and resources.

The main application view shows a Gantt chart of individual steps executed within the intent and their dependencies. The status of each step is indicated with colour codes (Green: successfully completed, Red: failed, Blue: currently executed, pale Blue: planned/waiting and Grey: Cancelled).

The progress bars within the dynamic Gantt view also use different shapes to indicate the different types of activities being displayed:

- Square ended/rectangle shaped bars - Lifecyle Transition
- Round ended/lozenge shaped bars - Operations
- Pointed ended/Diamond shaped bars - Relationships

The dependency lines drawn between the various progress bars are also colour coded:

- Black solid lines – execution dependency. The “destination” of the dependency (i.e. what event was triggered) is shown by a solid black circle at the end of the line.
- Orange dashed lines – relationship dependency. The “destination” of the dependency (i.e. what event was triggered by the relationship) is shown by a solid orange circle at the end of the line.

The Top bar shows the key details of the last intent and provides options to trigger a new intent on the service instance by clicking the “Change Intent” button or view the intent history by clicking the “Execution History” button.

Toggling between “Topology” and “Execution” views can be done by clicking the tabs on the bottom of the view.

The history of intents executed on the service instance is available by clicking the “Execution History” button on the top bar of the “Execution” view

### Execution history

The Intent history can be opened in a dialog box by clicking the “Execution History” button on the top bar of the Execution page. In the dialog box a list of executed intents is shown in chronological order with the oldest intent at the top of the list. By clicking the “Inspect” button next to an intent opens the “Execution” view to view details of the selected execution.

### Assembly instance search

Functionally the Assembly Instance Search view is the same as the Recent Assembly Instances view (see “Recent Assembly Instances” on page 36).

The difference is that this view is only populated once an Assembly Search term has been added into the search bar and the <Enter> key pressed. When first displayed there are no entries in the matching Assemblies list.

To search for an Assembly simply enter its name (or partial name) into the search bar and press the <Enter> key. Only searching by Assembly name is currently supported.

Currently it is not possible to change the order of the displayed results. They are shown in ascending alphabetical order of matching Assembly name.

## Service Designer

The Service Designer is a toolset for service developers to design new services and their components as well as define behaviour test scenarios for various testing, verification and diagnostics purposes.

### Assembly Descriptors

When opening the designer, the first view is the “Assembly Descriptors” listing all available descriptors that can be selected to be edited in the designer. The descriptors can define a top-level network service, a service segment or a VNF (Virtual Network Function).

At the top of the page is a search bar that allows Assembly Descriptors to be filtered by name. Partial name filtering is supported. The results panel below the search bar will automatically update as you enter the characters of the name. There is no need to click on any filter icon.

Beneath the search/filter bar is the list of matching Assembly Descriptors. Each Assembly Descriptor is shown on a separate line with the following information:

- Name
- Version
- Description
- “Open” button
- Ellipsis (“...”) to access further Assembly Descriptor actions (e.g. Delete)

Currently it is not possible to sort the contents of the Assembly Descriptor list.

There is also a possibility to create a completely new service design by clicking the “Create” button in the top right corner of the page.

### Designer: Composition View

In the Assembly Designer, there are two main views: “Composition” tab showing service composition with service elements including their properties and relationships and “Behaviour Testing” tab for designing, viewing, and executing testing scenarios. Toggling between these two views can be done using the tab labels at the bottom of the page.

The main part of the “Composition” tab is the design canvas presenting the service elements and their relationships. To the right of the design canvas there is a collapsible right-hand panel allowing the properties and relationships of the currently selected element to be viewed and modified.

The right-hand properties and relationships panel is collapsed/expanded by clicking on the 3 horizontal line “Menu” icon to the right of the “Save” button or by using the key combination <Ctrl> + <Shift> + 2.

When in view the right-hand properties and relationships panel contains individual collapsible sections for Properties, Relationships, Operations, Metrics, Clustering and Healing. These sections will only be shown if they are applicable to the service loaded into the Designer so you may only see a subset of these. The collapse/expand state of these panes within the panel is toggled by clicking in the header bar of each pane. In the darker header area of the properties and relationships panel is the name and description of the selected element. The description can be updated by clicking on the pencil icon shown to the right of the description text.

The contents of the lower portion of the properties and relationships panel can be scrolled independently of the design canvas. This is achieved either using the scroll wheel of the mouse or by dragging the slim scrollbar shown on the very right of the panel.

A new property can also be created from within the properties and relationships pane by scrolling down to the bottom of Properties pane and clicking the “+ Add New Property” link.

To provide even more space for the design canvas the left-hand navigation menu can also be collapsed out of view by clicking the “X” symbol in the top right-hand corner of the navigation menu. Doing so will display a 3 horizontal line “Menu” icon in the top left of the Designer canvas. Click this icon to bring the navigation menu back into view.

On the design canvas the contents of the current view can me moved by clicking in any empty part of the canvas and dragging the mouse whilst keeping the mouse button pressed.

You can zoom in/out of the service design using the scroll wheel of your mouse.

A service element can be selected by simply clicking the element symbol on the canvas. Once selected the element will be expanded automatically and the right-hand properties and relationships panel will be brought into view if it is not already visible.

To collapse an expanded service element simply click on the small up arrow enclosed within the coloured semicircle in the middle of the bottom bar of the expanded element border.

New elements, either assemblies or external references, can be added to the design by clicking the “Add Element” button at the top of the design canvas.

Saving changes to the design is done by clicking the “Save” button in the design canvas header bar, which saves the current design to the IBM Agile Lifecycle Manager catalog.

Properties of the service can be edited through the right-hand panel of the designer. New relationships can also be introduced by connecting the dots attached to opened element symbols on the design canvas.

For consistency, changes can be made only to the opened design (assembly). This includes the properties of the opened service itself and the relationships between its comprising elements. The assemblies that are defined as elements of the service cannot be changed without opening them to the designer independently.

### Designer: Behaviour Testing View

In the Assembly Designer, there are two main views: “Composition” tab showing service composition with service elements including their properties and relationships and “Behaviour Testing” tab for designing, viewing, and executing testing scenarios. Toggling between these two views can be done using the tab labels at the bottom of the page.

The Behaviour Testing view is itself made up of three different pages/views which are selected using a further set of tabs shown at the top of the page:

- Assembly Configurations – defines an Assembly Descriptor configured with specific property values to be used in a Behaviour Test Scenario.
- Scenarios – set of individual Behaviour Tests.
- Results – outcome of running Behaviour Test Scenario(s)

**Assembly configurations**

In order to use assemblies in behaviour testing scenarios, either in the role of Service Under Test, or as a test setup, requires the corresponding assembly configuration to be made on the “Assembly Configurations” page of the Behaviour testing tool. This is the page that you will see by default when clicking on the “Behaviour Testing” tab. Existing assembly configurations are listed on the page and can be opened for editing or deleted from the “more options” menu (accessed by clicking on the ellipsis icon “...”) at the end of the corresponding row. A new assembly configuration can be created by clicking the “Create Assembly Configuration” button above the list of existing assembly configurations. The assembly configuration defines the assembly descriptor to be used, a name and a description of the configuration, as well as property values to be used when instantiating an assembly based on the configuration.

**Scenarios**

Defined testing scenarios are listed on “Scenarios” page of the Behaviour Testing tool. The available Behaviour Test Scenarios are presented in a list displaying the following information for each Scenario: • Scenario Name • Description of the Scenario • Number of stages within the Scenario The list is presented sorted alphabetically by Scenario name. It is currently not possible to change this sort order. A scenario can be opened for viewing and editing by clicking the associated “Open” button. Editing the name or description of the scenario or deleting a scenario can be done through the “more options” menu (accessed by clicking on the ellipsis icon “...”) at the end of the corresponding row. Creating a new testing scenario can be done by clicking the “Create Scenario” button above the list of existing scenarios.

**Results**

The “Results” page of the Behaviour testing tool shows a summary of results of the latest testing scenario executions. The available results are presented in a list displaying the following information for each execution:

- Scenario Name
- Description of the Scenario
- Date and time the Scenario was executed
- Overall result of the Scenario execution

Further details of each execution can be viewed by clicking the “Open” button on the corresponding row. The “more options” menu (accessed by clicking on the ellipsis icon “...”) allows an execution to be removed from the list.

**Test Scenario Design**

The Scenario Design page allows editing of an existing scenario or designing a completely new Behaviour Test Scenario. Scenario design can be accessed through the Scenarios page of the behaviour testing tool either by clicking the “Create Scenario” button to create a new scenario or clicking the “Open” button to edit an existing one.

To maximise the space available to create new test scenarios the Scenario Design page does not show the “Assembly Configurations”, “Scenarios” and “Results” tab navigation under the header bar. To return to the list of available test scenarios you need to click on the left pointing arrow shown to the left of the test scenario name in the Scenario Design header.

A Behaviour Test Scenario comprises:

- An Assembly against which the test is to be run
- A list of sequential test “stages” that are configured to perform specific actions

The Scenario Design page comprises of a design area showing the detailed definition of the designed test scenario and a right-hand panel called “Scenario Palette” providing access to available types of test stages that can be included in the test scenarios.

The available test stages are organised into several categories:

- Assembly Configuration
- Assembly Events
- Intent Engine
- Intent Requests
- Metric Assertions
- Metric Definitions
- Metric Recording
- Utilities

The contents of each category can be viewed by expanding the category section in the right-hand Scenario Palette. The expand/collapse state of the test stage categories is changed by simply clicking the category name.

The individual test stages are written to support Behaviour-Driven Testing and use the Gherkin-like “Given – When -Then” structure. They are deliberately designed to be human-readable.

New test stages are added to the scenario by drag-and-drop from the Scenario Palette to the design area. The detailed definition of the test stages is completed in the design area by filling in the text boxes and selecting the applicable options from drop down menus.

The current scenario design can be changed by clicking the “Save” button on top of the design area. Clicking the “Run Scenario” button will start execution of the scenario and opens a view to see the step by step progress of the test.

When a scenario is executed it is added to the list of executions on the “Results” page and corresponding test results can be viewed by opening the execution.

## System

The System section contains tools for systems administration to make IBM Agile Lifecycle Manager aware of underlying available deployment locations where service components can be instantiated and resource managers managing specific VNFCs on defined deployment locations

### Resource Managers

The Resource Managers page lists the available resource managers that have been introduced to IBM Agile Lifecycle Manager. A resource manager is associated with a specific type and URL defining where the resource manager can be accessed.

The list shown on this page displays the following information for each Resource Manager:

- Resource Manager Name (given to IBM Agile Lifecycle Manager when the Resource Manager was added)
- Resource Manager Type (e.g. Ansible RM)
- Resource Manager URL
- An ellipsis giving access to the actions that can be performed on the Resource Manager

The list is sorted alphabetically by Resource Manager name. No other sorting options are currently available.

New resource managers can be added to IBM Agile Lifecycle Manager by clicking the “Add” button in the top right corner of the page.

The more options menu in the right end of each row enable options to:

- Edit the resource manager details
- Refresh the resource manager details and information on the resource types it can manage
- Get information on the deployment locations onto which the resource manager can deploy resources
- Remove the resource manager from IBM Agile Lifecycle Manager.

### Deployment Locations

Deployment Locations page provides details on deployment locations where IBM Agile Lifecycle Manager can deploy resources into. A deployment location is associated with a specific infrastructure type and resource manager that can manage VNFCs running on a specific infrastructure.

The list shown on this page displays the following information for each Deployment Location:

- Deployment Location Name (given to IBM Agile Lifecycle Manager when the Deployment Location was added)
- Resource Manager managing the location
- Infrastructure type used at the Deployment Location (e.g. Docker)
- An ellipsis giving access to the actions that can be performed on the Deployment Location

The list is sorted alphabetically by Deployment Location name. No other sorting options are currently available.

New deployment locations can be defined in IBM Agile Lifecycle Manager by clicking the “Add” button in the top right corner of the page.

The “more options” menu (shown as an ellipsis) provides options to either edit the deployment location details or remove the deployment location from IBM Agile Lifecycle Manager’s available deployment locations.

## VNF onboarding

IBM Agile Lifecycle Manager includes several tools and the CI/CD Hub reference environment for the development of new VNF and Network Services.

The tools that are used to develop VNF and Network Services are:

- IBM Agile Lifecycle Manager with Resource Manager setup (as well as the deployment locations)
  - CI/CD Hub provides you means to create your personal development environment (All-In-One IBM Agile Lifecycle Manager deployment and lmctl command line tools)
- LMCTL provided with IBM Agile Lifecycle Manager is a tool to deploy VNF and Network Services artefacts to target environments

A user develops new VNFs in their personal development environment. LMCTL tools can be then used to export and import VNFs artefacts between other IBM Agile Lifecycle Manager environments.

### Anatomy of a VNF

A VNF, or Virtual Network Function, provides a single function within a Network Service. A VNF consists of one or more VNFCs (Virtual Network Function Component).

A top-down overview of a VNF is:

- A Network Service (defined via an Assembly Descriptor) is made up of one or more VNFs
- A VNF (also defined via an Assembly Descriptor) is made up of one or more VNFCs
- A VNFC (defined via a Resource Descriptor) is made up of:
  - Lifecycle Ansible scripts
  - One or more VDUs (Virtual Deployment Unit) - Each VDU will be tied to a specific infrastructure type.

Starting from the bottom up, these are the components and their definitions:

**VDU**
A VDU is the package or script that can create an image for specific virtual infrastructure.

**VNFC**
This is the VNF component as a standalone network function that is managed by a Resource manager. In IBM Agile Lifecycle Manager this is also called a Resource.

- VDU or multiple VDUs
- Descriptor (Resource) is a yaml file that describes the interface for a VNFC. Contains properties and other details to describe the interface for the item.
- Lifecycle scripts. Scripts that map the standard lifecycle actions as start, stop, install, configure, stop to the VDU specific instructions, e.g. Ansible playbooks that drive the

**VNF**
Provides a single function within a Network Service.

- VNFC. A VNF can consist of multiple VNFCs
- VNF Descriptor. An assembly.yaml file that describes the interface for a VNF. Contains properties and other details to describe the interface for the VNF. Implemented within IBM Agile Lifecycle Manager by an assembly.
- VNFC. One or more VNFCs
- Service Behavior Test scenarios (if any)

## Creating a VNF structure

When creating a VNF, you begin by creating a structure.

### Before you begin

**Remember:** You must have a working IBM Agile Lifecycle Manager environment, and have installed the right tools (such as LMCTL) on your machine.

### Procedure

Create a new VNF by using this LMCTL command:

```
lmctl project create --version *1.0* --servicetype VNF --name *name*
```

The result of this command is a new folder with some artifacts already created ready for population as the VNF and associated VNFC are created. This step also creates stub descriptors that will work with the IBM Agile Lifecycle Manager tools.

## Creating image from VDU for each VNFC

A VDU is responsible for creating images that contain the required software for the VNFC and the target virtual infrastructure that is being worked with. This may be supplied by the vendor of the VNF. The images associated with the VDU will have to be either deployed to an image repository that the VIM used in the deployment can see, or the images will have to be pushed to the required VIM location.

### About this task

Tools such as Packer can be used to create images from the VDU for different target VIMs. These images will contain the base software for the virtual machine to run. VDUs will need built images to be distributed to the various environments. The images can be stored in a repository and when standing up for example a docker container the image will be pulled down from the repository.

The first time an instance of a VDU is requested for a location the image will be pulled from the repository that manages the images. Subsequent instantiations will use the image that already exists. In some environments it is preferable to have pushed the images to the locations prior to their use, for example if they are very large in size.

Very often ready-made images are provided by the VNF vendors for specific target platform and no VDU is available. In this case, you don’t use the VDU folder in the VNFC and simply specify the link to the image in the VNFC descriptor as below.

### Procedure

To specify the link to the image:

```
image:
    type: string
    description: "the image name for target VIM"
    default: image location
```

## Creating lifecycle scripts and operations for each VNFC

This step is to map the IBM Agile Lifecycle Manager standard lifecycle scripts to the specific commands for that VDU. Wrapping each VNFC specific lifecycle script in the standard IBM Agile Lifecycle Manager lifecycle model means that each VNFC will look the same and the Intent Engine can use these in its automated opinionated patterns.

### About this task

For each of the lifecycle steps and operations (to be used in relationships) a ‘script’ needs to be created. For example if you are using the Ansible Resource Manager, each lifecycle script maps to an Ansible playbook that is called the same as the lifecycle step.

The following scripts should be edited for the specific VNFC and be placed in the lifecycle folder in the VNFC:

- `Configure.yml`
- `Install.yml`
- `Integrity.yml`
- `Start.yml`
- `Stop.yml`
- `Uninstall.yml`

A VNFC does not need to implement all lifecycle scripts. In addition to the standard lifecycle scripts, specific operations may also be added. Operations are bespoke transitions that are invoked by the Resource manager when the IBM Agile Lifecycle Manager is establishing relationships between two VNFCs. These allow for action to take place between the VNFCs to enable a Network Service to work. Operations and their definitions are part of the public interface of a VNFC. An example is where VNFC 1 has to add itself to the IP pool table of VNFC2.

### Procedure

Create an operation by creating the script in the lifecycle folder where the name of the operation is the name of the script.

## Creating a VNFC descriptor

Each VNFC requires a descriptor in the YAML format. This descriptor is stored in ‘descriptor’ directory of the VNFC. The format of these files is defined by the IBM Agile Lifecycle Manager Resource Descriptor YAML Specification.

### About this task

Each VNFC must have a name and a version - it is advisable to name the VNFC assuming other people might use the same name, that is, `companyname.resourcename`. Any change to any part of the VNFC should result in creating a new VNFC with an updated version number.

Each VNFC may have a set of properties. These can be required or optional. Default values can be supplied.

**Note:** The resource_id property that has a value set to “$\{instance.id\}”. When the resource is created this will have the ID for the resource that the IBM Agile Lifecycle Manager has assigned it. Other options are “$\{instance.id\}” that will have the IBM Agile Lifecycle Manager name for the resource and “$\{instance.index\}” which will have the unique index number for a resource when more than one can be created by the IBM Agile Lifecycle Manager within an assembly.

*Table 5. Selected VNFC descriptor sections*

| Descriptor | Description |
|---|---|
| Name | Unique name and version number of the resource<br><br>**Note:** Each change in the underlying VNFC components should result in a new version/name of the VNFC. |
| Description | Description of the VNFC |
| Properties | Properties and values for the VNFC such as image location, networks, information for metrics and ID |
| Lifecycle | Lifecycles that apply to this resource |
| Metrics | Metrics such as integrity and publication period |
| Policies | Policies using these metrics such as heal based in the integrity metric |

*Table 5. Selected VNFC descriptor sections (continued)*

| Descriptor | Description |
|---|---|
| Operations | Transitions that are invoked when establishing relationships |

**Metrics and policies**

Each VNFC may emit metric information to help in its management. IBM Agile Lifecycle Manager is expecting all collected metrics from the VNFCs to be made available on Kafka. IBM Agile Lifecycle Manager listens to specific Kafka topics for events containing metrics. Metrics should be associated with necessary identifiers including timestamps, names of the metric and “Metric Identifiers” specifying the source resource of the metrics.

There are different scenarios on how the metrics can be published to Kafka depending on the capabilities of the Resource (I.e. VNF, PNF, etc.):

- The VNFC is capable of publishing metrics directly to Kafka in required format
- A metric collector is built and instantiated next to the VNFC to gather the metrics from the resource, transforming them to applicable format and publishing them to Kafka for IBM Agile Lifecycle Manager to consume. Metric collector can use various methods to get metrics from the resources including SNMP, log files, database queries, etc. The metrics need to be sent to a shared Kafka bus. To make this easier the IBM Agile Lifecycle Manager provides a microservice called the Metric Relay. It takes simple HTTP Rest requests and dispatches them to the Kafka bus. This saves the VNFC from having to have direct access to the Kafka bus.
- Specific monitoring resources can be instantiated to collect metrics on their own and publishing them to Kafka. A monitoring resource can be e.g. a passive probe monitoring network traffic and producing events and counters based on the monitored traffic.

Once the metrics are published to specific Kafka topics, IBM Agile Lifecycle Manager knows where to listen those and based on the Metric Identifier it can also associate the metrics with the correct resource. Received metrics can then be used by IBM Agile Lifecycle Manager on a VNF and Network Service for resource health monitoring, scaling policies, etc. purposes.

IBM Agile Lifecycle Manager supports two main metrics by default: integrity and load:

- Integrity enables a VNFC to be monitored for health. This can either be handled by a local process on the VNFC sending out details of its health. The other way is to use an external VNFC or VNF that monitors one or more VNFC and dispatches the required metrics on its behalf.
- The load metric allows the IBM Agile Lifecycle Manager to manage scaling of clusters based on a simple percentage value. The load metric can be sent by one VNFC and it can then be managed by a policy in a VNF or network service to scale another VNF or VNFC. Like the integrity metric it can be handled by a local process within the VNFC that is monitoring the load or by an external metric collector. External metric collectors will be discussed in a subsequent lesson.

A metric is defined in the VNFC Descriptor:

```
metrics:
 lb_integrity:
 type: metric::integrity
 publication-period: ${integrity_publication_period}
 lb_load:
 type: metric::load
 publication-period: ${load_publication_period}
```

In the VNFC Descriptor, you have to define the Integrity Policy. For Load, that is defined in the VNF or Network Service descriptor.

The Integrity policy in the VNFC Descriptor can be seen in the example below:

```
metrics:
  h_integrity:
    type: "metric::integrity"
    publication-period: "10"
policies:
    heal:
      metric: "h_integrity"
      type: "policy::heal"
      properties:
        smoothing:
          value: ${integrity_smoothing}
```

This shows the metrics section in the VNFC Descriptor that defines the metric called h_integrity. For the integrity metric a policy is also included that defines the number of consecutive smoothing periods that will pass before the VNFC will be either marked as BROKEN when the BROKEN integrity messages have sent, or if the VNFC has been missing.

**Operations**

Operations are bespoke transitions that are invoked by the Resource manager when the IBM Agile Lifecycle Manager is establishing relationships between elements. These allow for action to take place between the VNFCs to enable a service to work. An example of an operation is if one VNFC requires the IP address of another VNFC in order to add it to its pool. See below example for a voice gateway that requires the IP address and port of the Asterisk voice server. This operation is then later used once in the Network Service Design.

```
operations:
  addToDispatchList:
    properties:
      ipaddr:
        type: string
        description: address of the asterisk to add to pool
        required: true
      port:
        type: string
        description: port of the asterisk to add to pool
        default: 5060
        required: true
```

## Onboarding VNFC to IBM Agile Lifecycle Manager

Onboarding is the process of loading details of a VNFC into the IBM Agile Lifecycle Manager and the Resource Manager, so that both can manage the VNFC.

### About this task

The steps taken to onboard a VNFC are:

- Build a package from the source folders of the VNFC
- Onboard the package to the Ansible Resource Manager so that it can now handle that VNFC type
- Refresh the details of the packages supported by the Ansible Resource Manager instance in the IBM Agile Lifecycle Manager

### Procedure

You onboard a VNFC by running a command in the project folder to deploy this VNFC to the “dev” environment.

In this example we are using the Ansible Resource Manager to which we will onboard the VNFC. The VNFC descriptor is pushed to IBM Agile Lifecycle Manager and the Resource Manager.

```
lmctl project push dev
```

## Designing VNF

Once the VNFC has been onboarded, you can then move to the Lifecycle Manager UI to design the VNF.

### Before you begin

To create the VNF, the VNFC needs to have been onboarded.

**Procedure**

**Create VNF assembly**

1. In the IBM Agile Lifecycle Manager UI, create a VNF by going to the Create Assembly section in the Assembly Designer and clicking “Create”.
2. Provide the name, the version number, and a free text description of the VNF.

   This creates the VNF Descriptor.
3. To add an element such as a VNFC, click the “Create element” button.

   This opens up a new dialog where you need to provide a name.
4. You also need to select the Descriptor you want to use for that element, and whether it is a component or a reference.

   You will then see the element being added to the assembly.

   Click on the element to expand it and display the VNFC properties.
5. The next thing to do is to add the Properties that are needed for the VNF to run.

   For example, you can add properties such as a management network, a data network, a management and data address, the deployment location, what Resource Manager to use, or the Image location of the VNFC.

   a) Clicking on the Assembly Descriptor box on the left in the Designer will open up a panel on the right.

      In this panel, you are able to Add new properties.

   b) In the Add new Property dialog you give the property a name, a value, a default value, and you check whether the property is required and/or locked. You can also provide a short description of the property.

   c) Once you click Create, the property you just created will appear in the VNF box in the Designer.
6. The next step is to pass in properties between the VNF and the VNFC or element.

   For each property you will notice a greyed-out ball on the front and at the back of that. A ball icon on the left side of the property means it is an INPUT, the ball icon on the right side means it is OUTPUT.

   - You can pass in a property from one element to the other by simply dragging and dropping.

**Metrics and policies**

This section describes how metrics and policies are used on the VNF level. As described in the VNFC Descriptor section, IBM Agile Lifecycle Manager supports Load and Integrity as the standard metrics that are defined in the VNFC Descriptor. The policy for Integrity has already been defined in the VNFC Descriptor and will show up as properties for the VNFC. The policy for Load is defined on the VNF level.

In order to make metrics available on a higher level than the VNFC, (VNF, Network Service), they need to be promoted first. In this example, we want to use the Load metrics defined in the VNFC Descriptor to use it for a scaling policy in the Network Service.

7. We first need to promote the metrics from the VNFC to the VNF. You can do this by selecting the “...” button and selecting Promote Metrics.

   Once you have done this, you can see the Promoted Metrics section if you click on the VNF section in the VNF Assembly Design.

**Operations**

In the same way as the Metrics, the Operations that were defined in the VNFC Descriptor that need to be used in a higher level (VNF and/or Network Service), need to be promoted first.

8. You can do this by clicking the ... button in the Operations Property section and clicking Promote Operation.

   You will then see the Promoted Operations section in the Property section of the VNF in the VNF assembly design.

**Relationships**

Relationships are typically used in Network Services but can also be present at the VNF level. A VNF can for example consist of multiple VNFCs. You can add multiple VNFCs by adding new elements in the VNF

Design. It is possible to establish relationships between the VNFC elements, for example if one VNFC needs to be installed before the other, or if an operation needs to be performed.

9. In order to establish a relationship, you can drag and drop the orange ball on the element in the design between the two VNFC elements.

   Using the orange ball on the left of the element means a FROM relation that is then dragged to the output orange ball on the right side of the other element.

   Once the line is drag-and-dropped, a new screen pops up where you define the relationship.
10. You can enter the name of the relationship, the from and to elements, whether it needs to be a pre or post condition and select the source and target state of the element.
11. You can then also select the operations that need to be carried out once the relationship is created and when it is removed (that were promoted in the earlier part of the design from the VNFC to the VNF level).
12. You can then also provide additional properties and values.

**Clustering**

A cluster and scaling policy can be defined for a VNFC (or VNF in a Network Service). As described above, the (promoted) Load metric is used to automatically trigger a scale in or out. This can be the metrics of the VNFC, or it can be scaled based on a metric from another element (in a Network Service for example).

13. In order to design a cluster and scaling policy, you click the ... button and select Edit.

    A new popup window appears.
14. You first select what the cluster looks like.
15. You define the Initial quantity that is instantiated once the VNF is instantiated, the scaling increments, and the minimum and maximum number of nodes.
16. You can then select the auto-scaling policy and provide a name for the policy.
17. You select the Load metric that is used, and the scale-out and scale-in thresholds and smoothing periods.

**Create Behavior Driven test (optional)**

These behavior-driven tests are optional, so you can proceed without having implemented these tests.

The integrated behavior driven test tools in IBM Agile Lifecycle Manager are further explained in a later section in this guide.

18. The next (optional) step is to create test scenarios for the VNF that can be used to drive behavior driven tests. You can include a test assembly, for example if you want to add a probe for monitoring or a traffic generator to drive traffic through the VNF), the test scenarios you want to be tested (for example scale-in and scale-out or heal) in order to test all possible lifecycles for that VNF.
19. With the CI/CD hub or integration with an existing CI/CD environment, these tests can then be automatically run as part of a pipeline that builds and tests the VNF when a change was committed.

## Onboarding VNF

Once completed, a VNF can directly be used in your local IBM Agile Lifecycle Manager to be instantiated, or to be used as part of another Network Design.

### Procedure

1. If you are happy with the VNF Design, or if you have made changes in the VNF Designer that you want to commit, you can use the LMCTL command to pull the project back to your local VNF project directory by running the following command from the VNF project folder:

   ```
   lmctl project pull dev
   ```

   This allows you then to, for example, GIT push the changes back into the main GIT repository.
2. If you want to onboard the VNF to another IBM Agile Lifecycle Manager, then you can simply run a command:

```
lmctl project push AnotherDev
```

## Network service design

A Network Service can be defined as a composition of resources (like VNFs) connected together constructing a coherent service function. On highest level a network service corresponds to an orderable entity that provides end users with an end-to-end service chain.

Network services can be defined in a hierarchical manner. A network service can comprise of **other network services** and/or **individual resources** corresponding to service segments, or providing specific functions, all contributing to the behaviour of the higher layer service. The end-to-end network service behaviour is the result of the combination of the individual network function behaviours as well as the behaviours of the network infrastructure composition mechanism.

The following diagram shows an example of a network service comprising of one lower level network service and three VNFs related together. The sample service also has a dependency to another network service external to it.

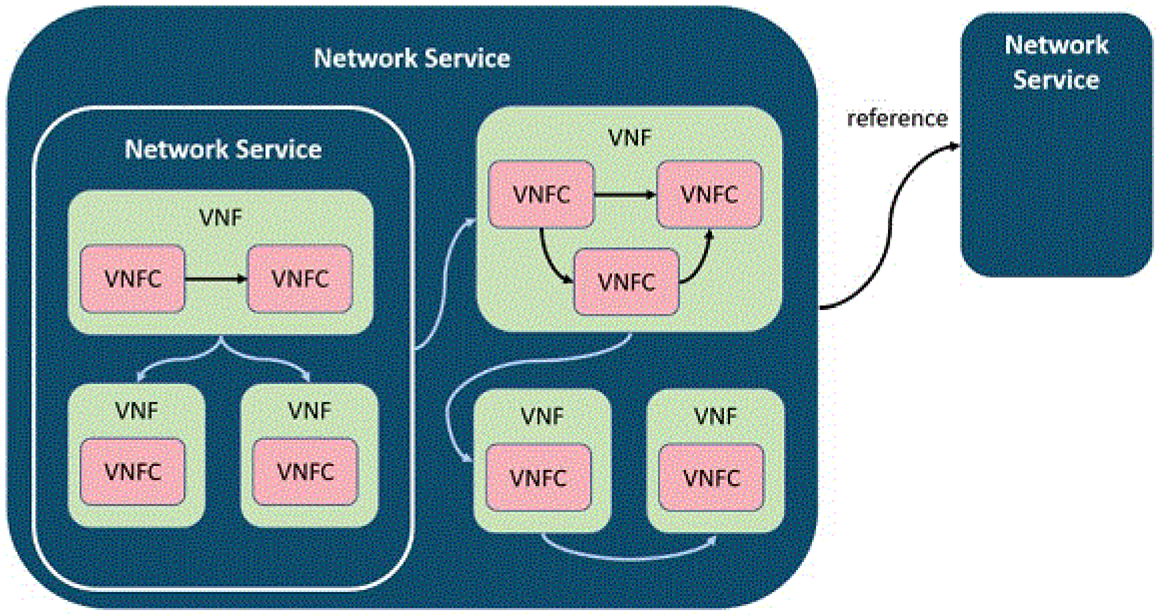


*Figure 1. Network service structure*

The structure of the network service is defined in a network service descriptor that contains the composition of the involved resources (VNFs and/or other network services), their relationships, and applicable policies. The descriptor is defined at the time of service design. For IBM Agile Lifecycle Manager, the network service descriptor functions as a service model used by its intent engine to resolve execution paths to perform specific lifecycle transitions on the service.

A network service descriptor can also contain references to external components. These can be either existing assembly instances or external resources that are managed directly by a resource manager. An example of a referenced resource can be existing network connectivity that already exists and to which the network service should be connected to.

All items referenced in the network service descriptor must be pre-existing before IBM Agile Lifecycle Manager will instantiate any of the network service components included in the network service composition.

## Designing a new network service

The process of designing a new service includes a number of steps, described here.

### About this task

The process of designing a new service consists of the following steps:

1. Define Header Information
2. Add Service Elements
3. Define Service Properties
4. Map Property Values
5. Promote Operations
6. Create Relationships
7. Promote Metrics
8. Define Clusters and Scaling Policies
9. Save Design

The order of individual steps can be changed, and the steps can be performed incrementally and parallel. Also, many of the steps are optional and might not be required for the specific service design. Each one of the illustrated steps is explained in subsequent sections.

### Procedure

1. Define header information of the service descriptor.

   When “Create” button is selected to create a new service design, the following dialog is opened to fill in the required information.
2. Add service elements.

   These can be either native components of the created service, or references to existing service or resource instances.
3. Define service properties.

   Select service by clicking the symbol on design canvas and existing properties will be listed on the right-hand panel. New properties can be added through “Add New Property” option in the end of the property list on right-hand panel. At start the property list is empty and only “Add New Property” option is shown. Typically, the properties needed on the service level includes two mandatory properties: “deploymentLocation” and “resourceManager”. Additionally, there needs to be in minimum the element properties that are required for the elements and not been set a value on the element level.

   For each property a “Property Name” must be given a value and optionally a preset “Property Value”, “Default Value” and “Property Description” (Optional) can be defined. Also, the property can be tagged as “Required” and/or “Locked”.
4. Map property values.

   Mapping the property values is needed to define value passing between the designed service and its components.

   Property value passing is defined by combining two properties together, one as the source and second as the target. A value passing between two properties can be done on the design canvas by clicking the greyed ball icon on the right side of the source property and dragging it on top of the ball icon on the left side of the target property. When this relation is defined the value of the source property will be passed to the target property.
5. Promote operations.

   In some cases, elements have operations to be called during lifecycle transitions. In order to be able to use these operations on higher level services they need to be first promoted. Promoting operations is applicable only in case you anticipate your designed service to be used as a element in a higher level service and you want to make it public on that level.

Available Operations can be found by first selecting the symbol of the element on the design canvas. While an element is selected, its available operations can be seen on the right-hand panel below the operations list. An operation can be promoted by opening the operations list, selecting the "more options" menu associated to the operation, and selecting "Promote Operation" option from the menu.

When you promote an operation you also need to give it a name that functions as the handle when using the operation. This is done through the operation naming dialog.

6. Create relationships

   Relationships are defined to create dependencies between service elements. Relationships between two elements enable the "requirements" of one element (known as the "target") to be satisfied by another element providing the "capability" (known as the "source"). Relationship also define operations to be called by the source against the target element when the relationship is created or ceased. A relationship also implies temporal dependency between the source and the target element when they are instantiated or deleted.

   A relationship between two service elements can be created on the design canvas by clicking the greyed orange ball icon on the right side of the source element and dragging it on top of the greyed orange ball icon on the left side of the target element.

   **Note:** The orange ball icons representing the possible relationships are visible only when the property details of both elements are collapsed.

   Ones the connection is drawn between the source and target elements a dialog is opened to configure the details of the relationship. The configuration is made in three steps.

   a. Source & Target: Naming the relationship, defining the source and target elements, and their temporal dependency
   b. Operations: Defines the operations to be called by the source element against the target element during creation and ceasing the relationship.
   c. Properties: Defining the required properties for the operations.

7. Promote metrics

   In case you anticipate the designed service to be used as an element in a higher-level service and you want to make a metric from any included service element visible for the parent service, the metric must first be promoted.

   Promoting a metric can be performed by first selecting the corresponding element, opening the metrics section from the right-hand panel of the designer, and from more options menu selecting "Promote Metric".

   When you promote a metric you also need to give it a name that functions as the handle when using the metric in the parent service design. This is done through the Metric Name dialog.

8. Define Clusters and scaling Policies

   In order to perform scaling on a service element it needs to be defined as a cluster. By default, clustering is not applied on an element and it needs to be separately configured. Configuring clustering for a service element is done by selecting the element, opening the clustering section on the right-hand panel, and selecting "edit".

   Adding clustering for an element opens a dialog to configure the applicable clustering options and optionally a scaling policy for automated scaling based on load. Clustering definition with an active scaling policy is shown in below.

9. Save design

   Once you are completed with the design, the design can be saved to IBM Agile Lifecycle Manager catalog by clicking the save button in the top right corner of the design canvas.

## Behavior-driven tests

One of the main advantages of IBM Agile Lifecycle Manager is that it has inbuilt behaviour driven test tools. This allows you to automatically run VNF and Network Services though their entire lifecycle.

As part of testing, you can for example instantiate a Network Service and a traffic generator to generate traffic that is required for the tests or instantiate additional probes that are needed for monitoring purposes. You can create many test scenarios such as scale-in and out, heal, or other scenarios. The test can be run manually or automatically as part of a CI/CD pipeline.

Once you have the design ready for your VNF and/or Network Service, you can go to the Behaviour Driven Test section by clicking the “Behaviour Testing” tab at the bottom of the design canvas.

### Creating behavior-driven tests

To create a behavior-driven test, you first create the test assembly configuration and then the behavior test scenario, before running the behavior test scenario

#### About this task

#### Procedure

**Create Test Assembly Configuration**

1. Configure the assembly that you want to use for the tests by pressing the **Create Assembly Configuration** button.

   This opens up a new screen
2. Select the assembly you want to use, and provide a name and description.

   The assembly might, for example, include the Network Service or VNF that you want to test, or you have already created a VNF or Network Service assembly that includes the required test resources such as a probe or a traffic generator. Once you select the assembly descriptor, the properties for that are loading within the same popup window that can be changed and used for that Test Assembly configuration.

**Create test scenario**

3. Click **Create Scenario** and enter a name and description.

   Once you have defined all Assembly Configurations that can be used in the tests that you are developing, you can click on Create Scenario. This is where you define the actual test cases and conditions that should be run. You provide a name and a scenario description. All saved scenarios will show up in the Scenario screen in a list from where you can edit or remove them.

   Once created, you are then presented with the scenario designer.
4. Design the test scenario.

   On the right-hand side you have your palette of steps you can chose to include in your scenario. For example you first chose the test assembly configuration you created before by simply drag-and-drop to the middle canvas. You can then decide what state you want the test assembly in when you start the scenario, and what to do once the scenario is run. After you have selected the assembly configuration, you then need to design the stages for the scenario by drag-and-drop a step from the right-hand side onto the stage. You can define multiple stages, and one stage can include multiple steps.
5. Extend the scenario.

   The behaviour driven test tools also include inbuilt metrics visualisation when the scenario is run. For example if you want to test a scale out and look at the various metrics it is producing while performing the scale test scenario, you can create the scenario so that it instantiates the Assembly, then defines the metrics it should monitor for, then start recording those metrics. You can continue the design by then introducing or changing a resource to produce more test traffic, and then define that you expect to see a scaleout. Then once that is completed you design the steps in the opposite order so that it stops recording the metrics and removes the test assembly.

**Run test scenario**

6. Once you have completed the scenario design, you can proceed to save it and then run the scenario.

   Once you run the scenario, you will see the ongoing scenario screen. On the right-hand side you will see metrics (if any). In the middle you see the stages and steps as it is executing these and you will see if these fail or pass. it will continue to move through the tests until the final step is passed.

   - While the test scenario is ongoing, you can go to Recent Assembly Instances screen in the Operations view, to see the Intent Execution graph to see all the tasks the Intent Engine is performing to automatically bring the assembly configurations through its lifecycle.

   Once the scenario is run, you can see these in the Results page in the Behaviour Driven test functionality.
7. From the Results page, you can open all executed test scenarios or delete them.

**Using Behaviour Driven Testing in your CI/CD process**

8. The test scenarios can be run manually but can also be run automatically.

   As part of a CI/CD pipeline, it can be setup so that the pipeline triggers building the software after a change has been committed, packages and onboards the VNF or Network Service to the appropriate IBM Agile Lifecycle Manager, and then executes the test scenario.

## Operations

One of the main advantages of IBM Agile Lifecycle Manager is that it has inbuilt behaviour driven test tools. This allows you to automatically run VNF and Network Services though their entire lifecycle.

As part of testing, you can for example instantiate a Network Service and a traffic generator to generate traffic that is required for the tests or instantiate additional probes that are needed for monitoring purposes. You can create many test scenarios such as scale-in and out, heal, or other scenarios. The test can be run manually or automatically as part of a CI/CD pipeline.

Once you have the design ready for your VNF and/or Network Service, you can go to the Behaviour Driven Test section by clicking the “Behaviour Testing” tab at the bottom of the design canvas.

## Applications

Once Network Services and VNFs are designed, they can be put into production. The Operations section in IBM Agile Lifecycle Manager allows for assemblies to be searched, and actions to be taken.

### Procedure

1. Find existing Assembly Instances

   - To find an **existing** assembly, you can go to the Recent Assembly Instances section to see the last Assembly Instances that have changed state.

     You see the name of the assembly instance, the status, descriptor name, and the last Intent and time that was executed. For each assembly instance, you have the option to manually change the intent (e.g. make inactive, upgrade, uninstall).
   - To search for an Assembly Instance, click **Search Assembly Instance** in the Operations section of IBM Agile Lifecycle Manager.

   You can open an assembly instance, which will bring you to the topology view. From the Assembly Instance when clicking the **...** button, you can open a current intent execution graph (when running), the assembly design, or view the intent execution history that shows all intents for that assembly instance. You can also Create a new Assembly.
2. Create Assembly Instance

   Once the design is finalized, the first thing you will do is to create an Assembly Instance. You can do this by clicking **Create Instance**. This will ask you to provide a name, select a descriptor, and in what intended state the instance should be in after installation (active, inactive or installed). It will then ask you to change any properties if needed and allows you to review before continuing.

Once you press Create, the Assembly Instance will show up on the Recent Assembly Instances tab and you will see that is being brought into its intended state by the Intent Engine. You are able to follow the Intent Engine execution tasks by going to the Execution Graph (see next section).

3. Intent Execution graph

   IBM Agile Lifecycle Manager includes intent based orchestration capabilities. Once you have onboarded and designed your VNF and Network Services, the Intent Engine automatically calculates and executes the best path to get an assembly from the current state to the desired state without having to manually program any of the tasks. The Intent Execution graph is a view all the tasks that the Intent Engine is executing to bring the Network Service or VNF into its indented state.

   When an Intent is being executed, you can view all the tasks the Intent Engine is performing in real-time in a moving graph. You can see what task is performed first, and what lifecycle action or operation is being performed.

   You can also view the Execution History of an Intent that has been executed. A popup screen will show all the Intents that were run for that assembly and you can select which one to view.

4. Topology

   When you click Open in the Recent Assembly Instance view, you are taken to the Topology view. This shows the Network Service or VNF, and the components inside of that and their status. As before in the design, you will see VNF/NS elements, clustered elements and referenced elements. You can also see the properties and relationships on the right panel and change the Intent. You can also go to the Execution Graph by clicking the tab on the bottom.

## Lifecycle transitions

Intents can be requested through specific API endpoints on the IBM Agile Lifecycle Manager Northbound API. This can be requested by a third-party system, by IBM Agile Lifecycle Manager itself for policy driven intents (such as auto-healing or scaling for example) or it can be manually requested through the IBM Agile Lifecycle Manager User Interface which then calls the corresponding API.

Available API end points to request intents include:

- createAssembly
- changeState
- healAssembly
- scaleInAssembly
- scaleOutAssembly
- upgradeAssembly
- deleteAssembly

The requests are forwarded to IBM Agile Lifecycle Manager’s intent engine to resolve the actual execution path to reach the target state of the intent. With support from other IBM Agile Lifecycle Manager components, like catalog and topology, intent engine resolves the necessary network service level actions, breaks them down to VNF and eventually to VNFC level lifecycle transitions, and figures out the correct order to execute the individual steps. The exact sequence of execution steps depends on the shape of the network service, the current state of the service, and the requested target state of the service. Eventually these VNFC level steps are requested to be executed by underlying resource manager(s) through IBM Agile Lifecycle Manager's southbound API (RM API).

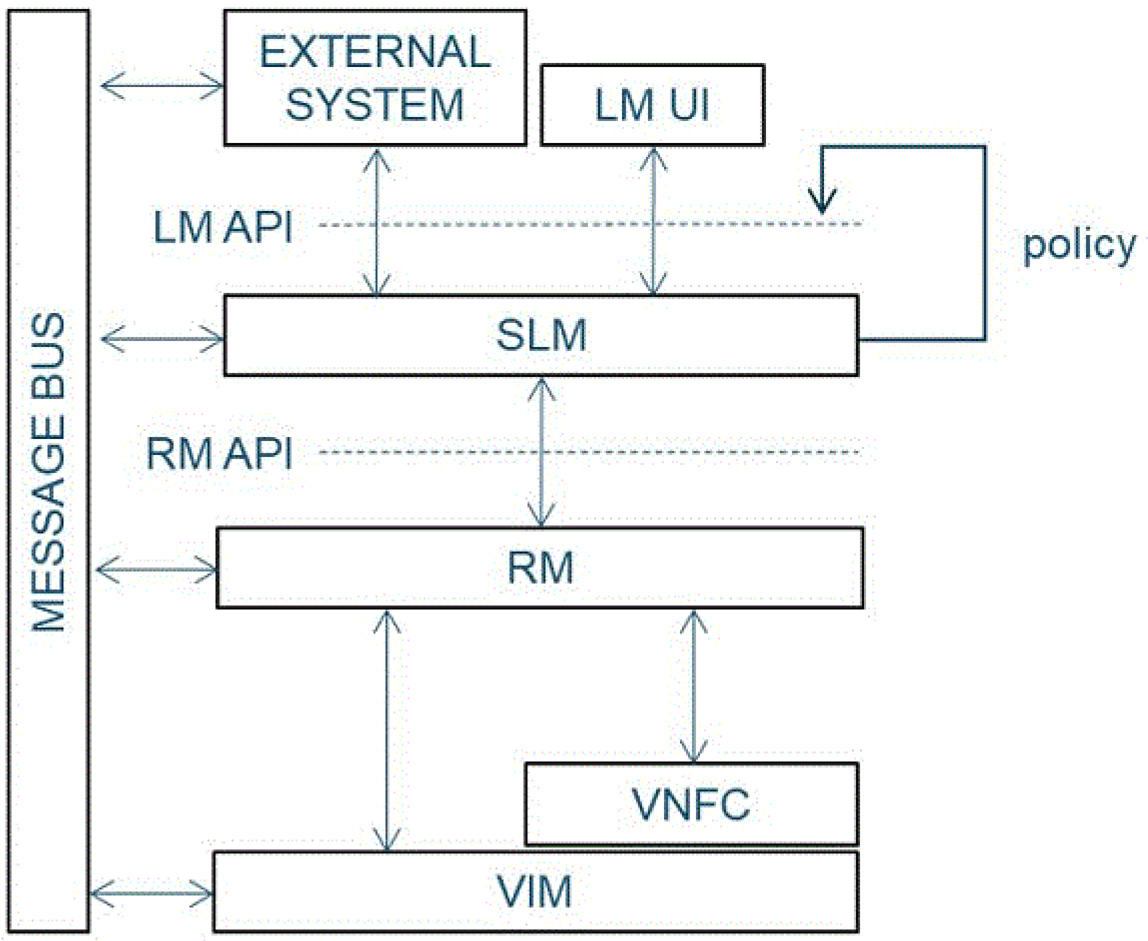


*Figure 2. The IBM Agile Lifecycle Manager related interfaces participating in intent execution*

In the following sections a set of intent scenarios is illustrated using a sample network service (NS) illustrated in the diagram below. For the sake of clarity the network service used in the scenarios is selected to be very simplistic. The sample NS comprises of one VNF which contains one VNFC. The VNF is defined to be deployed as a cluster to allow scaling of the VNF.

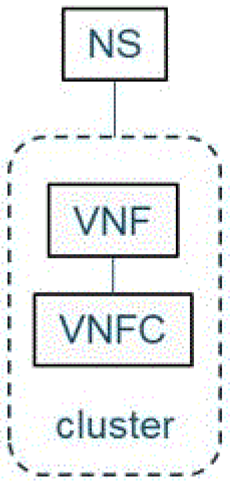


*Figure 3. Sample network service (NS).*

- – API calls are not written out complete, but only the necessary parts to understand the intent are included. See the related links for detailed specification of API endpoints both for the IBM Agile Lifecycle Manager API and Resource Manager API.

– Only successful scenarios are described.

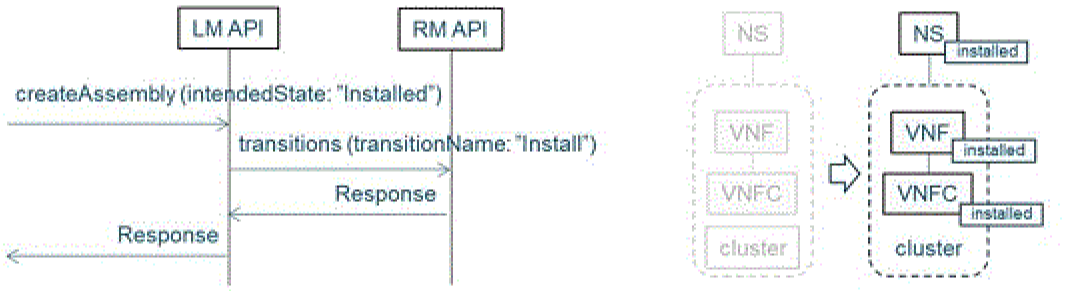
**Instantiating network service**
The purpose of create-intent is to create a new instance of a network service and its associated components. The target state of the intent can vary (installed, inactive, or active), and the actual execution path to complete the intent is different depending on the requested target state.

**Before you begin**
Before a network service can be created, below pre-requisites should be complied:

- The VNFC package has been onboarded to Resource Manager and IBM Agile Lifecycle Manager
- The VNF has been designed and onboarded to IBM Agile Lifecycle Manager
- The NS has been designed and onboarded to IBM Agile Lifecycle Manager

**About this task**

The following diagrams show the sample execution path to create a network service instance and run it to different allowed states: "installed", "inactive", and "active". It should be noted that the execution steps of the intent vary dynamically depending the requested target state.



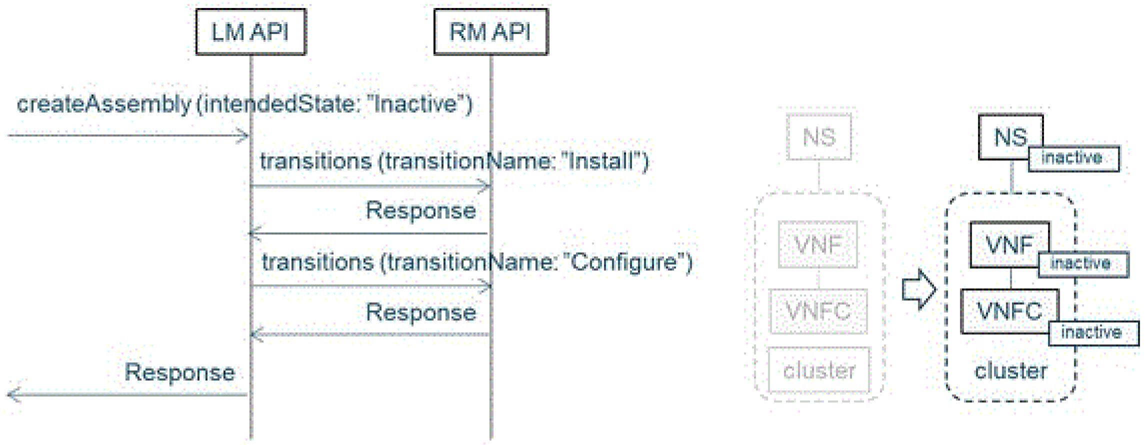

LM API
RM API
createAssembly (intendedState: "Inactive")
transitions (transitionName: "Install")
Response
transitions (transitionName: "Configure")
Response
Response
NS
VNF
VNFC
cluster
NS
inactive
VNF
inactive
VNFC
inactive
cluster

*Figure 4. Create a network service instance with 'installed' state*

*Figure 5. Create a network service instance with 'active' state*

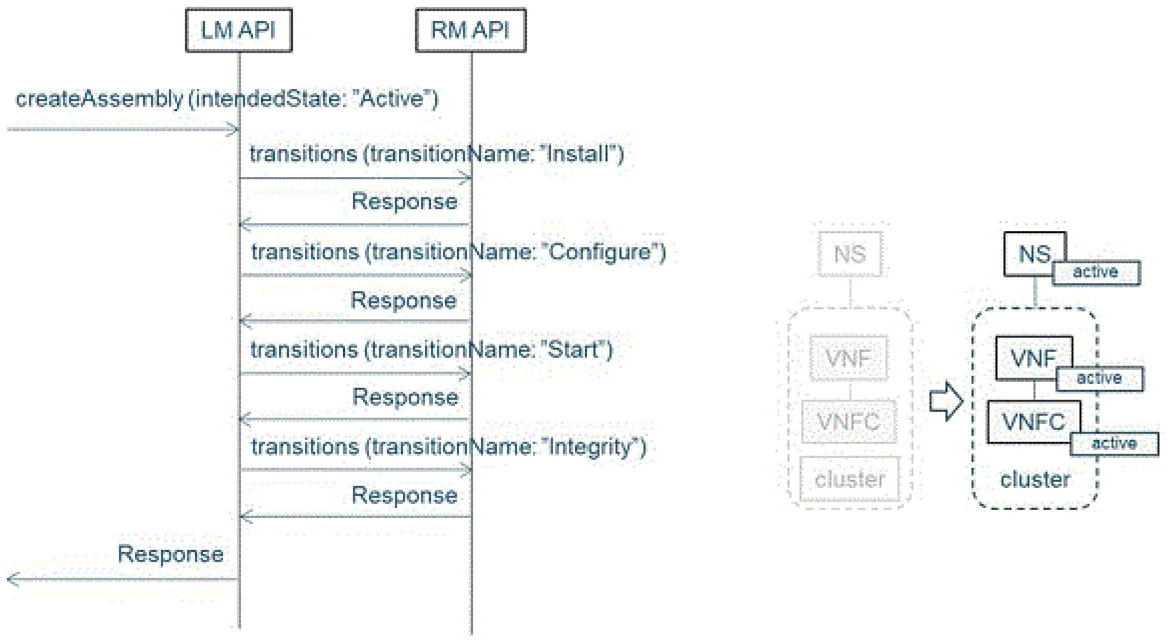

LM API
RM API
createAssembly (intendedState: "Active")
transitions (transitionName: "Install")
Response
transitions (transitionName: "Configure")
Response
transitions (transitionName: "Start")
Response
transitions (transitionName: "Integrity")
Response
Response
NS
VNF
VNFC
cluster
NS
active
VNF
active
VNFC
active
cluster

*Figure 6.*

### Procedure

1. From the IBM Agile Lifecycle Manager GUI, create a new Network Service.

   A new network service instance can be created by clicking the **Create** button in the top right corner of either “Recent Assembly Instances” or “Assembly Instance Search” page.

   A dialog is opened.
2. Fill in details required to create the service instance.

   This includes a unique name for the new instance, descriptor to be used, intended target state, and necessary property values.

   After filling in all necessary data, the definitions can be reviewed before triggering the instantiation.

### Terminating network service

Deletion of a network service instance can be requested on an existing NS instance in the ”installed”, ”inactive”, or ”active” state. Deletion removes all persistent components instantiated on target environment(s) and any references to the service instance.

### About this task

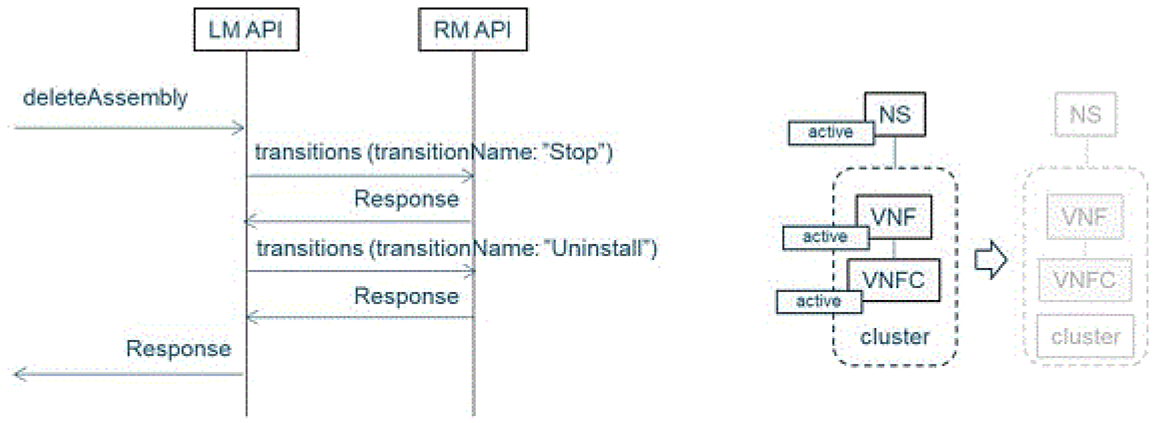


*Figure 7. Delete-intent run on a service instance in the 'active' state*

### Procedure

From the IBM Agile Lifecycle Manager GUI, terminate a network service.

A network service instance can be deleted by clicking the **Change Intent** button associated to the instance and selecting **Uninstall** from the dropdown menu.

The “Change Intent” button can be found on “Recent Assembly Instances”, “Assembly Instance Search”, and opened “Topology” and “Execution” pages.

## Changing network service state

Change-state intent can be requested to move an existing service instance from a state to another. Both the current and the requested target state can be any of the following three states allowed for an existing healthy service instance: ”installed”, ”inactive”, or ”active”.

### About this task

**Note:** The actual execution path is different depending both on the current and target state.

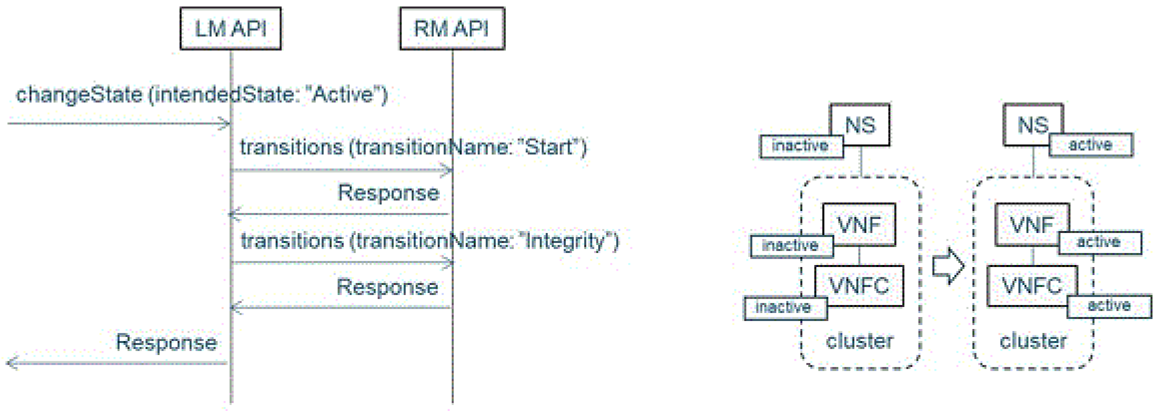

LM API
RM API
changeState (intendedState: "Active")
transitions (transitionName: "Start")
Response
transitions (transitionName: "Integrity")
Response
Response
NS
inactive
VNF
inactive
VNFC
inactive
cluster
NS
active
VNF
active
VNFC
active
cluster

*Figure 8. The transition from 'inactive' to 'active' state*

### Procedure

From the IBM Agile Lifecycle Manager GUI, change a network service state.

The operational state of an existing network service instance can be changed by clicking the **Change Intent** button associated to the instance and selecting desired transition from the dropdown menu.

The “Change Intent” button can be found on “Recent Assembly Instances”, “Assembly Instance Search”, and opened “Topology” and “Execution” pages. The options available in the dropdown menu are dependent on the current state of the service instance.

## Healing a VNFC

Heal can be requested on a VNFC being in the ”active” state. Request can be triggered either through the IBM Agile Lifecycle Manager API, or automatically by IBM Agile Lifecycle Manager itself based on a healing policy set for the VNFC.

### About this task

The healing sequence is adaptive first trying to restart the VNFC by moving it from ”active” state to ”inactive” and back to ”active”.

In case the VNFC is unreachable, it is re-installed by first deleting it and creating a new instance on the same name. After the healing sequence is completed successfully the healed VNFC is running in the ”active” state.

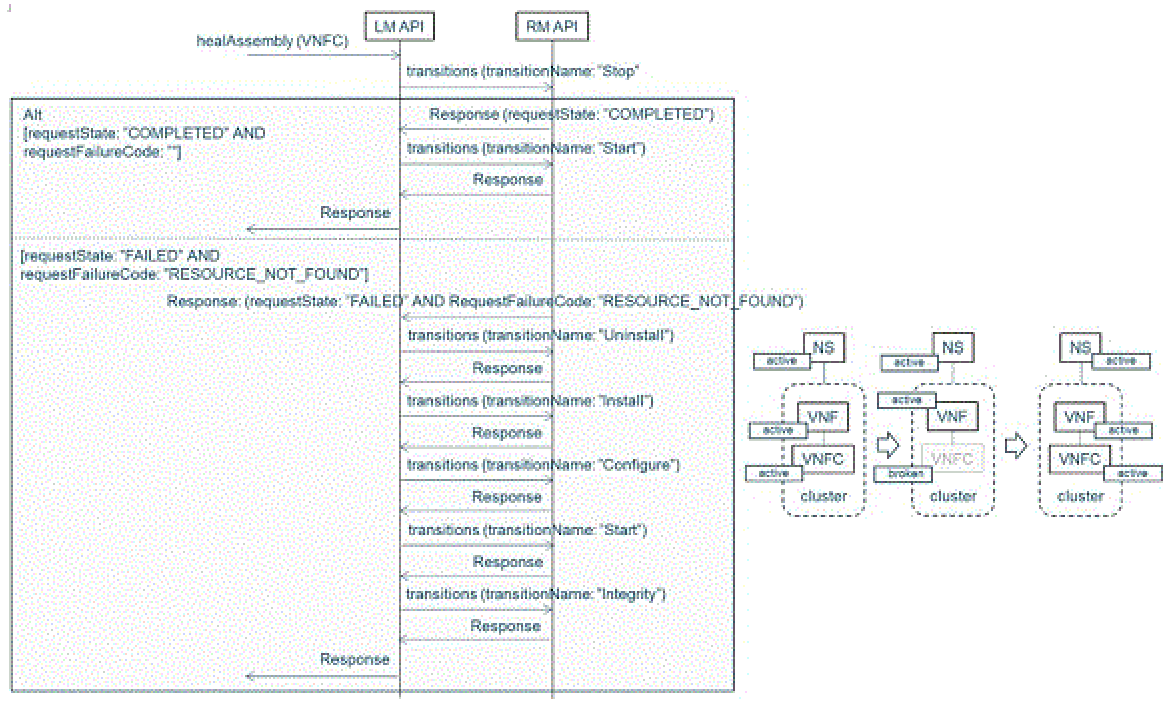

LM API
RM API
healAssembly (VNFC)
transitions (transitionName: "Stop"
Alt
[requestState: "COMPLETED" AND
requestFailureCode: ""]
Response (requestState: "COMPLETED")
transitions (transitionName: "Start")
Response
Response
[requestState: "FAILED" AND
requestFailureCode: "RESOURCE_NOT_FOUND"]
Response: (requestState: "FAILED" AND RequestFailureCode: "RESOURCE_NOT_FOUND")
transitions (transitionName: "Uninstall")
Response
transitions (transitionName: "Install")
Response
transitions (transitionName: "Configure")
Response
transitions (transitionName: "Start")
Response
transitions (transitionName: "Integrity")
Response
Response
NS
active
VNF
VNFC
broken
cluster

*Figure 9. Healing sequence*

### Procedure

From the IBM Agile Lifecycle Manager GUI, trigger 'heal'.

Manual healing for a VNFC can be triggered by navigating to the “Topology” view of a network service and selecting the target element from the service topology.

If the service contains nested assemblies and the actual VNFC is not on the highest level of service elements, lower level service elements can be seen by clicking the icon of the corresponding parent element. Once the target VNFC is selected, the healing sequence can be triggered by clicking the **Heal** button shown in the header of the right-hand panel.

## Scaling a VNF

### About this task

**Scale-in a VNF**

The purpose of scaling-in a VNF is to optimize resource utilization through reducing unnecessary service capacity by deleting one or more VNF instances from a cluster.

Scale-in can be requested on a VNF instantiated as a cluster and running in the ”active” state. The number of VNFs instantiated in the cluster should also exceed the minimum allowed instances.

Scale-in request can be triggered either through the IBM Agile Lifecycle Manager API, or automatically by IBM Agile Lifecycle Manager itself based on a scaling policy set for the VNF.

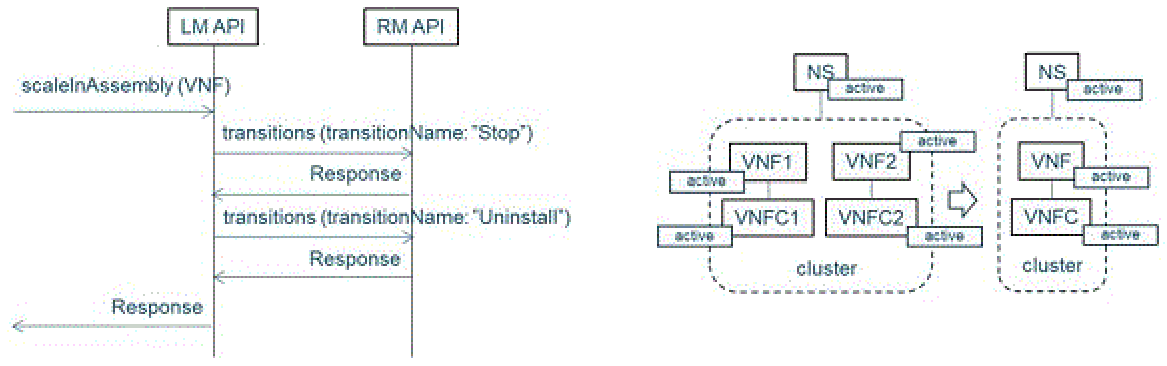


*Figure 10. Scaling-in a VNF*

**Scale-out a VNF**

The purpose of scaling-out a VNF is to increase service capacity to match the offered load by creating one or more additional VNF instances to a cluster.

Scale-out can be requested on a VNF instantiated as a cluster and running in the "active" state. The number of VNFs instantiated in the cluster should also be smaller than the maximum allowed instances.

Scale-out request can be triggered either through the IBM Agile Lifecycle Manager API, or automatically by IBM Agile Lifecycle Manager itself based on a scaling policy set for the VNF.

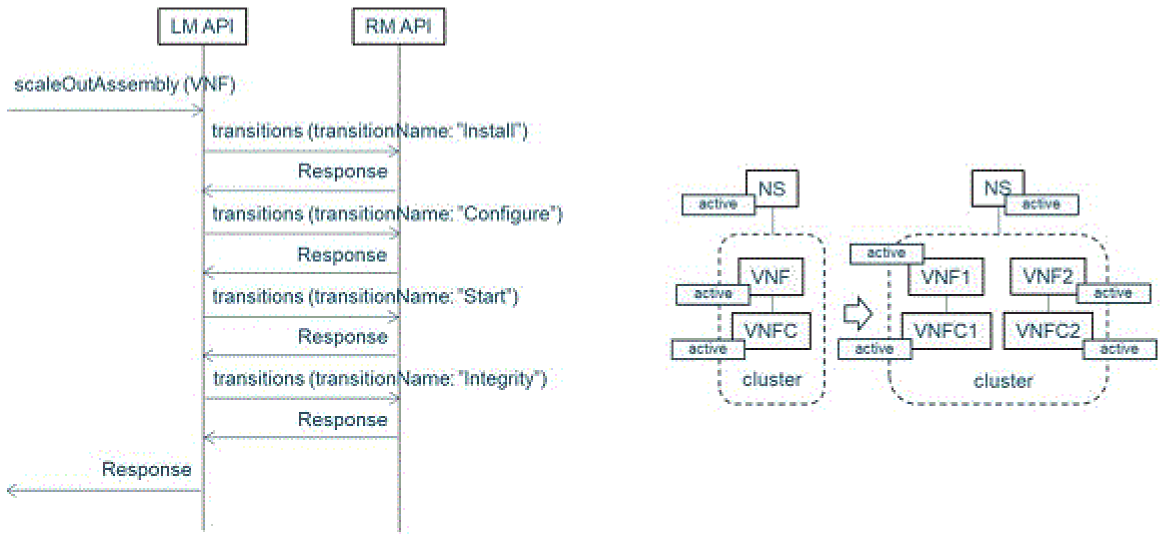

LM API
RM API
scaleOutAssembly (VNF)
transitions (transitionName: "Install")
Response
transitions (transitionName: "Configure")
Response
transitions (transitionName: "Start")
Response
transitions (transitionName: "Integrity")
Response
Response
NS
active
VNF
VNFC
cluster
VNF1
VNF2
VNFC1
VNFC2

*Figure 11. Scaling-out a VNF*

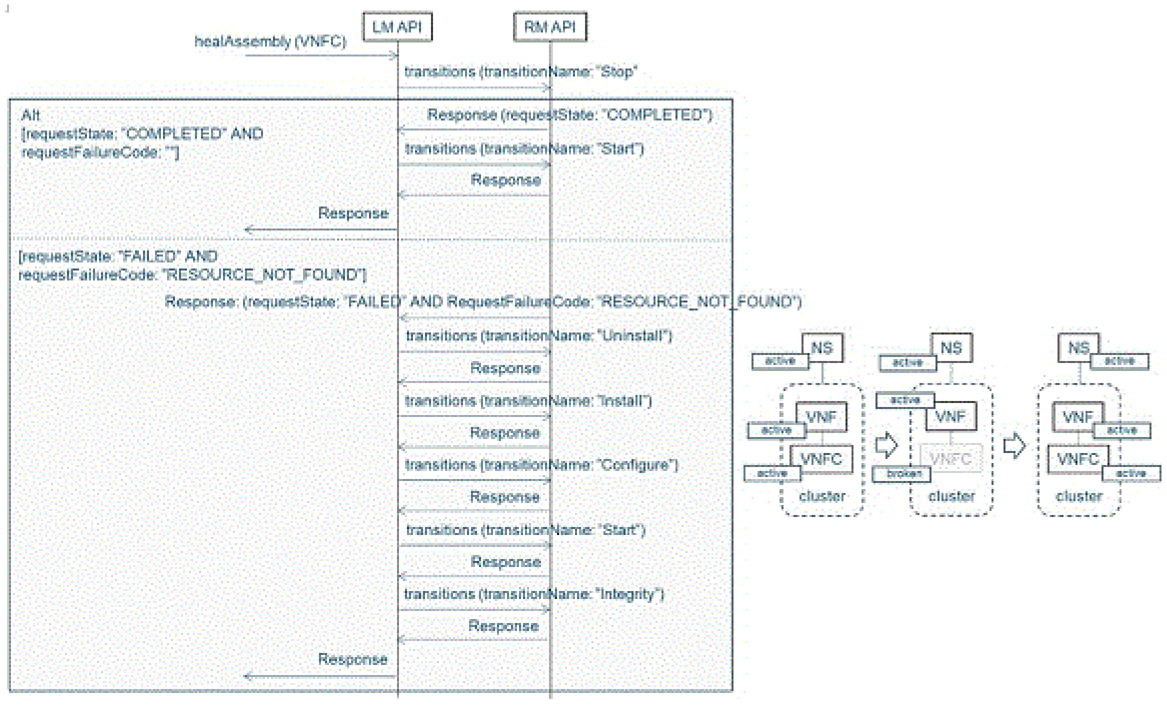

LM API
RM API
healAssembly (VNFC)
transitions (transitionName: "Stop"
Alt
[requestState: "COMPLETED" AND
requestFailureCode: ""]
Response (requestState: "COMPLETED")
transitions (transitionName: "Start")
Response
Response
[requestState: "FAILED" AND
requestFailureCode: "RESOURCE_NOT_FOUND"]
Response: (requestState: "FAILED" AND RequestFailureCode: "RESOURCE_NOT_FOUND")
transitions (transitionName: "Uninstall")
Response
transitions (transitionName: "Install")
Response
transitions (transitionName: "Configure")
Response
transitions (transitionName: "Start")
Response
transitions (transitionName: "Integrity")
Response
Response
NS
active
VNF
VNFC
cluster
broken

*Figure 12. Healing sequence*

### Procedure

From the IBM Agile Lifecycle Manager GUI, trigger scaling-in, or scaling-out.

Similar to healing, scaling of a clustered element can be triggered by navigating to the “Topology” view of a network service and selecting the target element from the service topology.

Scaling is only available for clustered service elements and thus the selected element should be defined as a cluster. Once the target cluster is selected, the scaling sequence can be triggered by clicking either the **“Scale In**” or “**Scale Out**” button shown in the header of the right-hand panel.

## Upgrading network service

The upgrade is requested on an existing service instance running in the ”active” state.

### About this task

The purpose of upgrade is to change the shape of a service instance while keeping the identity of the service instance the same. The change can be made on different levels on the service hierarchy: NS, VNF, or VNFC.

The intended target shape of the service is specified in a new version of the service descriptor given as input in the upgrade request. The upgrade sequence is impacting only the necessary parts of the service according to the scope of required change. The impacted parts of the service are first deleted and subsequently re-created according to the new version of the descriptor.

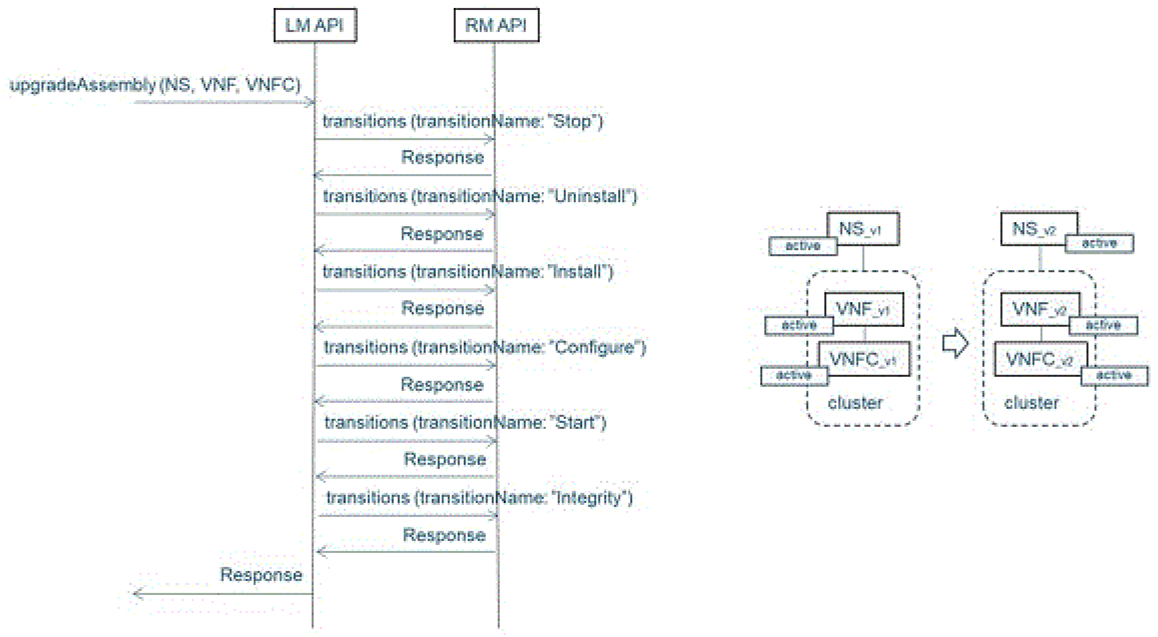

LM API
RM API
upgradeAssembly (NS, VNF, VNFC)
transitions (transitionName: "Stop")
Response
transitions (transitionName: "Uninstall")
Response
transitions (transitionName: "Install")
Response
transitions (transitionName: "Configure")
Response
transitions (transitionName: "Start")
Response
transitions (transitionName: "Integrity")
Response
Response
NS_v1
active
VNF_v1
active
VNFC_v1
active
cluster
NS_v2
active
VNF_v2
active
VNFC_v2
active
cluster

*Figure 13. Upgrading network service*

**Procedure**

From the IBM Agile Lifecycle Manager GUI, trigger upgrade.

Upgrade of a network service instance can be performed by clicking the **New Intent** button associated to the instance and selecting **Upgrade** from the dropdown menu.

The “New Intent” button can be found on “Recent Assembly Instances”, “Assembly Instance Search”, and opened “Topology” and “Execution” pages.

Once “Upgrade” option is selected, a dialog is opened to fill in details about the targeted upgrade. An upgrade can include changing the whole service descriptor and/or changing property values defined for the service. Actual upgrade sequence depends on the defined changes.

# Chapter 5. Getting started using the APIs

IBM Agile Lifecycle Manager provides both a graphical UI and an HTTP API allowing the creation and administration of assemblies. This section describes a set of basic scenarios to get started using the APIs.

## Configuration reference

This topic provides you with an overview of the IBM Agile Lifecycle Manager services settings you need to know when configuring the solution for your own environment, such as port numbers, Swagger URLs, and API details.

### API HTTP calls

Calls to the IBM Agile Lifecycle Manager API are made using either REST or RPC mechanisms, and each call returns an HTTP status code.

The REST APIs are accessed via the Gateway (Ishtar). Use the following command to determine what the url is:

```
oc get routes alm-ishtar -o jsonpath='{.spec.host}' --namespace=lifecycle-manager
```

Nimrod/UI url can be determine by running:

```
oc get routes alm-nimrod -o jsonpath='{.spec.host}' --namespace=lifecycle-manager
```

**Note:** Access to the (Ishtar) REST API is authenticated and must be accessed using OAuth. For deetails of how to do this, see "Authentication" on page 86.

### Service REST API

The Service REST API endpoints are available at the following URLs, and may be used as directed during a product support request:

**Runtime metrics**
http://docker-host:port/management/metrics

**Configuration properties**
http://docker-host:port/management/env

### Kafka

The following Kafka topics are exposed by IBM Agile Lifecycle Manager.

**alm__processStateChange**
State changes relating to Daytona processes, namely lifecycle processes to change state.

**alm__stateChange**
Component state change events, namely state change events relating to assembly and resource instances.

**alm__descriptorChange**
Events relating to any descriptor change (assembly or resource).

**alm__rmTransitionResponse**
Events relating to completed Resource Manager transitions. These events are consumed by Daytona/ Orchestrator and ASM (to build augment its topology model).

**Note:** There are two underscores (__) in the Kafka topics.

## Creating an assembly instance

You create a new assembly instance when you need to deploy a new service described in an assembly descriptor.

### Before you begin

IBM Agile Lifecycle Manager must be installed, with all included resources and test assemblies deployed to the catalog.

### About this task

A new instance of an assembly is created by using the API for Daytona (Orchestrator) service.

This task installs a new instance of a `t_bta` assembly called `test_1`, and then configures and starts it.

This example uses the basic test assembly `assembly::t_bta::1.0`. This assembly is composed of two resources named A and B, both of type `resource::t_simple::1.0`. It references three external resource instances, two networks of type `resource::openstack_neutron_network::1.0` and one image of type `resource::openstack_glance_image::1.0`. It has one relationship from A to B that is created when A and B are active. Resource B is in a cluster which on installation includes a single instance of B.

### Procedure

1. Identify the assembly properties requiring a value when creating a new assembly instance. To do so, explore the corresponding assembly descriptor (in this example `assembly::t_bta::1.0`).

   Retrieve the descriptor from the IBM Agile Lifecycle Manager catalog by running the following query on the Apollo API:

   ```
   GET /api/catalog/descriptors/assembly::t_bta::1.0
   ```

   The response to this query displays the descriptor. A sample extract is shown here. A full assembly descriptor sample can be viewed in the following topic: "Sample assembly descriptor" on page 82

   ```
   name: assembly::t_bta::1.0
   description: Assembly comprised of "components\\t_simple.yml"
   properties:
     data:
       default: "data"
       type: string
       description: 'parameter passed'
     output:
       description: an example output parameter
       type: string
       read-only: true
     deploymentLocation:
       type: string
       description: name of openstack project to deploy network
       default: admin@local
   ...
   ```

   The purpose of the 'properties' section in the API request is to give values to required assembly properties. The 'properties' section in the API request must set the value of any properties from the 'properties' section of the assembly descriptor that don't have a default value. You can override any default values. In the following example steps the default value of 'deploymentLocation' is changed.

2. Initiate a **createAssembly** event from the Daytona API. Use the swagger-url for the Daytona service to create a new assembly instance using the following POST command:

   ```
   POST /api/intent/createAssembly
   {
     "assemblyName": "test_1",
     "descriptorName": "assembly::t_bta::1.0",
     "intendedState": "Inactive",
       properties":{
   ```

```
"deploymentLocation":"admin@local"}
}
```

In this case, the `test_1` assembly instance (`assemblyName`) does not exist, and so IBM Agile Lifecycle Manager will attempt to install, configure and then start this new instance.

**Note:** Use the 'properties' section to define any assembly properties that are as yet undefined, or to override any already defined default values. In this example a value of `admin@local` is set for the deploymentLocation property.

3. If `test_1` has been successfully created, IBM Agile Lifecycle Manager will return a `Response Code 201`, as in this example:

```
Response code 201

{
  "location": "http://10.220.217.161:8280/api/processes/
5a65ce87-4637-401e-868b-e20ec254fd35",
  "date": "Mon, 11 Sep 2017 11:54:20 GMT",
  "server": "ALM Ishtar/1.1.0-SNAPSHOT",
  "transfer-encoding": "chunked",
  "x-application-context": "ishtar:prod,swagger:8280",
  "content-type": null
}
```

The process identifier in this response is `5a65ce87-4637-401e-868b-e20ec254fd35`. As soon as the new assembly instance is created, it can be referred to by name. that is, `test_1`.

4. To check progress of this request, copy the process identifier from the response into the **id** parameter of the following GET command (`GET /api/topology/assemblies{id}`):

```
GET /api/topology/assemblies/5a65ce87-4637-401e-868b-e20ec254fd35
```

The `processState` in the following sample response indicates that the request has completed, while `intendedState` indicates that it is as yet inactive.

```
{
  "processId": "5a65ce87-4637-401e-868b-e20ec254fd35",
  "assemblyId": " ef4ea879-e313-4e3b-ad11-6a0a7e7544eb",
  "assemblyName": "test_1",
  "assemblyDescriptorName": "assembly::t_bta::1.0",
  "intentType": "CreateAssembly",
  "intent": {
    "assemblyName": "test_1",
    "descriptorName": "assembly::t_bta::1.0",
    "intendedState": "Inactive"
  },
  "processState": "Completed",
  "processStartedAt": "2017-09-14T07:34:55.569Z",
  "processFinishedAt": "2017-09-14T07:34:58.806Z"
}
```

**Note:** If the request had not completed it would have been in the `InProgress` state. If there had been an issue, it would have been in the `Failed` state, in which case there would have been a `requestStateReason` property with text describing the failure.

5. Initiate a **Start** event from the Daytona API to start the new assembly instance, that is, move it from an inactive to an active state.

To start the new assembly instance, use the swagger-url for the Daytona service, using the following POST command:

```
POST /api/intent/changeAssemblyState
{
  "assemblyName": "test_1",
  "intendedState": "Active"
}
```

IBM Agile Lifecycle Manager executes Configure and then Start transitions, which move the state to 'active'.

6. To verify the status of the assembly instance, check it again:

```
GET /api/topology/assemblies/5a65ce87-4637-401e-868b-e20ec254fd35
```

The 'intendedState' in the sample response indicates that the Start event has completed successfully.

```
{
  "processId": "5a65ce87-4637-401e-868b-e20ec254fd35",
  "assemblyId": " ef4ea879-e313-4e3b-ad11-6a0a7e7544eb",
  "assemblyName": "test_1",
  "assemblyDescriptorName": "assembly::t_bta::1.0",
  "intentType": "CreateAssembly",
  "intent": {
    "assemblyName": "test_1",
    "descriptorName": "assembly::t_bta::1.0",
    "intendedState": "Active",
...
```

**Results**
A new assembly instance with the name 'test_1' exists in an 'Active' state, and configured according to the rules defined in assembly descriptor 'assembly::t_bta::1.0'.

## Exploring an assembly instance

This topic describes how to use IBM Agile Lifecycle Manager to see the assembly structure, and (optionally) the associated history of lifecycle transitions of a previously created assembly, in this case test_1.

**Before you begin**
You must create the example assembly instance called `test_1`, which is described in "Creating an assembly instance" on page 64.

**About this task**

To see the assembly structure and the history of lifecycle transitions, you explore the topology of the assembly instance by using the API for Galileo (the gateway service).

**Procedure**

1. Identify the name of the assembly instance from its `assemblyName` field, in this case `test_1`.
2. Query the assembly topology through the Galileo API using the following GET command:

   ```
   GET /api/topology/assemblies
   ```

   To see only the structure of the assembly, set the value of `numEvents` to zero (`0`). If you set it to a value greater than zero, the specified number of Assembly Lifecycle Request events that have occurred on the assembly and their associated State Change Events is displayed, starting with the latest.

   The following example depicts the GET request to obtain the topology of test_1, with no event history.

   ```
   GET /api/topology/assemblies?numEvents=0&name=test_1
   ```

   This query results in the following response:

   ```
   {
     "type": "Assembly",
     "id": "ef4ea879-e313-4e3b-ad11-6a0a7e7544eb",
     "name": "test_1",
     "state": "Active",
     "descriptorName": "assembly::t_bta::1.0",
     "properties": [
       {
         "name": "numOfServers",
         "value": "1"
       },
       {
   ```

```
        "name": "data",
        "value": "data"
      },
      {
        "name": "deploymentLocation",
        "value": "admin@local"
      },
      {
        "name": "resourceManager",
        "value": "test-rm"
      }
    ],
    "createdAt": "2017-09-12T06:38:43.365+0000",
    "lastModifiedAt": "2017-09-12T06:38:45.514+0000",
    "children": [
      {
        "type": "Component",
        "id": "c903c6db-9a79-4248-b62b-fd11ec01efe0",
        "name": "test_1__A",
        "externalId": "72857624-ccb4-4cf2-8ebf-0ab05e67a5a5",
        "state": "Active",
        "descriptorName": "resource::t_simple::1.0",
        "properties": [
          {
            "name": "data",
            "value": "data"
...
    ]
}
```

This sample response displays the identity, status and structure of the assembly instance. The assembly instance has two child instances, `test_1__A` and `test_1__B__1`. The names are generated from the assembly name `test_1` and the resource name from the `t_bta` descriptor. Resource B is in a scaling group, and each member of a scaling group is numbered. The relationship between the two resources can also be seen, as well as the references to resources used from the `deploymentLocation` property.

## Healing a component

A heal request targets a component (a resource that is 'broken') and attempts to return it to an 'Active' state.

### Before you begin

- You must create the example assembly instance with an assemblyName of `test_1`, as described in "Creating an assembly instance" on page 64.
- To heal the component of an assembly, the assembly to which the component belongs must be in an Active state.

### About this task

- IBM Agile Lifecycle Manager accepts the request to heal without performing any checks first.
- Heal is a pattern that calls 'Stop', 'Start', and then 'Integrity' on the component.
- If 'Integrity' is successful, then the heal is successful.
- The assembly containing the broken component must be in the 'active' state to call heal.
- The request to heal includes the ID of the component in the assembly instance to be healed.

### Procedure

1. To obtain the ID of the component in the assembly instance to be healed, you can query the assembly topology through the Galileo API using the following GET command:

   **Note:** You identify the component to be healed by the following combination: The name or ID of the assembly, **plus** the name or ID of the component.

   ```
   GET /api/topology/assemblies?numEvents=0&name=test_1
   ```

This query will return assembly topology information including the `id` in the `children` section, as depicted in the following sample extract:

```
…
  "children": [
    {
      "type": "Component",
      "id": "c903c6db-9a79-4248-b62b-fd11ec01efe0",
      "name": "test_1__A",
      …
```

2. To initiate a heal pattern from the Daytona API, run the following POST command from the swagger-url of the Daytona service. You can use the name or ID as identifier.

   In the following examples, the ID or name obtained in the previous step can be used to define the `componentId` value, which targets the component to be healed. Any of the following examples will initiate a heal pattern.

   **Assembly name and component name**

   ```
   POST /api/intent/healAssembly
   {
     "assemblyName": "test_1",
     "brokenComponentName": " test_1__A"
   }
   ```

   **Assembly name and component ID**

   ```
   POST /api/intent/healAssembly
   {
     "assemblyName": "test_1",
     "brokenComponentId": "c903c6db-9a79-4248-b62b-fd11ec01efe0"
   }
   ```

   **Assembly ID and component name**

   ```
   POST /api/intent/healAssembly
   {
     "assemblyId": "ef4ea879-e313-4e3b-ad11-6a0a7e7544eb",
     "brokenComponentName": "test_1__A"
   }
   ```

   **Assembly ID and component ID**

   ```
   POST /api/intent/healAssembly
   {
     "assemblyId": "ef4ea879-e313-4e3b-ad11-6a0a7e7544eb",
     "brokenComponentId": "c903c6db-9a79-4248-b62b-fd11ec01efe0"
   }
   ```

   IBM Agile Lifecycle Manager initiates the heal pattern, which cycles through 'Stop', 'Start', and 'Integrity'.

3. To check the status of the previously 'broken' component after healing, you query the assembly topology through the Galileo API using the following GET command, with numEvents set to 5 in order to see the sequence of heal events that occurred.

   ```
   GET /api/topology/assemblies?numEvents=5&name=test_1
   ```

**Results**

If the healing has been successful, the depicted state transitions will move from 'Broken' to 'Inactive' to 'Active'. A state of 'Active' connotes a healthy, in this case healed, component, as depicted in the following example.

**Example**

```
{
  "eventId": "d557dfcb-609b-40d3-9f87-c9cc8568ccc1",
  "rootAssemblyInstanceId": "acbd4cb1-1ec2-41f6-a2c7-693653291e5a",
```

```
  "rootAssemblyInstanceName": "test_1",
  "resourceInstanceId": "a4e6a96a-db96-4ef6-ac59-4e87c7efb5d2",
  "resourceInstanceName": "test_1__A",
  "resourceManager": "test-rm",
  "deploymentLocation": "admin@local",
  "externalId": "6d427f14-34a5-48f9-9575-2ec0b9ede2df",
  "eventCreatedAt": "2017-08-23T11:16:16.637Z",
  "previousState": "Broken",
  "newState": "Inactive",
  "successful": true,
  "changeStartedAt": "2017-08-23T11:16:16.546Z",
  "changeFinishedAt": "2017-08-23T11:16:16.637Z",
  "eventType": "StateChangeEvent"
}

{
  "eventId": "1786a172-7027-4223-943e-edc5673ad5bc",
  "rootAssemblyInstanceId": "acbd4cb1-1ec2-41f6-a2c7-693653291e5a",
  "rootAssemblyInstanceName": "test_1",
  "resourceInstanceId": "a4e6a96a-db96-4ef6-ac59-4e87c7efb5d2",
  "resourceInstanceName": "test_1__A",
  "resourceManager": "test-rm",
  "deploymentLocation": "admin@local",
  "externalId": "6d427f14-34a5-48f9-9575-2ec0b9ede2df",
  "eventCreatedAt": "2017-08-23T11:16:16.867Z",
  "previousState": "Inactive",
  "newState": "Active",
  "successful": true,
  "changeStartedAt": "2017-08-23T11:16:16.678Z",
  "changeFinishedAt": "2017-08-23T11:16:16.867Z",
  "eventType": "StateChangeEvent"
}

{
  "eventId": "70061a12-d42f-42c4-8c70-8ca9391ce37f",
  "eventCreatedAt": "2017-08-23T11:16:16.327Z",
  "assemblyInstanceId": "acbd4cb1-1ec2-41f6-a2c7-693653291e5a",
  "assemblyInstanceName": "test_1",
  "assemblyDescriptorName": "assembly::t_bta::1.0",
  "action": "Heal",
  "requestState": "Completed",
  "requestStartedAt": "2017-08-23T11:16:16.327Z",
  "requestFinishedAt": "2017-08-23T11:16:17.066Z",
  "properties": {
    "data": "data",
    "deploymentLocation": "admin@local",
    "numOfServers": "1",
    "resourceManager": "test-rm"
  },
  "eventType": "AssemblyLifecycleRequest"
}
```

## Scaling a component

Scaling the component of an assembly is a pattern that will either add or remove a component instance from a scaling group (including all relationships).

### Before you begin

- You must create the example assembly instance called `test_1`, which is described in "Creating an assembly instance" on page 64.
- The assembly to which the component belongs must be instantiated in the IBM Agile Lifecycle Manager topology, and be in the 'Active state'.
- A cluster definition for the component itself must exist in the assembly descriptor, in which the minimum and maximum size of the scaling group and the default increment when scaling out or in are also defined.

### About this task

Assembly descriptors are in the IBM Agile Lifecycle Manager catalog, which you can view using the process described in following topic: "Exploring an assembly descriptor" on page 73 For this task you use component B, which is part of the `t_bta` assembly (`assembly::t_bta::1.0`).

The cluster definition for component B is depicted in the following sample:

```
…
composition:
  A:
    type: resource::t_simple::1.0
    quantity: '1'
…
  B:
    type: resource::t_simple::1.0
    cluster:
      initial-quantity: '${numOfServers}'
      minimum-nodes: 1
      maximum-nodes: 4
      scaling-increment: 1
    properties:
 …
```

**Procedure**

1. To identify the name of the component of the `test_1` assembly to be scaled, you can query the assembly topology through the Galileo API using the following GET command:

   ```
   GET /api/topology/assemblies?numEvents=0&name=test_1
   ```

   This query will return assembly topology information including the descriptorName, as depicted in the following sample:

   ```
   "type": "Assembly"
     "id": "bf649336-c8c5-49d9-9f4e-60567fe54135",
     "name": "test_1",
     "state": "Active",
     "descriptorName": "assembly::t_bta::1.0",
     "properties": [
       {
   …
   ```

2. Retrieve the descriptor from the IBM Agile Lifecycle Manager catalog by running the following query on the Apollo API using the descriptorName obtained in the previous step:

   ```
   GET /api/catalog/descriptors/assembly::t_bta::1.0
   ```

   The scaling group definition for B is shown in the following sample.

   ```
   …
   composition:
     A:
       type: resource::t_simple::1.0
       quantity: '1'
   …
     B:
       type: resource::t_simple::1.0
       cluster:
         initial-quantity: '${numOfServers}'
         minimum-nodes: 1
         maximum-nodes: 4
         scaling-increment: 1
       properties:
    …
   ```

   Here one instance of B is created when an instance of the assembly is created. By scaling Out or In, the amount of B instances can be changed between one and four. As the increment is defined as one, each scale out or scale in pattern will increase or decrease the amount of B instances by one.

3. To scaleOut, that is to add another instance of resource B to test_1, use the swagger-url for the Daytona service, using the following POST command:

   You can use either the assemblyName or assemblyId to initiate the 'Scale Out' pattern.

   ```
   POST /api/intent/scaleOutAssembly
   {
     "assemblyName": "test_1",
   ```

```
  "clusterName": "B"
}
```

Or:

```
POST /api/intent/scaleOutAssembly
{
  "assemblyId": "ef4ea879-e313-4e3b-ad11-6a0a7e7544eb",
  "clusterName": "B"
}
```

The clusterName identifies the group of resources to be scaled (B), increasing the amount of B instances by one unless the maximum scaling group size (in this case four) has been reached.

4. To scaleIn, that is to reduce the instances of resource B, use the following POST command:

   You can use either the assemblyName or assemblyId to initiate the 'Scale In' pattern.

   ```
   POST /api/intent/scaleInAssembly
   {
     "assemblyName": "test_1",
     "clusterName": "B"
   }
   ```

   Or:

   ```
   POST /api/intent/scaleInAssembly
   {
     "assemblyId": "ef4ea879-e313-4e3b-ad11-6a0a7e7544eb",
     "clusterName": "B"
   }
   ```

   Here the group of B resources are decreased by one, unless the minimum scaling group size (in this case one) has been reached.

5. To check the status of the assembly after scaling, query the assembly topology using the following GET command, with numEvents set to 5 in order to see the sequence of scaling events.

   ```
   GET /api/topology/assemblies?numEvents=5&name=test_1
   ```

   The assembly topology will depict all the instances of component B and the relationships towards the new resource instances.

### Results

When viewing the topology after running ScaleOut, a new instance of B will be depicted called `test_1__B__2`, as well as a new relationship between it and the existing `test_1__A` instance.

## Uninstalling an assembly instance

To uninstall an existing assembly instance from the IBM Agile Lifecycle Manager topology, and the corresponding resources from the applicable resource managers, you use the swagger-url for the Daytona service.

### Before you begin

Before you can uninstall an assembly instance, you must identify it.

For this example uninstall scenario, you must first create the example assembly instance called `test_1`, which is described in "Creating an assembly instance" on page 64.

### About this task

### Procedure

1. Identify the name of the assembly instance from its `assemblyName` field, in this case `test_1`.

2. To uninstall the assembly instance, run the following POST command from the swagger-url of the Daytona service:

   You can use either the assemblyName or assemblyId to uninstall an assembly instance.

   ```
   POST /api/intent/deleteAssembly
   {
   "assemblyName":"test_1"
   }
   ```

   Or:

   ```
   POST /api/intent/deleteAssembly
   {
   "assemblyId":"ef4ea879-e313-4e3b-ad11-6a0a7e7544eb"
   }
   ```

   A successful uninstall will result in a system response similar to the following example:

   ```
   {  "x-application-context": "ishtar:prod,swagger:8280",  "date": "Mon, 27 Nov 2017 15:54:03
   GMT",
   "location": "http://9.20.65.179:8280/api/swagger/daytona/api/processes/86524eed-532e-47ad-
   aa8a-1d8e9ab0aae3",
   "transfer-encoding": "chunked",  "server": "ALM Ishtar/1.1.2.181",  "content-type": null}
   ```

**Results**
The uninstall process results in the removal of the test_1 assembly instance, as well as any corresponding resources from the applicable resource managers.

### What to do next

You can double-check that the assembly instance has been successfully uninstalled by querying the assembly topology by name, using the following GET command:

```
GET /api/topology/assemblies?numEvents=0&name=test_1
```

If the uninstall process was successful, a response code of `404 (NOT FOUND)` will be returned, indicating that the assembly instance has been removed from IBM Agile Lifecycle Manager.

## Browsing assembly descriptors

This task allows you to browse all descriptors existing in the IBM Agile Lifecycle Manager catalog. The descriptors include both the onboarded resource descriptors and assembly descriptors created in IBM Agile Lifecycle Manager.

### Before you begin

IBM Agile Lifecycle Manager must be installed, with any included resources and test assemblies deployed to the catalog.

**Remember:** Resources must exist (for example, must have been onboarded) before you can browse descriptors.

### Procedure

Query the existing descriptors in the IBM Agile Lifecycle Manager catalog.

The list of resource and assembly descriptors in the IBM Agile Lifecycle Manager catalog can be viewed by running the following query on the Apollo API:

```
GET /api/catalog/descriptors
```

The response to this query lists the names, descriptions and references to all existing descriptors in the catalog, as depicted in the following sample response:

```
[
  {
    "name": "resource::t_simple::1.0",
    "description": "resource for  t_simple",
    "links": [
      {
        "rel": "self",
        "href": "http://10.220.217.175:8280/api/catalog/descriptors/resource::t_simple::1.0"
      }
    ]
  },
…

…
  {
    "name": "assembly::t_bta::1.0",
    "description": "Basic Test Assembly",
    "links": [
      {
        "rel": "self",
        "href": "http://10.220.217.175:8280/api/catalog/descriptors/assembly::t_bta::1.0"
      }
    ]
  }
]
```

**Results**
You now have reference information about the resource and assembly descriptors that are in the IBM Agile Lifecycle Manager catalog.

## Exploring an assembly descriptor

This task allows you to investigate the full contents of a specific resource or assembly descriptor in the IBM Agile Lifecycle Manager catalog.

### Before you begin

To explore an assembly descriptor, it must have been created in IBM Agile Lifecycle Manager, or created in the IBM Agile Lifecycle Manager catalog during the onboarding of a resource descriptor.

### Procedure

Retrieve the descriptor from the IBM Agile Lifecycle Manager catalog by running the following query on the Apollo API:

The assembly descriptor explored in this example is `assembly::example::1.0`

```
GET /api/catalog/descriptors/assembly::t_bta::1.0
```

The response to this query displays the descriptor. A sample extract is shown here. The full assembly descriptor of this assembly can be viewed in the following topic: "Sample assembly descriptor" on page 82

```
name: assembly::t_bta::1.0
description: Basic Test Assembly
properties:
  data:
    default: "data"
    type: string
    description: 'parameter passed'
…
composition:
  A:
    type: resource::t_simple::1.0
…

  B:
    type: resource::t_simple::1.0
…
references:
  internal-network:
```

```
…
relationships:
  third-relationship:
    source-capabilities:
    - A.capability-3
    target-requirements:
    - B.requirement-3
…
```

**Results**
This task allows you to view a specific resource or assembly descriptor in the Browsing assembly descriptors IBM Agile Lifecycle Manager catalog.

## Creating a new assembly descriptor

This task creates a new assembly descriptor and inserts it in the IBM Agile Lifecycle Manager topology.

### Before you begin

IBM Agile Lifecycle Manager must be installed, with all included resources and test assemblies deployed to the catalog.

### About this task

An example assembly named `assembly::example::1.0` is used in this task.

### Procedure

1. Insert a new assembly descriptor into the IBM Agile Lifecycle Manager catalog by running the following request on the Apollo API.

   The content of the assembly descriptor must be in YAML format. The full assembly descriptor of this assembly is in the following topic: "Sample assembly descriptor" on page 82

   ```
   POST /api/catalog/descriptors

   {
   DESCRIPTOR OF THE NEW ASSEMBLY IN YAML FORMAT
   }
   ```
2. Verify that the new descriptor exists in the IBM Agile Lifecycle Manager catalog.

   The list of resource and assembly descriptors in the IBM Agile Lifecycle Manager catalog can be viewed by running the following query on the Apollo API:

   ```
   GET /api/catalog/descriptors
   ```

   The response to this query lists the new descriptor.

**Results**
After completing this task, the newly created assembly descriptor exists in the IBM Agile Lifecycle Manager topology and can be instantiated and managed by IBM Agile Lifecycle Manager.

## Updating an assembly descriptor

This task updates an existing assembly descriptor in the IBM Agile Lifecycle Manager catalog.

### Before you begin

An assembly descriptor must exist in the IBM Agile Lifecycle Manager catalog.

### About this task

This task changes an existing descriptor in the IBM Agile Lifecycle Manager catalog without changing the version number of the descriptor.

**Tip:** This procedure is mainly intended for service designers who want to change a descriptor during the development process.

### Procedure

1. Change the assembly descriptor as required.
2. Replace the previous assembly descriptor version in the IBM Agile Lifecycle Manager catalog by running the following request on the Apollo API.

   The assembly descriptor replaced is this example is `assembly::example::1.0`. The full assembly descriptor of this assembly is in the following topic: "Sample assembly descriptor" on page 82

   The content of the assembly descriptor must be in YAML format.

   ```
   PUT /api/catalog/descriptors/assembly::example::1.0

   {
   CHANGED DESCRIPTOR OF AN EXISTING ASSEMBLY IN YAML FORMAT
   }
   ```
3. Verify that the descriptor updates have been made.

   Retrieve the descriptor from the IBM Agile Lifecycle Manager catalog by running the following query on the Apollo API:

   ```
   GET /api/catalog/descriptors/assembly::example::1.0
   ```

   The response to this query displays the updated descriptor.

### Results

After completing this task the `assembly::example::1.0` descriptor in the IBM Agile Lifecycle Manager catalog is updated according to the YAML given as input in step 2.

## Removing an assembly descriptor

This task removes an existing assembly descriptor from the IBM Agile Lifecycle Manager catalog.

### Before you begin

An assembly descriptor must exist in the IBM Agile Lifecycle Manager catalog before you can remove it.

### About this task

After removing a descriptor from the IBM Agile Lifecycle Manager catalog it is not possible to create new instances of it. All existing instances of the assembly will remain and are not deleted. However, it is recommended that you not remove an assembly descriptor while there are existing instances of it in the IBM Agile Lifecycle Manager topology.

**Tip:** Assembly instances can be deleted by running the "Uninstalling an assembly instance" on page 71 task.

### Procedure

1. Remove an assembly descriptor from the IBM Agile Lifecycle Manager catalog by running the following request on the Apollo API.

   The assembly descriptor deleted in this example is `assembly::example::1.0`

   ```
   DELETE /api/catalog/descriptors/assembly::example::1.0
   ```
2. Verify that the descriptor has been deleted.

Attempt to retrieve the descriptor from the IBM Agile Lifecycle Manager catalog by running the following query on the Apollo API:

```
GET /api/catalog/descriptors
```

The response to this query should not list the descriptor anymore.

**Results**
After completing this task the descriptor has been removed from the IBM Agile Lifecycle Manager catalog and can no longed be viewed or managed by IBM Agile Lifecycle Manager.

## Upgrading an assembly instance

Once an assembly has been created it can be upgraded. An assembly instance can be upgraded to a new type, or have changed property values, or both new type and property values. This topic describes how to upgrade an assembly instance, with examples of how to change types and add a component.

### Before you begin

The assembly to be upgraded should be instantiated in the IBM Agile Lifecycle Manager topology and must be in the 'active' state.

For the following example, you must have created the example assembly instance called `test_1`, as described in "Creating an assembly instance" on page 64.

### About this task

This task upgrades a property of the 'test_1' assembly instance of the type 't_single::1.0' to the type 't_single::1.1', which also has an additional component named 'B'.

**Supported assembly upgrade scenarios**

**Definition of an assembly upgrade**

Change an active assembly instance from its current type and set of properties to a new type and/or new property values.

The 'type' of a component instance is determined by `[assembly|resource]::<type name>::<version>`

If the name of a property in the original and new type are the same, then they are assumed to be the same and can be mapped from the original to new properties.

If a property value is changed in a component, then the component will be re-installed with the new value.

If there is a new relationship between components of the new assembly type, the relationship is created. This may mean that a component must be transitioned to the correct states to create the new relationship.

If a relationship between components is removed from the upgraded assembly, the relationship is deleted.

If a property value of a relationship changes, then the relationship is deleted and re-created. This may mean that a component must be transitioned to the correct states to create the new relationship.

If a component, identified by name and type is not in the new assembly it is uninstalled.

If a component, identified by name and type is not in the original assembly it is created and transitioned to the active state.

If an assembly's properties are changed, only the resources impacted are changed, resources that are not impacted and are unchanged. That is, if after an upgrade, a resource has the same name, type and property values, then it will not be transitioned during the upgrade; it will remain in the active state, unless a transition was triggered by a relationship change.

If a component's descriptor changes in any way, it is expected that the type will have changed (that is, there is a new type name and or version) and the component will be re-installed.

If a reference to an external component is removed from the assembly then any relationships referring to it will be deleted.

If a reference to an external component is added to the assembly then any relationships referring to it will be created.

The size of a cluster before an upgrade is maintained after the upgrade.

The current limitations on an Assembly Upgrade are:

- Changing cluster property values to 'initial-quantity', 'minimum-nodes', 'maximum-nodes' or 'scaling-increment' is not currently supported by the Assembly Upgrade pattern.
- Changes to policy and metric property values is not supported by the Assembly Upgrade pattern.

**Procedure**

1. Obtain a copy of `t_simple::1.0` by browsing the catalog. Run the following query on the Apollo API to find it:

```
GET /api/catalog/descriptors
```

2. Change the version to `t_simple::1.1`
3. Add a new resource 'B' to the composition table by copying resource A and changing it as depicted in the following example:

```
B:
    type: resource::t_simple::1.1
    quantity: '1'
    properties:
      referenced-internal-network:
        value: ${internal-network.id}
      reference-public-network:
        value: ${public-network.id}
      image:
        value: ${xenial-image.id}
      key_name:
        value: "ACCANTO_TEST_KEY"
      data:
        value: ${data}
      output:
        value: "B_output"
      deploymentLocation:
        value: ${deploymentLocation}
      resourceManager:
        value: ${resourceManager}
```

4. Create a new assembly descriptor in the IBM Agile Lifecycle Manager catalog called `assembly::t_single::1.1` by running the following request on the Apollo API.

```
POST /api/catalog/descriptors/assembly::t_single::1.1
```

5. Query the assembly topology for the assembly instance 'test_1' by using the following API request on the Ishtar service.

```
GET /api/topology/assemblies?numEvents=0&name=test_1
```

The assembly type for `test_1` will be `assembly::t_single::1.0`, and there will not be a component of type 'B'.

6. Upgrade of assembly instance.

```
POST /api/intent/upgradeAssembly
{
  "assemblyName": "test_1",
  "descriptorName": "assembly::t_single::1.1"
}
```

7. Verify the updated assembly instance 'test_1' by using the following API request on the Ishtar service.

```
GET /api/topology/assemblies?numEvents=0&name=test_1
```

The assembly type for `test_1` will be `assembly::t_single::1.1`, and there will be a new component of type 'B'.

## List all onboarded resource managers

Use this task to see all resource managers that have been onboarded to IBM Agile Lifecycle Manager.

### Before you begin

**Note:** The Resource Manager API was deprecated with IBM Agile Lifecycle Manager 2.1.0.

IBM Agile Lifecycle Manager must be installed, with all included resources and test assemblies deployed to the catalog.

One or more resource managers must have been onboarded.

**Remember:**

Onboarding is the act of adding a resource manager to IBM Agile Lifecycle Manager. It lets IBM Agile Lifecycle Manager know that the resource manager exists, and it imports the descriptors of all the resource types managed by the resource manager. It also gathers the information about the deployment locations that the resource manager uses.

### About this task

IBM Agile Lifecycle Manager maintains a record of each resource manager it can use to create and manage resources. When a new resource manager record is created, the resource types managed by that resource manager instance are read into IBM Agile Lifecycle Manager using the resource manager's API.

### Procedure

Query the list of onboarded resource managers by running the following query on the Ishtar API:

```
GET /api/resource-managers
```

The response to the query returns the name, type and url of each resource manager present.

### What to do next

You now know which resource managers have been onboarded, and can use their identifiers to access them.

## Exploring an onboarded resource manager

You can explore the status of any onboarded resource managers using the API for Ishtar (Gateway).

### Before you begin

**Note:** The Resource Manager API was deprecated with IBM Agile Lifecycle Manager 2.1.0.

To confirm a resource manager has been onboarded, you need to know the name that was used to create it in IBM Agile Lifecycle Manager, in this case 'test-rm'.

**Remember:**

Onboarding is the act of adding a resource manager to IBM Agile Lifecycle Manager. It lets IBM Agile Lifecycle Manager know that the resource manager exists, and it imports the descriptors of all the resource types managed by the resource manager. It also gathers the information about the deployment locations that the resource manager uses.

### About this task

IBM Agile Lifecycle Manager uses resource managers to deploy new resource instances and execute transitions and operations on them. A resource manager manages instances of resource types in a number of resource locations where resource instances can be created.

IBM Agile Lifecycle Manager maintains a record of each resource manager it can use to create and manage resources. When a new resource manager record is created, the resource types managed by that resource manager instance are read into IBM Agile Lifecycle Manager using the resource manager's API.

In IBM Agile Lifecycle Manager, the resource types are stored in `var_alm/catalog/resources` as descriptor files. The deployment locations that a resource manager instance can deploy resources into are also onboarded. Each deployment location is considered local to a resource manager.

### Procedure

1. Ensure you have the correct resource manager name.

   The name of the resource manager is defined when the resource manager record is created, as described in "Creating a new resource manager record" on page 79.
2. Use the following GET command on the Ishstar API to confirm that a resource manager has been onboarded:

   ```
   GET /api/resource-managers/<your-rm>
   ```

   If the resource manager exists, the following response will be displayed.

   ```
   {
     "name": "<your-rm>",
     "type": "default",
     "url": "http://<your-rm>:8295/api/resource-manager"
   }
   ```

   As *<your-rm>* is the name of a resource manager instance, the response body comprises the instance name and the type of the resource manager (in this case `default`). The type of the resource manager is local to IBM Agile Lifecycle Manager to allow the user to describe resource manager instances that are of the same type.

   The `url` property contains the URL of the resource manager that is used by IBM Agile Lifecycle Manager to make requests to the resource manager instance. The `hostname:port` (*<your-rm>*`:8295`) must be reachable from the IBM Agile Lifecycle Manager host. If using the Swagger API, the `{id}` parameter in the URL is set to the 'name' value (*<your-rm>*).

   **Tip:** If the resource manager has not been onboarded, then the response code to the GET will be `404 - NOT FOUND`

### What to do next

To perform another check of successful onboarding, you can look in `var_alm/catalog/resources` and check that any resource descriptors that the new resource manager supports have been onboarded. To view descriptors, follow the steps described in "Browsing assembly descriptors" on page 72.

## Creating a new resource manager record

You create a new resource manager record in IBM Agile Lifecycle Manager in order to make the system aware of the resource manager, and enable it to access the resources it manages.

### Before you begin

**Note:** The Resource Manager API was deprecated with IBM Agile Lifecycle Manager 2.1.0.

This assumes a resource manager has been installed, configured and is ready to be used by IBM Agile Lifecycle Manager at a given URL, which in this example is `http://`*<rm-ip-address>:<rm-api-port>*`/api/resource-manager`

**Remember:** A resource manager record, which is created in this task, is not the same as an actual resource manager.

IBM Agile Lifecycle Manager maintains a record of each resource manager it can use to create and manage resources. When a new resource manager record is created, the resource types managed by that resource manager instance are read into IBM Agile Lifecycle Manager using the resource manager's API.

### About this task

To create a new record of a resource manager in order to make IBM Agile Lifecycle Manager aware of its existence, you must give it a unique name, in this example 'ucd'.

**Tip:** 'Type' is a name local to IBM Agile Lifecycle Manager and can be used to describe the type of resource manager.

### Procedure

1. To create a new resource manager record, use the following POST command on the Ishstar API:

```
POST /api/resource-managers
{
  "name": "ucd",
  "type": "ucd",
  "url": "http://<rm-ip-address>:<rm-api-port>/api/resource-manager"
}
```

   If the response code to the POST is `201 - Created`, then the record has been created within IBM Agile Lifecycle Manager. If the response code is anything other, then a problem has been encountered.

2. Use the following GET command on the Ishtar API to confirm the status of the new 'ucd' resource manager record.

```
GET /api/resource-managers/ucd
```

   If the record has been created, the following response will be displayed:

```

{
  "name": "ucd",
  "type": "ucd",
  "url": "http://<rm-ip-address>:<rm-api-port>/api/resource-manager"
}
```

   In addition, the response body of the request returns information on the onboarded resource types and possible deployment locations, as depicted in the following example:

```
{
  "resourceManagerOperation": "ADD",
  "deploymentLocations": {
    "test@local2": {
      "operation": "ADD",
      "success": true
    },
    "admin@local": {
      "operation": "ADD",
      "success": true
    },
    "admin@local2": {
      "operation": "ADD",
      "success": true
    }
  },
  "resourceTypes": {
    "resource::<your-rm>::1.0": {
      "operation": "ADD",
…
```

**Results**
A new resource manager record has been created in IBM Agile Lifecycle Manager, which is now aware of the resource manager it references, and is able to access the resources it manages.

**What to do next**

If you attempt to create another record with the same name, a 409 error code will be returned.

You can obtain more information on the newly created resource manager by following the steps in "Exploring an onboarded resource manager" on page 78.

## Updating a resource manager

Once a record for a resource manager instance has been created, you can perform an update to re-onboard the resource types.

**Before you begin**

**Note:** The Resource Manager API was deprecated with IBM Agile Lifecycle Manager 2.1.0.

The resource manager must have been onboarded before you can update it.

**About this task**

**Tip:** If a resource descriptor with the same name already exists, it is not overridden. If a resource descriptor has been updated, but has the same name, it should be manually deleted from the catalog before trying to onboard the new version.

**Procedure**

1. To update an existing resource manager instance, use the following PUT command on the Ishstar API.

   The following example updates the record for '*<your-rm>*' by checking the original onboarding location of the resource manager for new or updated resources. If using the swagger API, the **{id}** parameter in the URL is set to the name value, in this case '*<your-rm>*'.

   ```
   PUT /api/resource-managers/test-rm
   {
     "name": "<your-rm>",
     "type": "<your-rm>",
     "url": "http://<your-rm>:8295/api/resource-manager"
   }
   ```

2. Use the following GET command on the Ishstar API to check the status of the resource manager following the update:

   ```
   GET /api/resource-managers/<your-rm>
   ```

   In addition, the response body of the request returns information on the onboarded resource types and possible deployment locations, as depicted in the following example:

   ```
   {
     "resourceManagerOperation": "ADD",
     "deploymentLocations": {
       "test@local2": {
         "operation": "ADD",
         "success": true
       },
       "admin@local": {
         "operation": "ADD",
         "success": true
       },
       "admin@local2": {
         "operation": "ADD",
         "success": true
       }
     },
     "resourceTypes": {
   ```

```
    "resource::<your-rm>::1.0": {
      "operation": "ADD",
…
```

**Results**
The resources for the updated resource manager are updated in the IBM Agile Lifecycle Manager catalog.

## Deleting a resource manager record

You can delete the record of a resource manager instance within IBM Agile Lifecycle Manager.

### Before you begin

**Note:** The Resource Manager API was deprecated with IBM Agile Lifecycle Manager 2.1.0.

The resource manager must have been onboarded before you can delete it.

### About this task

After you delete the record of a resource manager instance, resources can no longer be created by that resource manager, and any resources already created can no longer be managed by that resource manager.

**Note:** Neither resource managers nor deployment locations can be deleted if there are active resource instances running in the deployment location. Instead, an error message will be returned indicating that there are active resource instances.

**Note:** The deployment locations for the resource manager are deleted. However, the resource descriptors in `var_alm/calalog/resources` are not deleted, and must be managed manually. See the following topic for more information on removing resource descriptors: "Removing an assembly descriptor" on page 75

### Procedure

- To delete a resource manager record, use the following DELETE command on the Ishstar API.

  The following example deletes the record for *<your-rm>*. If using the swagger API, the **`{id}`** parameter in the URL is set to the name value, in this case '*<your-rm>*'.

  ```
  DELETE /api/resource-managers/<your-rm>
  ```

  Successful deletion of *<your-rm>* is indicated by a 204 response code (Resource Manager was deleted).

**What to do next**
You can double-check that the *`<your-rm>`* has been deleted by running the GET command. If the response code to the GET is `404 – NOT FOUND`, then the DELETE action was successful.

## Sample assembly descriptor

This reference topic contains the full assembly descriptor of the example assembly (`assembly::example::1.0`) used in the Getting Started section.

### `assembly::example::1.0` sample descriptor

```
name: assembly::example::1.0
description: Assembly comprised of "components\\t_simple.yml"
properties:
  data:
    default: "data"
    type: string
    description: 'parameter passed'
  output:
    description: an example output parameter
```

```
    type: string
    read-only: true
  deploymentLocation:
    type: string
    description: name of openstack project to deploy network
    default: admin@local
  resourceManager:
    type: string
    description: name of the resource manager
    default:test-rm
composition:
  A:
    type: resource::t_simple::1.0
    quantity: '1'
    properties:
      referenced-internal-network:
        value: ${internal-network.id}
      reference-public-network:
        value: ${public-network.id}
      image:
        value: ${xenial-image.id}
      key_name:
        value: "ACCANTO_TEST_KEY"
      data:
        value: ${data}
      output:
        value: ${output}
      deploymentLocation:
        value: ${deploymentLocation}
      resourceManager:
        value: ${resourceManager}
references:
  internal-network:
    type: resource::openstack_neutron_network::1.0
    properties:
      deploymentLocation:
        value: ${deploymentLocation}
      resourceManager:
        value: ${resourceManager}
      name:
        type: string
        value: VIDEO
  public-network:
    type: resource::openstack_neutron_network::1.0
    properties:
      deploymentLocation:
        value: ${deploymentLocation}
      resourceManager:
        value: ${resourceManager}
      name:
        type: string
        value: public
  xenial-image:
    type: resource::openstack_glance_image::1.0
    properties:
      deploymentLocation:
        value: ${deploymentLocation}
      resourceManager:
        value: ${resourceManager}
      name:
        type: string
        value: xenial
```

# Chapter 6. Administration

Use the following topics to understand administration tasks, such as viewing system logs, or adjusting global timeout limits for resource managers.

## Providing custom configuration for IBM Agile Lifecycle Manager

During installation, the easiest way to get application configuration into the internal configuration repository (Vault) is to use the lm-configurator helm chart. Alternatively, and after the installation is complete, configuration can be added to Vault via the built-in Vault UI.

### Adding application configuration into Vault

**Vault**

Vault is the chosen storage mechanism for any secure configuration required by the microservices. It is an industry standard approach to storing sensitive information and stores all secure configuration in a highly encrypted format.

To access the Vault UI you will need to add the 'vault.lm' ingress hostname to your DNS server or to the hosts file on your client machine. After the hostname has been registered visit `https://vault.lm:32433/`. Note that the port used is the NodePort of the ingress service installed by Helm Foundation.

Once in the Vault UI, it will be necessary to login to Vault using a token. Vault will have been setup with the default token of `c1abbc33-5b54-4bdf-9db4-7cf7807e31d6` unless this has been overridden in the Helm Foundation values.

Vault should have been configured by the `foundation-vault-init` pod to have a backend called 'lm'. This should be present in the `Secrets Engines' list.

Once signed in, select the ‘lm’ Secrets Engine. Within this ‘lm’ Secrets Engine, configuration can be stored for any of the microservices under a secret with a name matching the microservice being configured, e.g. Nimrod. Any configuration stored under the secret ‘application’ will be shared amongst all services.

Configuration is stored as name value pairs, where the name is the dot-separated name of the configuration item.

The key/values in Vault represent individual Spring parameters and their associated values. They will be picked up the appropriate application if they stored under a secret with a name that matches that application.

Click the ‘Edit’ toggle switch to enter edit mode.

Alternatively, any configuration under the secret `application' will be shared by all the applications in SLM.

### Restarting the Vault Service

In order to serve secure config to Vault clients, the Vault server needs to be initialised upon first install, and then unsealed. The unseal process must happen after every restart of the Vault service.

If you are using the Foundation installation of Vault that comes with SLM, then Vault will auto-unseal after a restart.

## Monitoring system health

IBM Agile Lifecycle Manager microservice Docker containers have built-in health monitoring, which you can use to check if a service is still available.

### Performing a health check

See the service REST API endpoints.

### Integrating IBM Agile Lifecycle Manager log files into Kibana

You can configure the Red Hat OpenShift Container Platform cluster to collect IBM Agile Lifecycle Manager logs and to display them in Kibana.

To do this, you need to deploy and configure the Red Hat OpenShift Container Platform cluster logging feature within your cluster. For instructions, refer to the Red Hat OpenShift Container Platform documentation which is available on the following website:

https://docs.openshift.com/

## Authentication

The authentication mechanism in use in IBM Agile Lifecycle Manager is OAuth2 combined with LDAP for user credential verification. Below is an explanation of some of the concepts and how authentication should be used in IBM Agile Lifecycle Manager.

### Client credentials

The resources of IBM Agile Lifecycle Manager are protected by an authentication layer in place on the Gateway, Ishtar. In order to gain access to any of these protected resources, an authorisation process must take place. As a minimum, this would require providing some valid client credentials (client Id and secret) which would normally be deemed as sufficient security for any calling system into IBM Agile Lifecycle Manager. Additionally, it is possible to require further user credentials (username and password), which would be administered on a per-user basis and should exist in the configured LDAP database. The IBM Agile Lifecycle Manager UI (Nimrod) would be protected as such, where the Nimrod service itself provides the client credentials and the end user provides the additional user credentials.

The level of authorisation required is dictated by the Grant Type which must be specified when creating client credentials.

### Bearer token

The OAuth2 mechanism makes use of Bearer tokens for authorisation. Once a client authenticates they are provided with a bearer token which can be used to authorise any subsequent interactions up until this token expires. It may then be possible to refresh this Bearer token (using an additional refresh token) to regain access after expiry of the Bearer token without needing to resupply all user credentials.

### OAuth2 token lifetime configuration

When creating client credentials, there are 2 configurable values with these system defaults that control the lifetime of the OAuth2 tokens:

```
accessTokenValidity: 1200 #20 minutes
refreshTokenValidity: 30600 #8.5 hours
```

The unit for these values is seconds: 20 x 60 = 1200.

With these values, the behaviour will be that after 20 minutes, the system will refresh the Bearer token with a new one. At this point LDAP is checked to verify the user is still active. This happens every 20 minutes for 8.5 hours, after which the user will be forced to re-enter login credentials.

These default values are specified in the Ishtar configuration YAML file:

```
alm:
    ishtar:
        security:
            defaultAccessTokenValidity: 1200 #20 minutes
            defaultRefreshTokenValidity: 30600 #8.5 hours
```

### Grant types

The following grant types are supported in IBM Agile Lifecycle Manager:

*Table 6. IBM Agile Lifecycle Manager grant types*

| Grant type | Description |
|---|---|
| client_credentials | Authorisation requires only a valid **Client Id** and **Secret**. Recommended for calling systems. |
| password | Authorisation requires a valid **Client Id** and **Secret** and additionally valid **User Credentials**. Recommended for a UI providing human interaction. |
| refresh_token | This grant type can be used in addition to the password grant type and indicates that a caller or user can re-authenticate themselves using a Refresh Token without having to re-provide any user credentials. |

### Authorizing

Once client credentials are setup, it is possible to authorise using them. This example demonstrates how to authenticate given a client with a grant type of 'client_credentials'. To acquire a certificate, the client must post to the auth endpoint below:

```
POST http://<ingress-host>:8280/oauth/token
```

The request must contain the client credentials in the header. This is concatenation of the id:secret, Base64 encoded, prefixed with 'Basic' under a header key of 'Authorization' (this is a standard OAuth2 mechanism).

The request body should be `x-www-form-urlencoded` and include this key/value:

```
grant_type=client_credentials
```

This will return a response similar to this:

```
{
    "access_token": "fdf8e754-1abe-42ae-b064-7969b05788ca",
    "token_type": "bearer",
    "expires_in": 1199,
    "scope": "all"
}
```

### User credentials in LDAP

If the client has a grant type of '`password`', then an additional authentication step of verifying user credentials will take place. These credentials are expected to belong in LDAP. An out-of-the box installation of Open LDAP is provided which IBM Agile Lifecycle Manager is configured to point to. IBM Agile Lifecycle Manager can be configured to use an existing LDAP database.

IBM Agile Lifecycle Manager can be configured to use an existing LDAP database. This would involve modifying the following config in the `var_alm\config-repo\ishtar.yml` file:

```
alm:
    ishtar:
        security:
            ldap:
                url: ldap://alm-openldap:389
                base: dc=alm,dc=com
                managerDn: cn=admin,${alm.ishtar.security.ldap.base}
                managerPassword: almadmin
                userSearchBase: ou=people
                userSearchFilter: uid={0}
                passwordAttribute: userPassword
```

**Tip:** IBM Agile Lifecycle Manager provides a mechanism for initially loading users into the OpenLDAP database on initial creation, but beyond this provides no mechanism for managing users. There are many available LDAP clients which can be used for such purposes. One such client is the free Windows software LDAP Admin. See the following site for more information: http://www.ldapadmin.org/

### Secret and Password Encryption

By default, any client secrets stored in IBM Agile Lifecycle Manager will be encoded with the BCrypt algorithm before they are stored in the local database. This is handled seamlessly and has no apparent difference to any calling systems.

Additionally, BCrypt encoding is the default option for LDAP passwords. In this case this is more apparent, as there is an expectation that any passwords stored in LDAP are encoded with BCrypt.

# Chapter 7. Reference

Use the following reference information to enhance your understanding of the IBM Agile Lifecycle Manager APIs and YAML specifications.

**Restriction:** Code samples provided in this section may contain references to test data and other example text not relevant to your own scenario.

## LDAP directory service examples

The examples in this section explain in detail how to set up an LDAP directory service and plug in the same with an IBM Agile Lifecycle Manager deployment.

### Example 1: Configuring IBM Agile Lifecycle Manager to integrate with the external LDAP server using simple authentication

This example explains how to set up an external LDAP directory service with predefined IBM Agile Lifecycle Manager groups and users, and to authenticate them using simple authentication with one of those predefined users.

1. **Setting up the LDAP directory service populated with predefined LDAP groups and users for simple authentication mechanism**

   **Note:** All commands (run as root or use sudo) in the following instructions are for OpenLDAP running on Ubuntu 18.04.

   a. Installing OpenLDAP on Ubuntu 18.04.

      1) Install OpenLDAP:

         On the OpenLDAP server, run the following command:

         ```
         sudo apt install slapd ldap-utils
         ```

         The installation will request the creation of an administrator password for the LDAP directory. Make a note of the password that you chose. In this example "`password`" is used as the password of the LDAP administrator.

      2) Reconfigure your existing OpenLDAP.

         To reconfigure the suffix of your OpenLDAP server, run the following command:

         ```
         sudo dpkg-reconfigure slapd
         ```

         Use the following values when configuring the OpenLDAP Server:

         ```
         Omit OpenLDAP Server Configuration: No
         DNS Domain Name: example.org
         Organization Name: dc=example,dc=org
         Administrator Password: password
         Confirm Password: password
         Database backend to use: MDB
         Do you want database to be removed when sldap is purged? : No
         Move Old Database?: Yes
         ```

         **Note:** All commands in the instructions below use the suffix `dc=example,dc=org`. For the purpose of the example we assume the user specifies `example.org` as the DNS domain and `dc=example,dc=org` as the organization during reconfiguration, and `password` as the password of the LDAP Administrator.

   b. Populating OpenLDAP with a predefined IBM Agile Lifecycle Manager LDAP configuration (consisting of roles, ldapgroups, and users).

**Note:** IBM Agile Lifecycle Manager mandates that all passwords are stored in `bcrypt` format. The `bcrypt` hashing mechanism that is used in IBM Agile Lifecycle Manager currently only supports hashed passwords with the $2a prefix. So ensure that you chose the $2a prefix for each LDAP user password, otherwise the user's login will fail. In the following `ldif` configuration file the `userPassword` value `$2a $10$nyG5R781Vbowtj73XohFNOY6RHoMz6fInUMwtHO0lkib.mh0yo4T6` is "password" when encrypted using `bcrypt`.

1) Create a file named `predefined-user.ldif`.

   On the OpenLDAP server, create a file called `predefined-user.ldif` with the following content, making sure to preserve the new lines between sections:

```
dn: ou=groups,dc=example,dc=org
objectClass: organizationalUnit
objectClass: top
ou: groups

dn: ou=people,dc=example,dc=org
changetype: add
objectClass: organizationalUnit
objectClass: top
ou: people

dn: uid=almuser,ou=people,dc=example,dc=org
changetype: add
objectClass: person
objectClass: uidObject
cn: almuser
sn: almuser
uid: almuser
userPassword: $2a$10$nyG5R781Vbowtj73XohFNOY6RHoMz6fInUMwtHO0lkib.mh0yo4T6

dn: uid=almoperator,ou=people,dc=example,dc=org
changetype: add
objectClass: person
objectClass: uidObject
cn: almoperator
sn: almoperator
uid: almoperator
userPassword: $2a$10$nyG5R781Vbowtj73XohFNOY6RHoMz6fInUMwtHO0lkib.mh0yo4T6

dn: uid=almoeditor,ou=people,dc=example,dc=org
changetype: add
objectClass: person
objectClass: uidObject
cn: almoeditor
sn: almoeditor
uid: almoeditor
userPassword: $2a$10$nyG5R781Vbowtj73XohFNOY6RHoMz6fInUMwtHO0lkib.mh0yo4T6

dn: uid=almadmin,ou=people,dc=example,dc=org
changetype: add
objectClass: person
objectClass: uidObject
cn: almadmin
sn: almadmin
uid: almadmin
userPassword: $2a$10$nyG5R781Vbowtj73XohFNOY6RHoMz6fInUMwtHO0lkib.mh0yo4T6

dn: uid=secadmin,ou=people,dc=example,dc=org
changetype: add
objectClass: person
objectClass: uidObject
cn: secadmin
sn: secadmin
uid: secadmin
userPassword: $2a$10$nyG5R781Vbowtj73XohFNOY6RHoMz6fInUMwtHO0lkib.mh0yo4T6

dn: uid=readonly,ou=people,dc=example,dc=org
changetype: add
objectClass: person
objectClass: uidObject
cn: readonly
sn: readonly
uid: readonly
userPassword: $2a$10$nyG5R781Vbowtj73XohFNOY6RHoMz6fInUMwtHO0lkib.mh0yo4T6
```

```
dn: cn=SLMAdmin,ou=groups,dc=example,dc=org
objectclass: groupOfNames
cn: SLMAdmin
member: uid=almadmin,ou=people,dc=example,dc=org

dn: cn=Portal,ou=groups,dc=example,dc=org
objectclass: groupOfNames
cn: Portal
member: uid=almuser,ou=people,dc=example,dc=org
member: uid= almoperator,ou=people,dc=example,dc=org
member: uid= almoeditor,ou=people,dc=example,dc=org

dn: cn=RootSecAdmin,ou=groups,dc=example,dc=org
objectclass: groupOfNames
cn: RootSecAdmin
member: uid=secadmin,ou=people,dc=example,dc=org

dn: cn=ReadOnly,ou=groups,dc=example,dc=org
objectclass: groupOfNames
cn: ReadOnly
member: uid=readonly,ou=people,dc=example,dc=org
```

2) Add the above data to the LDAP server.

   On the OpenLDAP server, add the configuration created in the previous step.

```
ldapadd -H ldap://localhost:389 -D "cn=admin,dc=example,dc=org" -w password -f
predefined-user.ldif
```

2. **Creating an LDAP secret**

   On the cluster in the IBM Agile Lifecycle Manager namespace, create a secret that will be used to connect to the LDAP server. The secret should contain the data that you provided while setting up the OpenLDAP (for example LDAP_URL as `ldap://serverIP:port` and LDAP_BASE_DN as `dc=example,dc=org` ).

   The following code shows the secret content for the `Simple Authentication Mechanism`:

```
oc create secret generic ldap-simple-auth \
--from-literal=LDAP_BASE_DN=dc=example,dc=org \
--from-literal=LDAP_BIND_DN=cn=admin,dc=example,dc=org \
--from-literal=LDAP_ADMIN_PASSWORD=password \
--from-literal=LDAP_PASSWORD_ATTR=userPassword \
--from-literal=LDAP_PASSWORD_ENC=BCRYPT \
--from-literal=LDAP_URL=ldap://<ldapHostMachineIp>:389 \
--from-literal=LDAP_USER_SEARCH_BASE=ou=people \
--from-literal=LDAP_USER_SEARCH_FILTER=uid={0} \
--from-literal=LDAP_GROUP_SEARCH_BASE=ou=groups \
--from-literal=LDAP_GROUP_SEARCH_FILTER=member={0}
```

   When created, the following output message should appear: `secret/ldap-simple-auth created`

   **Note:** If there was already a secret for LDAP, you must delete it and create a new secret with a different name. Otherwise, the upgrade will not take effect.

3. **Installing or updating IBM Agile Lifecycle Manager using the newly created LDAP secret, and authenticating**

   a. On the cluster, update IBM Agile Lifecycle Manager to pick up the secret.

```
helm upgrade <alm_release> --set global.security.ldapConfig.secretName=ldap-simple-auth --
install <chart_repo_name>/ibm-alm-prod --reuse-values --tls  --tiller-namespace
<cs_namespace>
```

   Where the parameter `<alm_release>` is the name chosen for the release and `<chart_repo_name>` is the repository chosen to load the archive during the IBM Agile Lifecycle Manager installation procedure.

   b. Wait for vault and ishtar pods to restart. You can use the following command to monitor them:

```
watch oc get pods -l \'app in \(vault,ishtar\)\'
```

c. When the pods have restarted, verify that you can log in to the IBM Agile Lifecycle Manager User Interface using the user `almadmin` and password `password`.

## Example 2: Configuring IBM Agile Lifecycle Manager to integrate with the external LDAP server using `ldapbind` authentication

This example explains how to set up an external LDAP directory service with predefined IBM Agile Lifecycle Manager groups and users, and to authenticate using `ldapbind` authentication with one of the predefined users.

The following example assumes that an OpenLDAP server has been set up as described in Example 1.

1. **Modifying predefined IBM Agile Lifecycle Manager LDAP groups and users for the ldapbind mechanism**

   a. On the OpenLDAP server, create a file called `predefined-bind-user.ldif` with the following content, making sure to preserve the new lines between sections:

```
dn: uid=almuser,ou=people,dc=example,dc=org
changetype: modify
add: objectClass
objectClass: simpleSecurityObject
-
replace: userPassword
userPassword: password

dn: uid=almoperator,ou=people,dc=example,dc=org
changetype: modify
add: objectClass
objectClass: simpleSecurityObject
-
replace: userPassword
userPassword: password

dn: uid=almoeditor,ou=people,dc=example,dc=org
changetype: modify
add: objectClass
objectClass: simpleSecurityObject
-
replace: userPassword
userPassword: password

dn: uid=almadmin,ou=people,dc=example,dc=org
changetype: modify
add: objectClass
objectClass: simpleSecurityObject
-
replace: userPassword
userPassword: password

dn: uid=secadmin,ou=people,dc=example,dc=org
changetype: modify
add: objectClass
objectClass: simpleSecurityObject
-
replace: userPassword
userPassword: password

dn: uid=readonly,ou=people,dc=example,dc=org
changetype: modify
add: objectClass
objectClass: simpleSecurityObject
-
replace: userPassword
userPassword: password
```

   b. Apply the changes on the OpenLDAP server.

   On the OpenLDAP server, run the following command:

```
ldapmodify -H ldap://localhost:389 -D "cn=admin,dc=example,dc=org" -w password -f
predefined-bind-user.ldif
```

2. **Creating an LDAP secret**

On the cluster in the IBM Agile Lifecycle Manager namespace, create a secret that will be used to connect to the LDAP server. The secret should contain the data that you provided while setting up the server (for example LDAP_URL as `ldap://serverIP:port` and LDAP_BASE_DN as `dc=example,dc=org`)

The following code shows the secret content for the `Ldapbind Authentication Mechanism`:

```
oc create secret generic ldapbind-secret \
--from-literal=LDAP_BASE_DN=dc=example,dc=org \
--from-literal=LDAP_BIND_DN=cn=admin,dc=example,dc=org \
--from-literal=LDAP_ADMIN_PASSWORD=password \
--from-literal=LDAP_PASSWORD_ATTR=userPassword \
--from-literal=LDAP_PASSWORD_ENC=PLAIN \
--from-literal=LDAP_URL=ldap://<ldapHostMachineIp>:389 \
--from-literal=LDAP_USER_SEARCH_BASE=ou=people \
--from-literal=LDAP_USER_SEARCH_FILTER=uid={0} \
--from-literal=LDAP_GROUP_SEARCH_BASE=ou=groups \
--from-literal=LDAP_GROUP_SEARCH_FILTER=member={0} \
--from-literal=AUTH_PROVIDER=ldapBind
```

When created, the following output message should appear: `secret/ldapbind-secret created`

**Note:** If there was already a secret for LDAP, you must delete it and create a new secret with a different name. Otherwise, the upgrade will not take effect.

3. **Installing or updating IBM Agile Lifecycle Manager using the newly created LDAP secret**

   a. On the cluster, update IBM Agile Lifecycle Manager to pick up the secret.

   ```
   helm upgrade <alm_release> --set global.security.ldapConfig.secretName=ldapbind-secret --
   install <chart_repo_name>/ibm-alm-prod  --reuse-values --tls  --tiller-namespace
   <cs_namespace>
   ```

   Where the parameter `<alm_release>` is the name chosen for the release and `<chart_repo_name>` is the repository chosen to load the archive during the IBM Agile Lifecycle Manager installation procedure.

   b. Wait for vault and ishtar pods to restart. You can use the following command to monitor them:

   ```
   watch oc get pods -l \'app in \(vault,ishtar\)\'
   ```

   c. When the pods have restarted, verify that you can log in to the IBM Agile Lifecycle Manager User Interface using the user `almadmin` and password `password`.

## Example 3: Using a custom LDAP user to authenticate to IBM Agile Lifecycle Manager using a predefined group and role

This example explains how to add a new user to one of the IBM Agile Lifecycle Manager predefined groups and to authenticate.

The following example assumes that an OpenLDAP server has been set up as described in Example 1.

1. Create a new user (for example `newuserpartofSLMADMIN` and add it to the predefined group `SLMAdmin`).

   On the OpenLDAP server, create a file `newuserpartofSLMADMIN.ldif`, making sure to preserve the new lines between sections:

   ```
   dn: uid=newuserpartofSLMADMIN,ou=people,dc=example,dc=org
   changetype: add
   objectClass: person
   objectClass: uidObject
   cn: newuserpartofSLMADMIN
   sn: newuserpartofSLMADMIN
   uid: newuserpartofSLMADMIN
   userPassword: $2a$10$nyG5R781Vbowtj73XohFNOY6RHoMz6fInUMwtHO0lkib.mh0yo4T6

   dn: cn=SLMAdmin,ou=groups,dc=example,dc=org
   changetype: modify
   ```

```
add:member
member: uid=newuserpartofSLMADMIN,ou=people,dc=example,dc=org
```

Apply the changes on the OpenLDAP server:

```
ldapmodify -H ldap://localhost:389 -x -D "cn=admin,dc=example,dc=org" -w password -f
newuserpartofSLMADMIN.ldif
```

2. Check authentication.

   On the OpenLDAP server, run the following command which should respond with a prompt to "Enter LDAP Password":

```
ldapsearch -x -H ldap://localhost:389 -b "dc=example,dc=org" -D
"uid=newuserpartofSLMADMIN,ou=people,dc=example,dc=org" -W
```

   When prompted for the LDAP password, enter the `userPassword` value for user `newuserpartofSLMADMIN` as configured in the `newuserpartofSLMADMIN.ldif` file, namely `$2a$10$nyG5R781Vbowtj73XohFNOY6RHoMz6fInUMwtHO0lkib.mh0yo4T6`

   On the IBM Agile Lifecycle Manager UI, login with the username `newuserpartofSLMADMIN` and password `password`.

## Example 4: Creating a custom LDAP group and associating it to a predefined role

This example explains how to create a new ldapgroup in Openldap and associate it to a pre-defined role in IBM Agile Lifecycle Manager.

The following example assumes that an `OpenLDAP` server has been set up as described in Example 1.

1. Create a new group *cn=Designers,ou=groups,dc=example,dc=org* and add a member *uid=almdesigner,ou=people,dc=example,dc=org* to it.

   On the OpenLDAP server, create a file called `new-user-group.ldif` with the following content, making sure to preserve the new lines between sections:

```
dn: uid=almdesigner,ou=people,dc=example,dc=org
changetype: add
objectClass: person
objectClass: uidObject
cn: almdesigner
sn: almdesigner
uid: almdesigner
userPassword: $2a$10$nyG5R781Vbowtj73XohFNOY6RHoMz6fInUMwtHO0lkib.mh0yo4T6

dn: cn=Designers,ou=groups,dc=example,dc=org
objectClass: groupOfNames
cn: Designers
member: uid=almdesigner,ou=people,dc=example,dc=org
```

2. Add the group and user definition to OpenLDAP by running the following command:

```
ldapadd -H ldap://localhost:389 -D "cn=admin,dc=example,dc=org" -w password -f new-user-
group.ldif
```

3. Create a `ConfigMap` that associates the `Designers` group to the `Portal` role:

   On the cluster, create a YAML file called `alm-role-group.yaml`:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: alm-role-group
  namespace: default
data:
  alm-role-group.json: |-
    {
       "alm" : {
          "roles" : {
           "Portal" : {
              "ldapGroups" : [
                 "Designers"
```

```
                ]
            }
          }
      }
    }
```

4. Create the `ConfigMap` by running the following command:

```
oc create -f alm-role-group.yaml
```

5. Update IBM Agile Lifecycle Manager to pick up the `ConfigMap` by running the following command:

```
helm upgrade <alm_release> --set global.security.almRoles.configMapName=alm-role-group --
install <chart_repo_name>/ibm-alm-prod  --reuse-values --tls --tiller-namespace
<cs_namespace>
```

   Where the parameter `<alm_release>` is the name chosen for the release and `<chart_repo_name>` is the repository chosen to load the archive during the IBM Agile Lifecycle Manager installation procedure.

6. Wait for vault and ishtar pods to restart. You can use the following command to monitor them:

```
watch oc get pods -l \'app in \(vault,ishtar\)\'
```

7. When the pods are restarted, verify that you can log in to the IBM Agile Lifecycle Manager User Interface using the user *almdesigner* and password *password*.

## Example 5: Create a new IBM Agile Lifecycle Manager role and associate it to an OpenLDAP group

This example explains how to create a new IBM Agile Lifecycle Manager role and associate it to an OpenLDAP group.

This example assumes that an OpenLDAP server has been set up as described in Example 1 and that a group *cn=Designers,ou=groups,dc=example,dc=org* and a member *uid=almdesigner,ou=people,dc=example,dc=org* have been defined in OpenLDAP. See Example 4 for the steps on how to create the group *cn=Designers,ou=groups,dc=example,dc=org* and the member *uid=almdesigner,ou=people,dc=example,dc=org*.

1. Create a file called `alm-roles.yaml` containing a `ConfigMap` with the definition of the new role `Designer`, making sure to preserve the new lines between sections:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: alm-roles
  namespace: default
data:
  alm-roles.json: |-
    {
       "alm" : {
          "roles" : {
           "Designer" : {
             "privileges" : {
                 "DeployLocMgt" : "read,write,execute",
                 "VnfInstsMgt" : "read,write,execute",
                 "VnfDesMgt" : "read,write,execute",
                 "BehvrScenDes" : "read,write",
                 "NsinstsMgt" : "read,write,execute",
                 "IntentReqsOps" : "read",
                 "nsDesMgt" : "read,write,execute",
                 "IntentReqslMgt" : "read",
                 "VduMgt" : "read,write,execute",
                 "BehvrScenExec" : "read,write,execute",
                 "VduGrpMgt" : "read,write,execute",
                 "VduDesMgt" : "read,write,execute",
                 "MaintModeOride" : "read,execute",
                 "VduInstsMgt" : "read,write,execute",
                 "RmDrvr" : "read,write",
                 "ResourcePkg" : "write"
              },
              "ldapGroups" : [
                    "Designers"
                ]
            }
          }
      }
  }
```

2. Create the `ConfigMap` by running the following command:

```
oc create -f alm-roles.yaml
```

3. Update IBM Agile Lifecycle Manager to pick up the `ConfigMap` by running the following command:

```
helm upgrade <alm_release> --set global.security.almRoles.configMapName=alm-roles --install
<chart_repo_name>/ibm-alm-prod  --reuse-values --tls --tiller-namespace <cs_namespace>
```

   Where the parameter `<alm_release>` is the name chosen for the release and `<chart_repo_name>` is the repository chosen to load the archive during the IBM Agile Lifecycle Manager installation procedure.

4. Wait for vault and ishtar pods to restart. You can use the following command to monitor them:

```
watch oc get pods -l \'app in \(vault,ishtar\)\'
```

5. When the pods are restarted, verify that you can log in to the IBM Agile Lifecycle Manager User Interface using the user `almadmin` and password `password`.

## Using the CI/CD Hub

This section describes the IBM Agile Lifecycle Manager CI/CD Hub, its installation and use.

This is an example, for reference, of a continuous delivery implementation.

### CI/CD Hub functionality

The CI/CD Hub (HUB) is a collection of open source tools set up to perform example continuous integration and deployment processes with the Lifecycle Manager (LM).

The HUB provides a way to manage the artefacts used by the LM through a development and production cycle. These include:

**Gogs git server**
A repository to manage the files used by the LM

**Nexus repository**
A repository for managing the built artefacts produced by the process. This includes a docker registry, a python registry, and a raw repository for storing archives, including a package of the LM All-in-one single server instance of the LM.

**Jenkins server**
A platform to perform automated tasks such as running tests, packaging built objects.

**Lifecycle Manager**
An instance of the Lifecycle Manager to be used by the Jenkins server

#### Why the hub?

When working with customers venturing into the NFV market, there is a need to explain how the artefacts of such deployments are developed, tested, released. These processes are well understood in development but are not common within the networking world, where up til now the components of delivery have been physical pieces of equipment and their configurations. With the introduction of NFV the main components are software and therefore the best practices for managing software need to be used. The HUB is a reference implementation of such a set of tools and this document will explain an example process that may be used by those managing the NFV environment.

## Using the hub

To develop a working Network Service the following actors might exist:

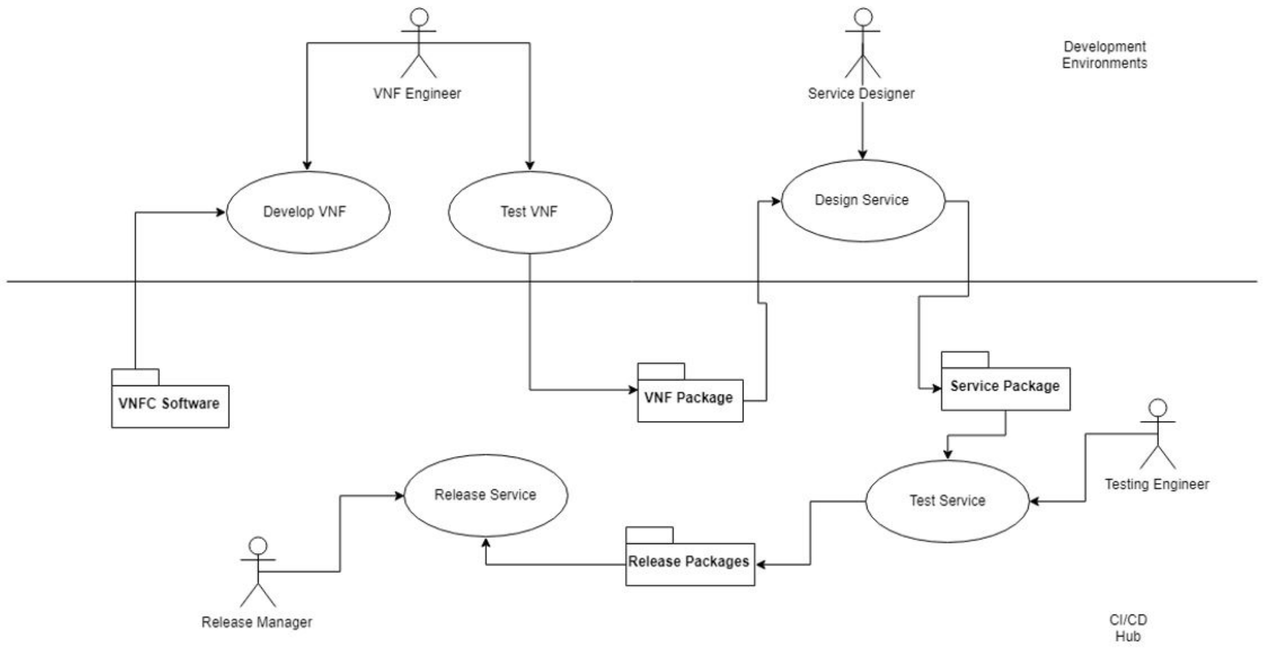
**VNF Engineer**
: The VNG engineer wraps the VNFC software into a working VNF that can then be used in multiple services. They may reuse the VNFC across multiple VNF should the underlying software offer multiple usages.

**Service Designer**
: The service designer takes a number of VNF and designs a working Network Service for a particular situation. In reality a network service is likely to be made up of several linked Network Services that work in concert to provide an Customer solution.

**Test Engineer**
: The test engineer is responsible for testing the Network Service in both Test and Pre-production environments. In reality once the tests have been built by a combination of the Service Designer and the Test Engineer the actual testing will be fully automated by the Hub

**Release Manager**
: Upon successfully passing the testing the Release Manager is responsible for approving the use of the tested Network Service and its components within the production environment. This will include the creation of production release packages that can be referenced and deployed into the production environment.

The following diagram shows a set of typical use cases:



Above the line are the use cases that may utilize the HUB for storage of files and artifacts, but the work is expected to take place on a development environment that is regarded as not part of the HUB. Below the line it is expected that the use cases would be implemented on the HUB and of course the HUB offers both a repository for the file base artifacts of the VNFC, VNF and the NS, but also the packaged versions of those artifacts.

## CI/CD Hub hardware requirements

*Table 7. CI/CD Hub hardware requirements*

| Requirement | Setting |
|---|---|
| Operating system | Ubuntu 16.04 (or later) |

*Table 7. CI/CD Hub hardware requirements (continued)*

| **Requirement** | **Setting** |
|---|---|
| CPU | 8 cores |
| Memory | 32 GB of RAM |
| Disk space | 2 TB |

## Installing the CI/CD Hub

The hub is delivered in a packaged tgz archive.

### Before you begin

To install the HUB you will need a server with at least the following capabilities:

- Linux Server – Ubuntu 16.04 (or higher)
- 32gb RAM (minimum)
- 4+ CPU (minimum) – 8 CPU recommended
- 2TB+ disk.

It is also advised that you have an installation machine that has python and ansible installed on it. This will be used to drive the installation on to the target machine. The Windows Linux subsystem can be set up to perform this task. It must also have sshpass installed.

### Procedure

**On the installation machine:**

1. Copy the installation package to the machine
2. Untar the package in a suitable location.
3. Move to the top-level directory of the package, which this contains a file called getting_started.md that then suggests that you open the file in the `cicdhub` directory called `install_instructions.md`.
4. Follow the instructions in this file.

### What to do next

The following URLs will take you to the various components of the Hub.

**Docker Registry**
http://<YOUR_SERVER>:32736

**Gogs (Git)**
http://<YOUR_SERVER>:8001

**Nexus**
http://<YOUR_SERVER>:8002

**Jenkins**
http://<YOUR_SERVER>:8003

**Openldap**
http://<YOUR_SERVER>:32737

## Using the CI/CD Hub

Once installed the hub comes with some artefacts already installed.

### Procedure

The following process should be followed for each of the four elements.

1. Creating a Pipeline

   a) Create a Pipeline
   b) Create a multibranch pipeline
   c) Complete the branch
   d) Add Git repository details
2. Triggering a build

**Example**

The following is a sample of the Jenkins file from one of the VNF:

```
pipeline{
    agent any
    stages{
        stage('onboard ip-pbx'){
            when {
                branch 'master'
            }
            steps{
            sh "lmctl env list"
                sh "lmctl project push testing"
            }
        }
        stage('test ip-pbx'){
            when {
                branch 'master'
            }
            steps{
                sh "lmctl project test testing"
            }
        }
        stage('package ip-pbx'){
            when {
                branch 'master'
            }
            steps{
                sh "curl -v -u admin:admin123 --upload-file ./_lmctl/_build/ip-pbx-1.0.tgz
http://$NEXUS_SVC_NODEPORT_SERVICE_HOST:$NEXUS_SVC_NODEPORT_SERVICE_PORT_NEXUS_HTTP/
repository/raw/ip-pbx/"
            }
        }
    }
}
```

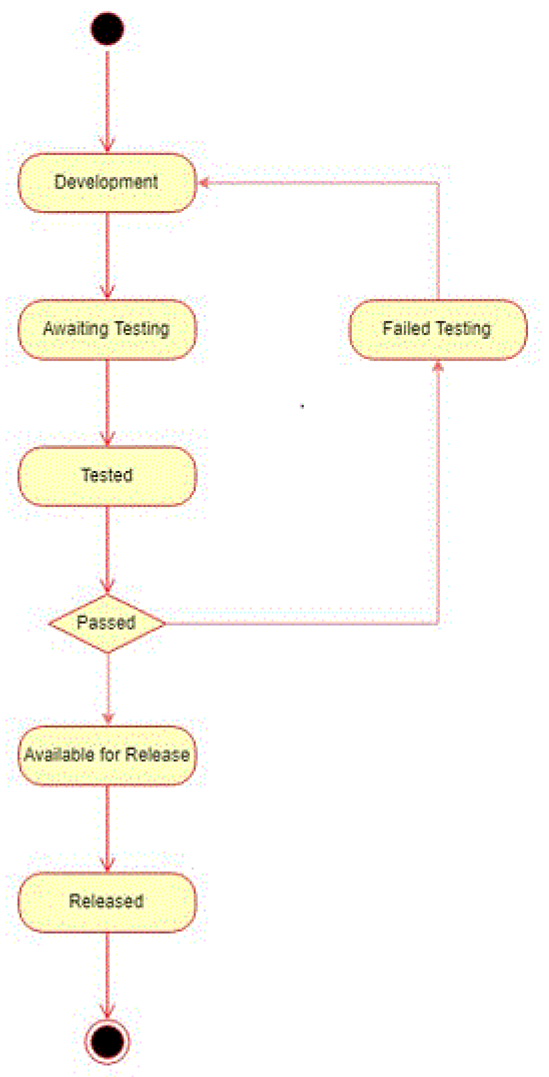
You can see three stages – onboarding, test and package. These are run in order. The stages are marked to be run on the branch ‘master’ – it is quite possible to have other branches for ‘development’ and ‘pre-production’ that might perform different tasks.

The first two stages utilise the lmctl command.

## Managing Artefacts

So far we have looked at the details of the Hub installation and an example set of projects that have been set up to work with Jenkins. We have not looked at how these steps fit into a with Continuous Integration process for managing VNF and Network Services.

The initial stage of the following example process involves the create of a project, this will include setting the project up in Git and possibly creating the pipeline as we have seen above, even though it may not yet be able to pass any tests.



The VNF Engineer would checkout of Git onto their local development machine any files for VNF and their associate VNFC. This will then allow them to create the set of descriptors for the VNFC and VNF and also to create the lifecycle scripts appropriate to the resource manager that they are planning to use. They will have access to an LM to create the VNF and an associated test Network Service. They will also be able to create tests and run these within that environment. Typically any code developed during this development stage would be done on a branch.

Once completed the branch may be merged into the master branch. This may include a code review stage with other team members. Quite often the merge can not be done by the same person who created the code on the branch. This acts as a gate. Once on the master branch the changes are ‘Awaiting Testing’ This is an optional state as Jenkins can be set up to automatically Run the master branch pipeline. (see `https://medium.com/@salohyprivat/getting-gogs-and-jenkins-working-together-5e0f21377bcd` for details of how this can be done)

As we have seen the pipelines can be set up to publish the built package into the Nexus repository. These packages can then be used by other projects to test them. Pipeline for Network Services are likely to use several of these packages representing any VNF or NS that they are dependent upon. E.g:

```
sh "wget http://
$NEXUS_SVC_NODEPORT_SERVICE_HOST:$NEXUS_SVC_NODEPORT_SERVICE_PORT_NEXUS_
HTTP/repository/raw/docker-network/docker-network-1.0.tgz"
sh "lmctl pkg push docker-network-1.0.tgz testing"
sh "wget http://
$NEXUS_SVC_NODEPORT_SERVICE_HOST:$NEXUS_SVC_NODEPORT_SERVICE_PORT_NEXUS_
HTTP/repository/raw/ip-pbx/ip-pbx-1.0.tgz"
sh "lmctl pkg push ip-pbx-1.0.tgz testing"
sh "wget http://
$NEXUS_SVC_NODEPORT_SERVICE_HOST:$NEXUS_SVC_NODEPORT_SERVICE_PORT_NEXUS_
HTTP/repository/raw/sip-performance/sip-performance-1.0.tgz"
sh "lmctl pkg push sip-performance-1.0.tgz testing"
```

Note the use of the environment variables to point at the nexus server and port. In reality there may be two testing phase for an artefact one to do simple testing and one todo further testing in an environment similar to the final deployment.

The Available for Release is an optional state that allows manual approval steps to be applied to any artefact that has successfully passed its testing phases. This can of course be fully automated with the Release being packaged upon Test completion. It is not recommended however, as it is quite possible that the final set of released artefacts might be a group of more than one artefact and doing a coordinated release of all the associate components will provide a more coherent release.

## Container Security

IBM Agile Lifecycle Manager microservices run on a Red Hat Openshift Container Platform restricted Security Context Constraint. Microservices run with an arbitrary user and are not permitted to run as root or with escalated privileges.

The Security Context definition for IBM Agile Lifecycle Manager containers sets the following fields:

```
privileged: false
readOnlyRootFilesystem: true
allowPrivilegeEscalation: false
runAsNonRoot: true
```

Refer to Red Hat Openshift Container Platform documentation (https://docs.openshift.com) for details on Security Context Constraints and container SecurityContext.

# Notices

This information applies to the PDF documentation set for IBM Agile Lifecycle Manager.

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement might not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
958/NH04
IBM Centre, St Leonards

## Trademarks

SC28-3145-00