Codar

Software version: 1.81

Integration with HPE Operations Orchestration

Document release date: May 2017

Software release date: May 2017

Legal notices

Warranty

The only warranties for Hewlett Packard Enterprise products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Hewlett Packard Enterprise shall not be liable for technical or editorial errors or omissions contained herein. The information contained herein is subject to change without notice.

Restricted rights legend

Confidential computer software. Valid license from Hewlett Packard Enterprise required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright notice

Trademark notices

Adobe® is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

UNIX® is a registered trademark of The Open Group.

RED HAT READY™ Logo and RED HAT CERTIFIED PARTNER™ Logo are trademarks of Red Hat, Inc.

The OpenStack word mark and the Square O Design, together or apart, are trademarks or registered trademarks of OpenStack Foundation in the United States and other countries, and are used with the OpenStack Foundation's permission.

Documentation updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.

- Document Release Date, which changes each time the document is updated.

- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to the following URL and sign-in or register: https://softwaresupport.hpe.com.

Select Manuals from the Dashboard menu to view all available documentation. Use the search and filter functions to find documentation, whitepapers, and other information sources.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your Hewlett Packard Enterprise sales representative for details.

Support

Visit the Hewlett Packard Enterprise Software Support Online web site at https://softwaresupport.hpe.com.

Codar **1.81**

Contents

Overview

Most IT tasks (e.g., provisioning of VMs, software installation, registration of users into a system, etc.) can be using HPE Codar. Codar provides your organization with a simple and scalable way to create complex cloud topology designs using our declarative topology designer. Tasks that would normally take hours take only minutes with our innovative approach to service design.

With support for infrastructure providers such as VMware and Amazon, as well as many HPE products such as HPE SiteScope, you have the tools and integrations at your disposal to meet your service needs.

The set of IT tasks required for infrastructure or platform provisioning is wrapped by Codar components with a simple lifecycle: deploy, manage, and undeploy. Codar can import design components from supported providers, and then these components can be modeled together into a declarative topology design.
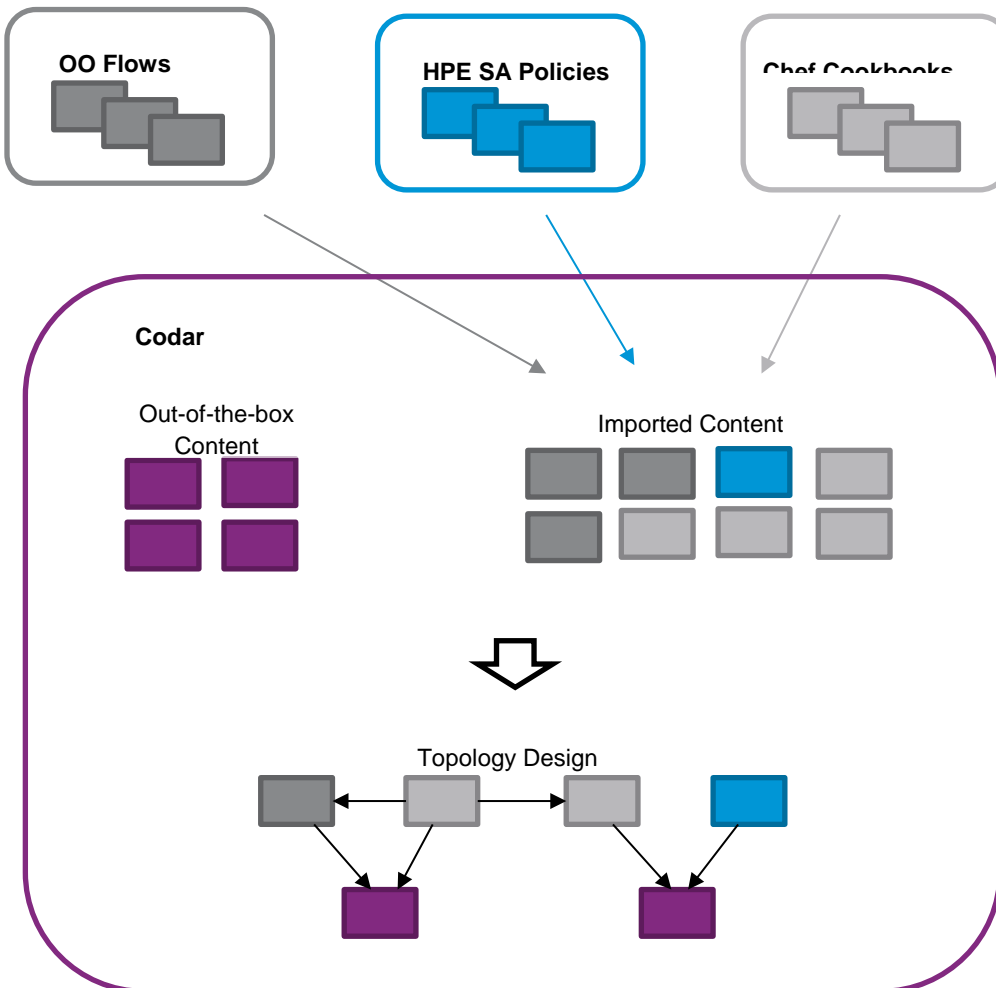
The topology design consists of components connected with relationships. Components represents the final shape of the deployment, and the relationships are dependencies between the components. The designer user just declares *what* should be done and Codar figures out *how* it should be done. The sequence of the tasks is automatically computed based on designed dependencies among components.

After a topology design is created, Codar will generate a proper execution plan consisting of tasks required to get the design deployed. Codar uses Operations Orchestration (OO) for the processing of the execution plan. The execution plan is represented by an OO master flow. The OO master flow consists of steps representing the deployment of each component. The details behind each individual component can be customized with your specific business logic.

Codar works with various kinds of content. There are components available in Codar out-of-the-box. These are infrastructure components like server or network provisioned by VMware vCenter or Amazon EC2.

Besides the out-of-the-box components, Codar can load content from other systems, representing it as components and use these imported components in topology designs. Currently, Operations Orchestration flows is supported.

Terms and Prerequisites

These terms are used in the following text.

- *Topology design* composed of *Components* connected with *Relationships.*
- *OO* is an abbreviation for Operations Orchestration.
- *OO master flow* is a complex flow representing the whole design execution plan.
- *Standard OO content* is content created in accordance with the best practices. It is described more in this document.
- *Server capability* is an annotation of infrastructure components denoting that these are server-like components.

Codar Content Sources

Topology designs are composed of components. Codar works with components coming from sources listed below.

Codar Out-of-the-Box Content

This is the content present in Codar after installation (VMware vCenter, Amazon EC2 components). The document does not go into further detail on these components.

HPE Operations Orchestration

Operations Orchestration (OO) links automated tasks into flows. From Codar point of view, the result of a flow execution can be wrapped as component that can be used as a building block for more complex designs.

Let's clarify that Codar uses OO in several ways:

- Process executor for sequenced designs.
- Process executor for topology designs –OO master flow is the implementation of the design execution plan.
- Component import source – New topology components can be imported from existing OO flows. This use of OO is discussed in depth later in this document.

Content Provided by Operations Orchestration

Operations Orchestration (OO) serves as a content provider for Codar. The OO flows can be imported in Codar as components and form more complex topology designs.

Codar – OO Integration

Codar comes with a number of components. Many of them rely on integration between Codar and OO. Codar must be configured after installation so that it communicates with a running OO server.

OO Server Setup

For OO server installation help, please, check the Operations Orchestration documentation. Some Codar specific hints follows.

Log into the OO Central web interface to create an *admin* user.

- Username: admin

- Password: cloud

- Roles: ADMINISTRATOR
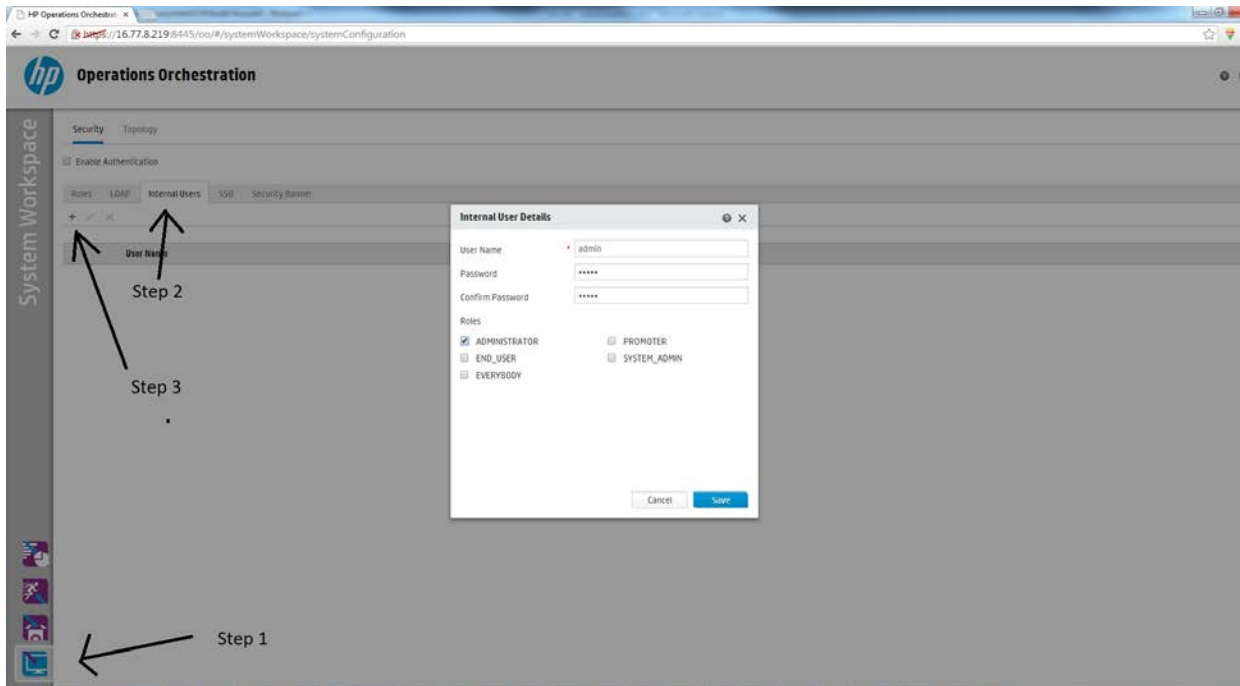
Upload OO Content

Log into the OO Central web interface to upload the content.

Upload the Codar content capsule using the CSL content installer. You can access the CSL content installer from the tools/CSLContentInstaller directory after installing Codar.

Upload the following content packs to OO in the order specified below.

1. `$Codar_HOME/oo/OOContentPack`

   a. `oo10-base-cp-<version>.jar`

   b. `oo10-cloud-cp-<version>.jar`

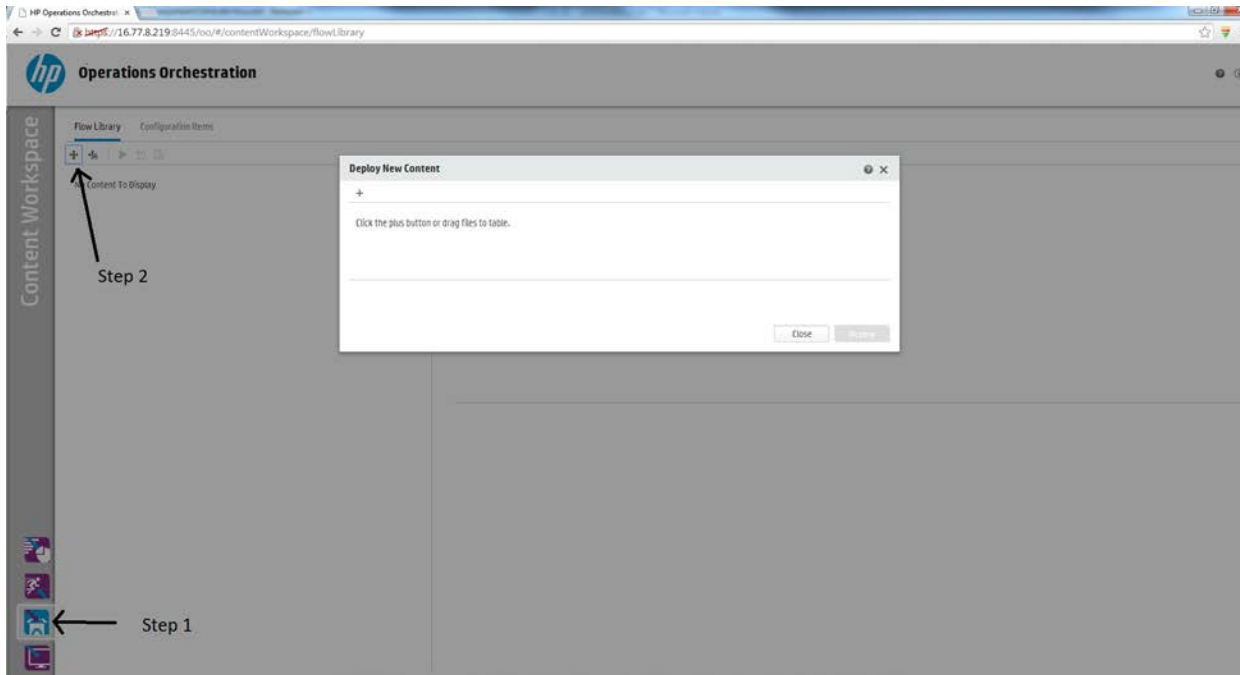   c. `oo10-hp-solutions-cp-<version>.jar`

   d. `oo10-virtualization-cp-<version>.jar`

   e. `oo10-sm-cp-<version>.jar`

2. `$ Codar_HOME/Tools/ComponentTool/contentpacks`

   a. `CSA-CONFIG-CP-4.10.0000.jar`

   b. `CSA-VMWARE-CP-4.10.0000.jar`

   c. `CSA-AMAZON-CP-4.10.0000.jar`

   d. `CSA-HP-HELION-PUBLIC-CLOUD-CP-4.10.0000.jar`

   e. `CSA-SITESCOPE-CP-4.10.0000.jar`

Standard vs. Custom Content

Codar is able to create components automatically by importing existing OO content, specifically OO flows. To enable the feature, Codar expects content to follow a specific naming and structure convention. The convention is described in the guideline section below.

OO content following the guideline is called *standard content*. Note that Codar is capable of importing almost any (existing) content even if it does not follow the guidelines. Such OO content is called *custom content*.

Standard OO Content Guideline

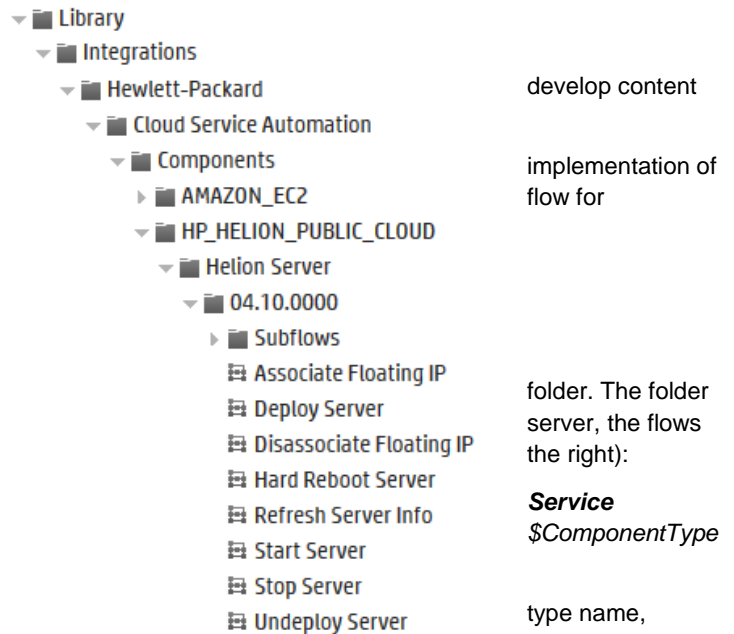These instructions are for developers who are going to develop content for OO and want to import that content into Codar.

A component is built from a set of flows. The flows are an implementation of the component operations. At a minimum there must be a flow for deployment or provisioning.

Folder Structure

All flows representing a component need to be placed in one folder. The folder structure is important. In the content pack or on the OO server, the flows need to be stored in a path of the following form (picture on the right):

Library -> Integrations -> Hewlett-Packard -> Cloud **Service** **Automation ->** *Components -> $ProviderTypeName ->* **$ComponentType** *-> $Version -> @Flows*

The values beginning with *$* are placeholders for the provider type name, component type, and version, which will be discussed further in the following section. The *@Flows* value represents a list of individual flows in the folder. Occasionally, there can be a *Subflows* folder among the flows gathering additional entities used in the flows.

```
▾ Library
   ▾ Integrations
      ▾ Hewlett-Packard
         ▾ Cloud Service Automation
            ▾ Components
               ▸ AMAZON_EC2
               ▾ HP_HELION_PUBLIC_CLOUD
                  ▾ Helion Server
                     ▾ 04.10.0000
                        ▸ Subflows
                          Associate Floating IP
                          Deploy Server
                          Disassociate Floating IP
                          Hard Reboot Server
                          Refresh Server Info
                          Start Server
                          Stop Server
                          Undeploy Server
```

Provider Type

These are out-of-the-box provider types in Codar. You can also create your own. The provider type name values are inferred from the provider type label. If you create "Custom Provider Type," the name will be uppercase with underscores like CUSTOM_PROVIDER_TYPE. The uppercase provider type name must be used as the folder name for the standard content. The provider type name value is also available in the Codar Provider Management UI for reference.

Provider type enum:

- AMAZON_EC2
- CHEF
- HP_DMA
- HP_ONEVIEW
- HP_SA
- HP_SITESCOPE
- HP_VIRTUALIZATION_PERFORMANCE_VIEWER
- OPENSTACK
- PUPPET
- VMWARE_VCENTER

Component Type

The *$ComponentType* is a placeholder for component type name, which appears as one of the folders in the flow path used for standard content. The folder name is used as the component type name in Codar. Note that that name can be changed later when

editing the component. Keep in mind that the provider type related folder (the parent) does not affect the component name. If you have a folder structure such as: *Library -> Integrations -> Hewlett-Packard -> Cloud Service Automation -> VMWARE_VCETER -> Server -> 4.20.0000*, your component would be called 'Server'. With this approach, you will end up with many 'Server' components in Codar. Moreover, the component name together with provides unique identification of the component. Codar does not let you import multiple components named 'Server' if they have the same version, regardless of the related provider type. Consider using more descriptive names for a component, such as 'OpenStack Server' to avoid such naming conflicts.

Version

Codar does not provide component version management capabilities. However, import of versioned standard content is supported. A component is created with a version matching the leaf folder in the flow path. This component version, along with name, uniquely identifies the component to Codar.

Flows

The individual flows are placed under the version folder. The name of a flow is used automatically as an operation name in the new component. The flow name is important for recognition of the lifecycle phase related to the operation.

Each OO flow should have an input property defined as a constant called *LIFECYCLE_PHASE*. The property does not relate to the flow logic. It is used as meta-information marking the flow purpose related to the component lifecycle.

As you can see, there is a duplicate indicator of the operation lifecycle phase. Both the flow name (beginning of the name in fact) and *LIFECYCLE_PHASE input property need to be set appropriately. This duplication will be removed in a future release. Currently,* the flow name and the value of the constant property should be set as follows:

Flow Purpose	Flow Name Prefix	LIFECYCLE_PHASE
Deployment	*Deploy* or *Create*	deploying
Undeployment	*Undeploy* or *Delete*	undeploying
Modification	*Modify*	modifying
Undo the modification	*Unmodify*	unmodifying
Handle a failure during deployment		deploying_failure
Handle a failure during undeployment		undeploying_failure
Handle a failure during modification	*Modify Failure*	modifying_failure
Custom public action executable on a deployed instance		deployed

More information about the component lifecycle can be found in the section Lifecycle Operations

Input and Output Properties

OO server automatically decorates all flows with a ***Result*** output property. However, it is suggested to define flow outputs explicitly as output properties.

- ***response*** – Mandatory property. Each OO flow used in a component operation should have a ***response*** output property. Codar relies on the ***response*** property to determine the state of the execution.
  - For deployment, undeployment flows and flows related to public actions, the value should be either ***success*** or ***failure*** based on flow execution response.
  - For modification flows, the value should be success, noop, or failure. Here success indicates that the attempted modification was successful, noop indicates that the attempted modification failed but was rolled back successfully, while failure indicates that the attempted modification unsuccessful and wasn't rolled back.
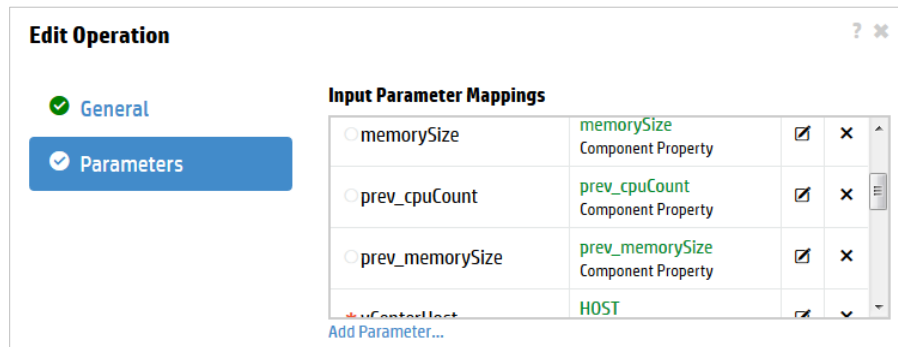
- If you include the deploy failure handler and/or undeploy failure handler, make sure that the **response** is being set correctly. The fact that these actions are called should result in the **failure** response.

- *result* – Optional property. Each OO flow used in a component operation should have a *result* output property. It can hold any contextual message, like detailed info about failure.

Codar supports the modification of properties for an active subscription. Codar will pass previous property values to modification OO flows if the flows explicitly express interest in knowing these values - by defining additional flow inputs with a *prev_* prefix. As an example, if the modification OO flow defines an input of *memorySize* and is interested in knowing its previous value, it must also define an input of *prev_memorySize*.
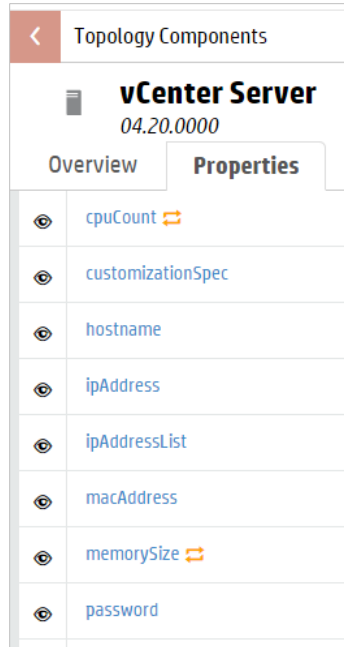
Illustrated in the picture alongside, are the *cpuCount* and *memorySize* properties that the flow has expressed interest in knowing previous values of. This is denoted by an icon ⇄ next to the property name in the Codar Topology Components view.

Note that the *prev_* OO flow inputs do not show up as properties for the component; only their unprefixed counterparts show up and are appropriately marked. However, the *prev_* OO flow inputs do show up on Operations' Input Parameter Mapping. These mappings are used by the modification operation and should be not edited.

A component's *Modify* flow may communicate the status of the modification, an error code or any other useful information to *Modify Failure* flow by placing such information in an output field called *modifyReturnValue*. This value may be received by the *Modify Failure* flow in an input field called *modifyFailureValue* and allows the Modify Failure flow to do cleanup the effect of failure in Modify flow intelligently, especially for a complex multi-step flows when failure might have occurred at any step which can be indicated by returning specific error code from the Modify flow via *ModifyReturnValue*. If these fields are set on the OO flows, the mapping between them is automatically handled within Codar.

OO Content Import

This chapter describes the process of Codar component creation from OO flows. It covers usage of both standard and custom OO content. The content is imported into Codar and encapsulated into components. The component can be then used in topology designs, but first it usually requires a bit of manual adjustments and fine-tuning.

All Codar components have properties, operations and relationships.

- Property values are the input parameterizing the component realization. Properties can also hold results coming from the realization to be displayed or even used by another component.

- Operations are of two kinds:

  - Lifecycle operations are coupled with the component deployment and undeployment.

  - Custom operations can later be exposed as public actions so that user can execute them in Operation console.

- Relationships to other components that can express both optional and required dependencies. During provisioning, a component can use outputs of components it is depending on.

Because the component is made of OO flows, each flow is represented by one component operation.

Import Wizard

1. From the Topology Components area, click Import, and provide the following information:

   - Import Source - Select the provider type to use as the import source for the component.Provider Instance - Select the provider instance that contains the component you want to import.

- Provider instances must first be configured in the Providers area to enable import for Chef, HPE Server Automation, and Puppet provider types. For Operations Orchestration, content can be imported from the instance that is configured to integrate with HPE Cloud Service Automation.

- Image - Select an Image that will display for the component.

- Tags - Select one or more tags that will include the component. Tags are user-defined, color-coded labels and images used to provide a structure for organizing and grouping topology components.

2. Depending on your selections in the General tab, select the content for the component:

- Chef - Select one or more Chef cookbooks to be imported as new Cloud Service Automation topology components. Each selected cookbook will create a separate component.

- OO - Browse the OO library and select one or more standard component elements that will be imported as new topology components. Each selected item will create a separate component.

- Server Automation - Select one or more Server Automation policy to be imported as new topology components. Each selected policy will create a separate component.

- Puppet - Select one or more Puppet classes to be imported as new topology components. Each selected class will create a separate component.

3. Review your import selections, and click Import. The imported components will appear in the Topology Components list.

OO Import Limitations

Standard content import works seamlessly. *Custom* content (not following the best practices) can be in most cases imported as well. There are some known limitations, however.

Custom Server Component

Component import is capable of creating a new component from a group of OO flows. However, in case of a new server-like component (meaning a machine you can install things on - like an VMWARE_VCENTER server for example) the import wizard does not provide a way to annotate the component with the *Server* capability. It is not possible to assign the capability to the component even from the component editor UI.

So, the component is successfully imported but it does not behave like other OOTB server components. Relationships targeted to the *Server* capability are not available for the new server. The relationships need to be added manually and specially for the new component.

Moreover, the *privateKey* property field on the new component cannot be used because the UI provides no way to modify the default 255 character length restriction for custom component properties.

There are other ways to get the new component annotated with the *Server* capability:

- After the import, adjust the component using public REST-full API, which does not have the limitations of UI. Use endpoint

  ```
  http://<codar_url>//api/topology-model/component-type
  ```

- Or import the custom server component using the Component Tool shipped as part of the Codar installation.

  - Prepare OO flows to use them with Component Tool.

  - Create a property mapping file for the component. Look for examples in the Component Tool *mappingFile*s folder.

  - Update the capability mapping file *mappingFile*s/*capability_mapping.properties* to assign the Server capability to the component and to map required properties like *ipAddress, username, password, privateKey*.

OO Operations

Among the OO entities, there are flows and low-level operations. The OO operations are only building blocks that should be used as part of flows. In contrast to OO flows, OO operations cannot be imported as Codar components.

Single Flow Operations

An imported component operation invokes an OO flow when the component is provisioned as part of topology design provisioning. Each operation is linked to and can invoke just one OO flow.

Unique Lifecycle Operations

Codar defines the following lifecycle operations:

- Deploy
- Deploy Failure Handler
- Modify
- Unmodify
- Modify Failure Handler
- Undeploy
- Undeploy Failure Handler

A component can have a single lifecycle operation of each type, e.g. single *Deploy* operation etc. So, when an operation is set to be *Deploy*, any other operation that was previously configured to be *Deploy* will have its lifecycle operation unset.

Fine-Tuning the Imported Content

Once the content is imported, components are created. The components usually need some additional adjustments like relationship definition, lifecycle configuration for operations, operation parameter mapping and so on. The amount of additional work required depends on the nature of the imported content and user requirements.

Component Overview

The imported content is represented as components in Codar. Components are parameterized with input properties, have relationships to another components, go through a lifecycle via operation execution possibly generating values for output properties and optionally provide additional operations, called public actions that can be requested to run by subscribers.

Before you start modifying components, note that not all changes are possible.

- The *Server* capability is read-only and cannot be changed.
- Other components can be changed. However, in case a component is already used in a design, changes are restricted.
    - The component name, description and icon can be changed.
    - The component can be tagged.
    - Non-required properties can be added.
    - Non-required relationships can be added.
    - Custom non-lifecycle operations (public actions) can be added.
    - Other changes related to required properties and relationships and lifecycle actions are prohibited once the component is used in a design.

The most important part of a component are probably operations and the trickiest part is an operation parameter mapping.
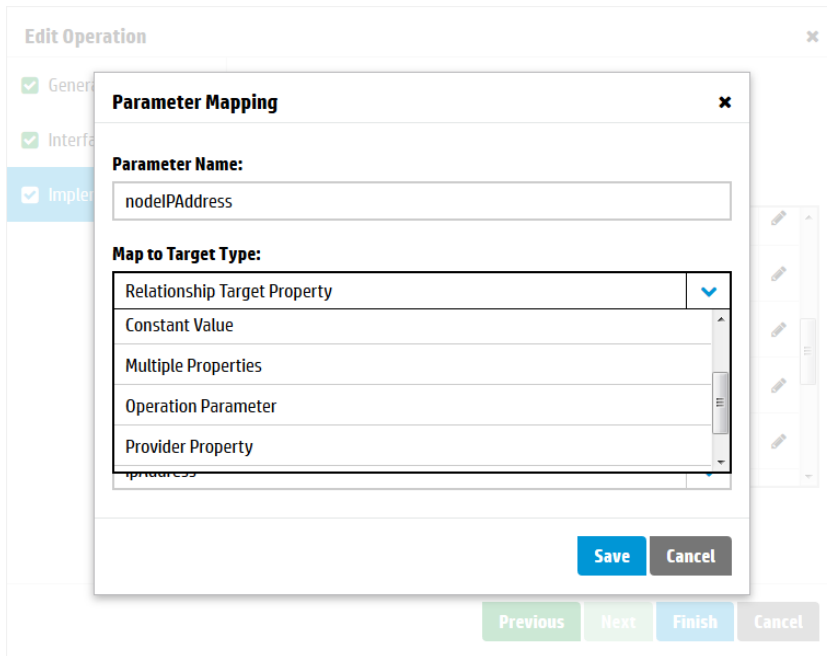
There are a couple of different ways how to get a value for an operation input parameter.

- *Not Mapped* – the operation parameter gets no value automatically. This is invalid state for a lifecycle operation (e.g. *Deploy, Undeploy*). All parameters need to be mapped.

  However, in case of a public action, *Not Mapped* parameter is valid option. It results in value prompt during execution.

  Note that there is a known defect breaking the value prompt. The defect will be fixed in the nearest patch.

- *Component Property* – the operation parameter gets filled using the component property value. This is the most common and straightforward way.

- *Constant Value* – the operation parameter value is a constant.

- *Multiple Properties* – a value for the operation parameter is combined from multiple places. It is a kind of recursion, so a next level of mapping resides here.

- *Operation Parameter* – This option has not been fully implemented and should not be used.

- *Provider Property* – the operation parameter uses a property defined on the provider instance used for provisioning.

- *Relationship Target Property* – a property defined on another component connected via relationship is used for the operation parameter. In case the selected relationship is not in place, the source component property is used instead. The behavior is the same as the case of *Component Property* mapping. It is a kind of fallback used in scenarios utilizing optional relationship that is there in some designs but is not there in others.

Server Capability

The topology components do not form a hierarchy. Useful aspects of inheritance are covered by a concept of capabilities. A capability can be considered a tag indicating what a component is good for or capable of. However, it is more than just a tag. A capability can have properties and relationships making it look more like an abstract component.

The *Server* is the only capability in Codar so far. Components with this capability have server related properties *ipAddress, hostname, username, password, privateKey* and *instanceId*.

Note that at design time, when a Server component is used, you need to decide which authentication method should be used by provisioning. In case you would like to use *username* + *pasword*, leave *privateKey* field empty. Otherwise, *privateKey* will be used.
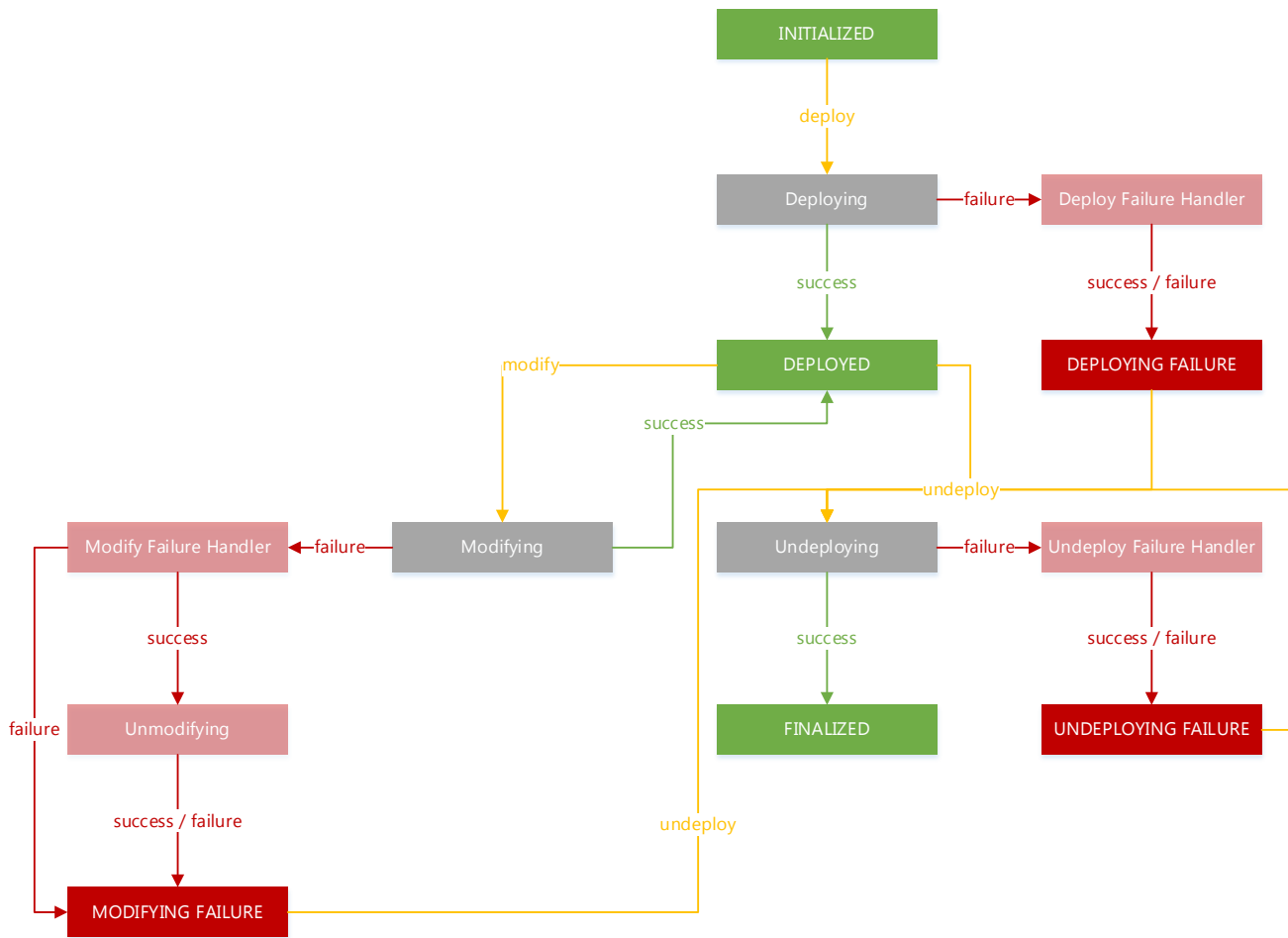
Lifecycle Operations

Components in Codar go through lifecycle stages as they are provisioned. The lifecycle of topology components is simpler than the lifecycle of sequenced components.

- Green boxes are stable success states.

- Red boxes are stable failure states.

- The faded gray and pink boxes are execution states. Transition to the next state triggers once the execution is finished.

- Yellow transitions from a stable state triggers when a component operation is explicitly invoked.

- Other transitions are automatically triggered.

The lifecycle operations that may be defined on a component are *Deploy*, *Undeploy*, *Modify*, *Unmodify*, *Deploy Failure Handler*, *Undeploy Failure Handler*, *and Modify Failure Handler*.

The *Deploy* operation is triggered when a new request is submitted. An *Undeploy* is triggered when an existing subscription is cancelled. A failure in deploying or undeploying automatically results in the invocation of the corresponding failure handlers.

A previously deployed subscription may be modified. When the modification operation is successful the component moves back to the Deployed state. If a failure occurs during modification, there's an attempt to roll back all the operations executed so far in order to take the involved components to the state prior to the modification attempt. A failure at the time of modification results in the invocation of the *Modify Failure Handler*, which in turn triggers the *Unmodify* operation. If Unmodify is successful, the component's properties are rolled back. However, if Unmodify is not successful, the component's properties are retained and the consumer may resubmit the last modification. It is likely that resubmitting the last modification may result into similar failures as before and the operator may have to intervene to resolve the situation.

To define a lifecycle operation, edit the component, select a desired operation and select the appropriate value from the drop-down menu.

Note that lifecycle operations are automatically recognized if the component has been created using standard OO content.

Custom Operations

Besides the operations automatically created for you during content import, you can define additional custom operations. The Import button on the Operations tab initiates a wizard similar to the wizard used for component creation (see OO Content Import). Just select an OO flow that you want to use as implementation of the new Operation.

Component Properties

The new created component comes with a set of input and output properties that match the OO flow input and output. That way the component is just a wrapper around the OO flow used for the deployment. We want the component to be more than that.

There are several typical groups of properties for a component.

- Properties related to the resource provider like service endpoint and credentials for Amazon EC2.

- Properties related to other components like IP address of a server used for installation.

- Properties really parameterizing the component.

Properties from the first group of resource provider related properties are not shown on the component if a proper parameter mapping has been used during the content import.

The second group of properties related to values provided by other components could be dropped from the component in case we could somehow specify how to get the value for the OO flow input. See Relationships to Other Components Relationships to Other Components chapter below.

The last group is about properties that really belongs to the component and need to be filled by a user.

A property can be configured so that it is required or optional, visible or hidden in designer, modifiable, or it can have a default value. Property type can be selected and string properties can be set confidential. Confidential properties are rendered using password field so that the value is not visible in plain text. Note that all values are passed to OO master flow during provisioning and the confidential values are not obfuscated. An exception are the out-of-the-box components that have the password obfuscation implemented.

Relationships to Other Components

A component usually works together with other components in deployments that are more complicated than a simple server. Topology designs capture the component dependencies and connections using the concept of relationships.

Although it is not visible in the designer, relationships are oriented from a source component to a target component. The orientation is important because it marks a dependency of the source component on the target component.

A relationship has several functions:

- Denotes dependency between two components in a topology design.

- The order of component realization during the provisioning is driven by dependencies represented by the relationship.

Property values can be passed from one component to another along the relationship.

Testing and Debugging

To test a component, you have to put it in a design and test the whole design. Codar provides Test Run feature for topology designs. Test Run results are displayed including events details together with links to the OO master flow execution log.

Refer to the OO related hints and limitations in chapters OO Import Limitations .

Modification Scenarios

The expectation with a typical modification is that the *Modify* operations of each of the components run successfully to completion. In the picture below, a successful execution of the *Modify* operation for two components is depicted.

A successful *Modify* operation should return a response of **success**.

If the *Modify* operation for a particular component fails, its corresponding *Modify Failure Handler* is invoked. The failure handler provides an opportunity for cleanup. The **modifyReturnValue** output from the Modify operation feeds into the **modifyFailureValue** input of the *Modify Failure Handler* if information must be communicated from the *Modify* operation to its corresponding *Modify Failure Handler*.

On successful completion of the *Modify Failure Handler*, a rollback of previously deployed components is initiated. When all previously modified components are successfully unmodified, they move back to the *Deployed* state while the component that failed (Component 3 in the picture below), is set to *Modifying Failure*.

Additionally, the options selected when *Modify Subscription* was initiated, are reverted when the rollback is successful. Subscribers are free to make different selections while resubmitting their previously failed modification request.

The *Modify Failure Handler* operation must always return a **failure** response. In the example below, the **failure** response would indicate that the modification failed for Component 3. A successful *Unmodify* operation, should return a response of **noop**, indicating that the rollback for that component was successful. While, a failed *Unmodify* operation should return a response of **failure**.
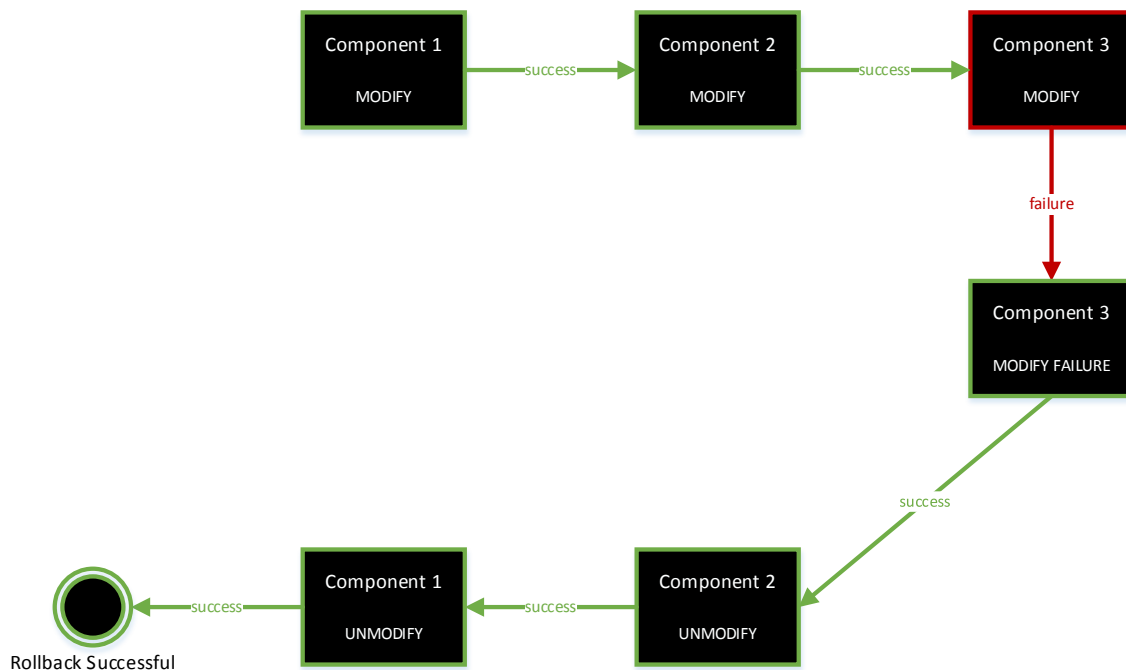
Failures may occur while executing the *Modify Failure* operation or the *Unmodify* operations, as depicted in the next two examples.
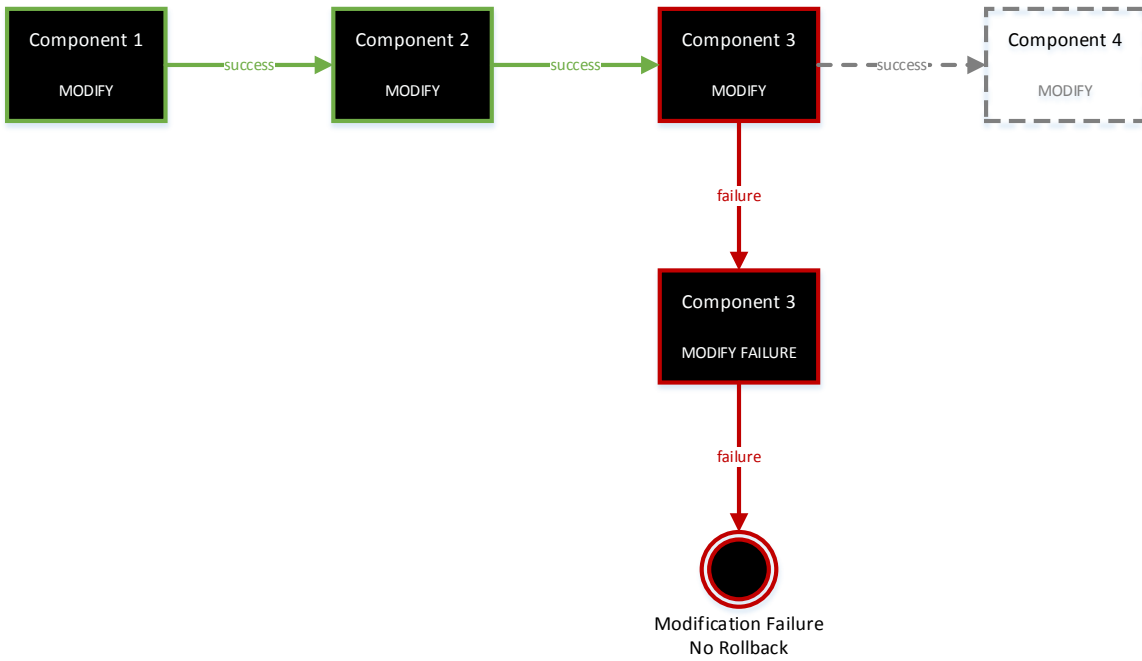
A rollback is either not initiated or stopped as soon as a *Modify Failure Handler* or an *Unmodify* operation fails. In both scenarios, the subscriber may not change the options that were previously selected for a subsequent resubmission. It is likely that the operator may have to intervene.
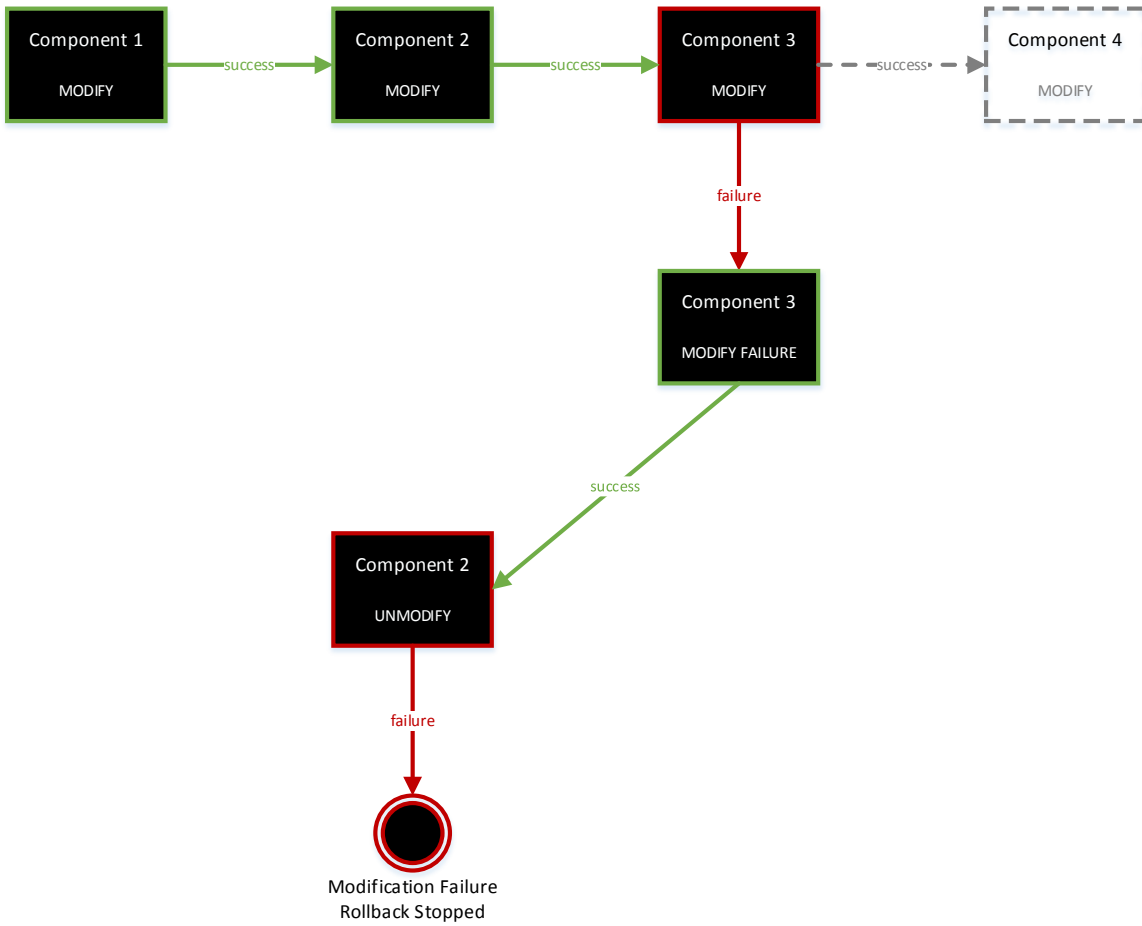
Configuring Modifiable Properties

Properties on a topology-based service design must explicitly be marked as *modifiable*. Designers may choose to mark properties as *modifiable* only at the time of service creation or both at the time of service creation and service modification.

Only properties defined with their *prev_* counterparts may be marked as *modifiable* during service modification. In the illustration, **cpuCount** and **memorySize** can be marked by the designer as *modifiable* during modification; the designer has chosen to mark only **cpuCount**. Notice that the other properties, such as vmTemplateReference or vmNamePrefix, can't be marked as *modifiable* during modification.

Properties for various profiles may also be marked as *modifiable* during service creation and modification.

Send documentation feedback

If you have comments about this document, you can contact the documentation team by email.

If an email client is configured on this system, click the link above and an email window opens. Add subject and your feedback to the email, and click send. If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to clouddocs@hpe.com.

We appreciate your feedback!