Hewlett Packard
Enterprise

Cloud Service Automation

Topology Components Guide

Software version: 4.80

Document release date: January 2017

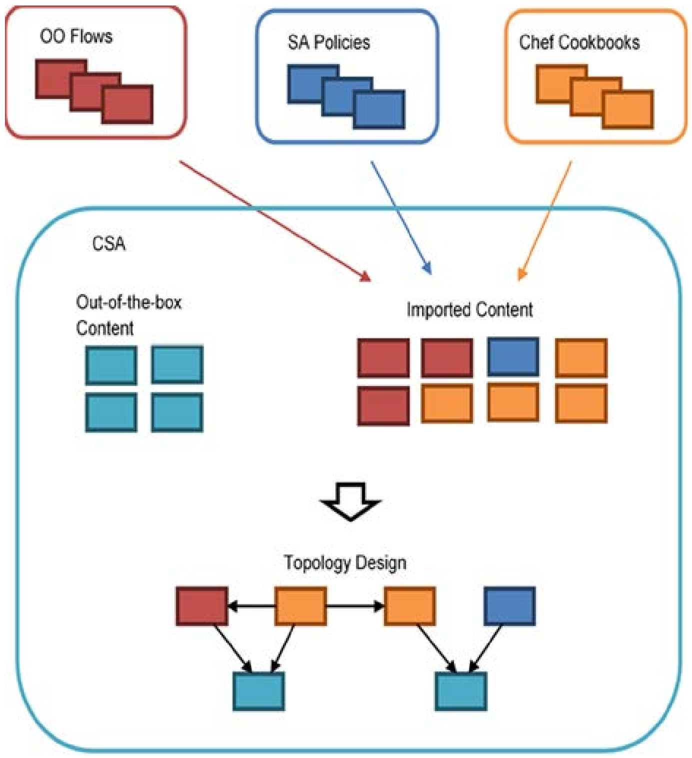
Software release date: January 2017

Contents

Concepts overview

HPE Cloud Service Automation (CSA) automates and substantially speeds up many IT tasks, such as provisioning VMs, installing software, or registering users. With the CSA Topology Designer you can design scalable, complex cloud topology. CSA integrates with various infrastructure providers, such as Helion OpenStack, VMware, Amazon, and Chef, and many enterprise products, such as HPE SiteScope (SiteScope) and HPE Server Automation (SA).

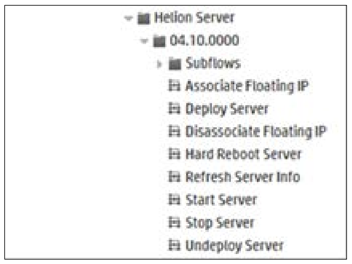
CSA design components execute the set of IT tasks required for infrastructure or platform provisioning using a simple lifecycle: deploy, manage, and undeploy. In addition, CSA can import design components from supported providers. For example, in the case of a Chef provider, CSA can import Chef cookbooks into CSA design components. Imported components are then modeled into a declarative topology design.



Each topology design consists of *components* connected by *relationships*. Components represent the final shape of the deployment, and the relationships represent the dependencies between the components. The designer user declares *what* should be done, and, based on the design, CSA performs the necessary actions. The sequence of the tasks is automatically computed, based on designed dependencies among components.

Once a topology design is created, CSA generates a proper execution plan consisting of the tasks required to get the design deployed. CSA uses Operations Orchestration (OO) to process the execution plan, which 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.

Important: A **component** is built from a set of flows. These flows implement the component operations. At a minimum, there must be a flow for deployment or provisioning. The following graphic shows examples of flows.


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

CSA works with content of various types, including CSA out-of-the-box components. These are infrastructure components (like server or network) that can be provisioned by VMware vCenter, Amazon EC2, or Helion OpenStack.

Besides the out-of-the-box components, CSA can load content from other systems, representing this content as components, and then use these imported components in topology designs. Currently, OO flows, SA policies, and Chef cookbooks are supported.

Component overview

The imported content is represented as components in CSA. Components are parameterized with input properties and have relationships to other components. In addition, the components go through a lifecycle during operation execution, which can generate values for output properties and optionally provide additional operations, called *public actions*, which can be requested by Subscribers.

Component characteristics

Components can be changed *unless*:

- They are server-capability components or Helion OpenStack components, which are read-only.
- They are already used in a design. If so, changes are restricted to the following:
  - Component name, description, and icon.
  - Component tag.
  - Non-required properties, non-required relationships, custom non-lifecycle operations (public actions).

  Other changes related to required properties, relationships, and lifecycle actions are not allowed once the component is used in a design.

Component operations

Note: Use caution when configuring operation-parameter mapping. If you map something incorrectly, an erroneous value might be passed, and your deployment will fail.

There are several ways to get a value for an operation-input parameter:

Name	Description
`Not mapped`	The operation parameter automatically gets no value. This is an invalid state for a lifecycle operation (for example, *Deploy, Undeploy*). All parameters must be mapped. However, in the case of a public action, the *Not Mapped* parameter is a valid option and results in a value prompt during execution.
`Component Property`	The operation parameter is filled using the component property value (preferred).
`Constant Value`	The operation parameter value is a constant.
`Multiple Properties`	A value for the operation parameter is combined from multiple places. This is a kind of recursion, so another level of mapping resides here.
`Operation Parameter`	This option should not be used as it has not been fully implemented.
`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 a relationship is used for the operation parameter. In case the selected relationship is not in place, the property with the same name (if present on the component) is used. The behavior is the same as Component Property mapping. A fallback mechanism is used to map this parameter to a property with the same name on the component itself, if that property is present in the component.

Component Properties

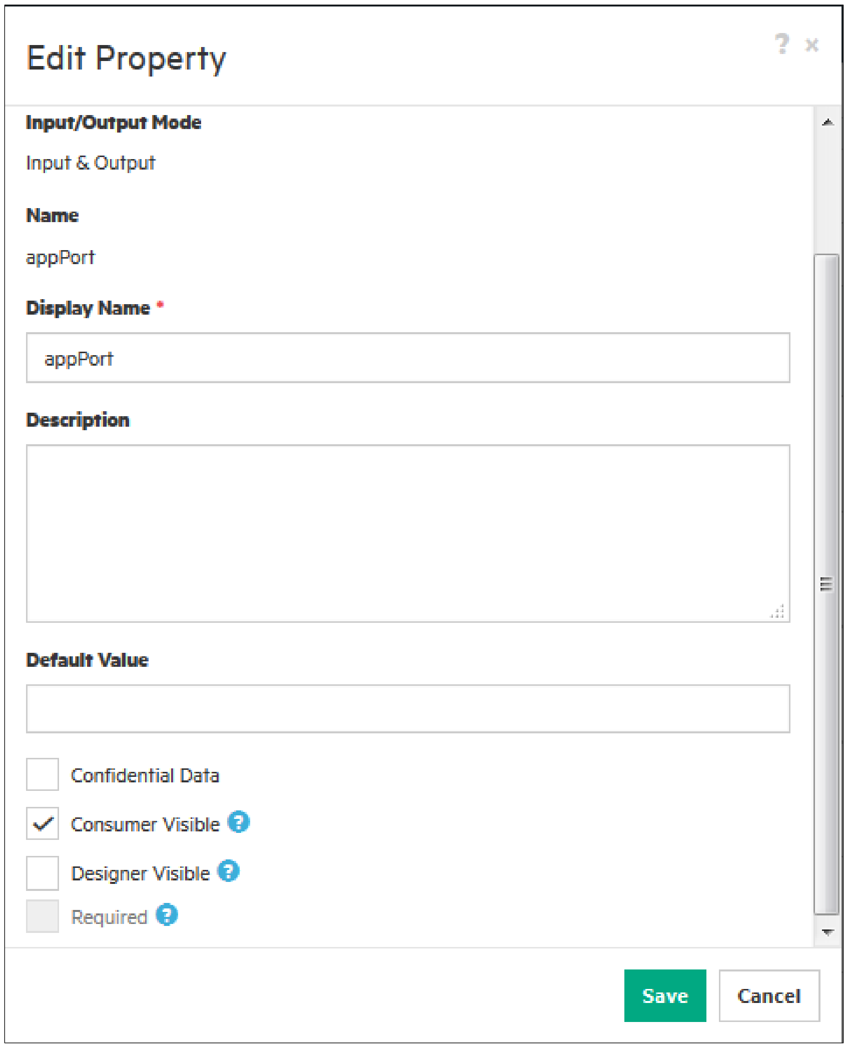
A new component automatically has a set of input and output properties matching those of the input and output OO flow. These properties allow the component to be an OO-flow or Chef-deployment cookbook wrapper.

Components have the following groups of properties:

- Properties related to the resource provider, such as service endpoint and credentials for Amazon EC2.
- Properties related to other components, such as the IP address of the server used for installation.
- Properties parameterizing the component.

  Note: When a component has a defined relationship, you do not need to define a property for an input parameter in that component, if the parameter value originates from the property on another component. See Relationships to other components for more information.

- Properties specified by users in the Edit Property screen:



Edit Property

Input/Output Mode

Input & Output

Name

appPort

Display Name *

appPort

Description

Default Value

- [ ] Confidential Data
- [✓] Consumer Visible
- [ ] Designer Visible
- [ ] Required

A component property is either **required** or **optional**, **visible** or **hidden** in the Designer, **modifiable**, or has a **default** value. Selected property types and set string properties can be configured as confidential data that is consumer and designer visible. Confidential data properties are not visible in plain text and will be in the MPP portal. Designer-visible properties will make this property visible and editable in the topology designer GUI.

Note: All values are passed to the OO master flow during provisioning. Passwords are obfuscated in OO, but out-of-the-box components already have password obfuscation implemented.

Relationships to other components

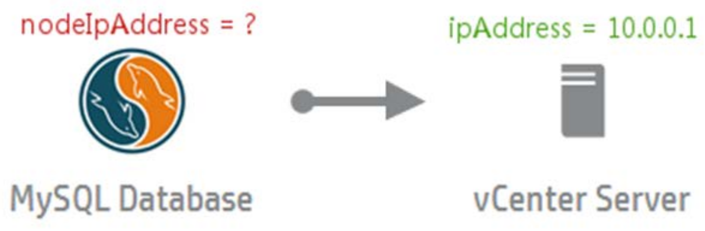
A component usually works with other components in deployments that are more complicated than the deployment of a single server. Topology designs capture the component dependencies and connections using the concept of *relationships*.

A relationship has several functions:

- Denotes dependency between two components in a topology design.
- Specifies the order of component realization during provisioning, which is driven by dependencies represented by the relationship.
- Defines property values that can be passed from one component to another along the relationship. 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. If Component A has a relationship to Component B with the arrow pointing to Component B, it implies that Component A depends on Component B and will be realized after Component B is realized.

Passing values between components

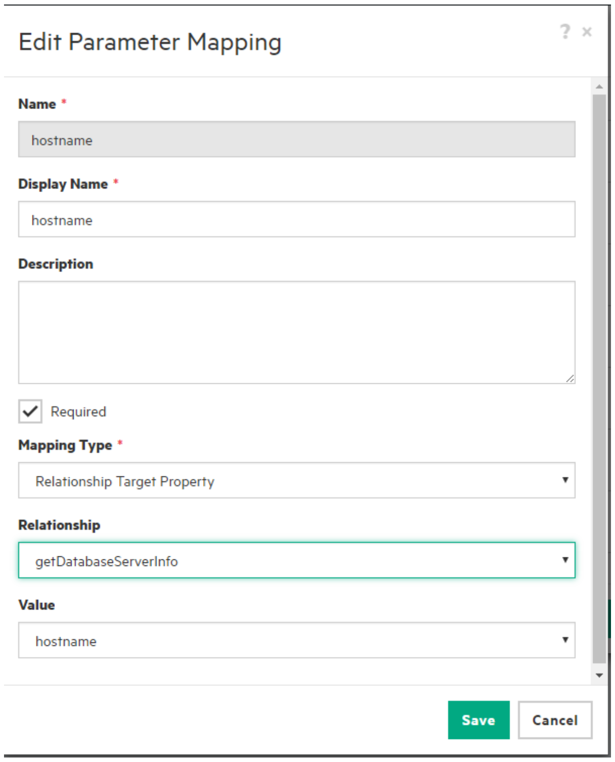
The common example of a relationship is a piece of software installed on a server. *MySQL* component realized by a Chef cookbook has a relationship *hostedOn* that targets Server capability. During the provisioning, the *MySQL* reads the server IP address using the *hostedOn* relationship.



At provisioning time, CSA first resolves dependencies between components, then creates the server, and lastly creates the database. During database creation, the server IP address is required so CSA can run the Chef cookbook for MySQL installation. The Deploy lifecycle operation runs provisioning, taking *`nodeIpAddress`* as one of the inputs. The value is captured using the relationship from the `vCenter Server` output property, which is already provisioned. The following formula describes how the IP address value used during MySQL deployment is retrieved from the target of the *hostedOn* relationship (vCenter Server).

```
MySQL.Deploy.nodeIpAddress = MySQL.hostedOn.ipAddress
```

The value that is passed is configured in the component editor. You can edit an operation and configure the parameter mapping in the Implementation section.



Name *
hostname

Display Name *
hostname

Description

✓ Required

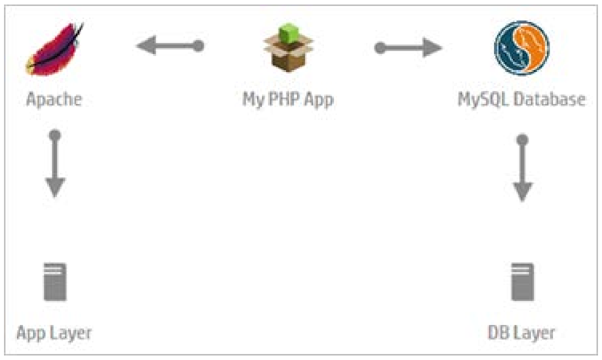
Mapping Type *
Relationship Target Property

Relationship
getDatabaseServerInfo

Value
hostname

Sometimes a value must be passed along with multiple relationships. For example, in a two-tier LAMP design, My PHP App needs to know the `ipAddress` of the application-layer server, so that My PHP App can be installed on the application layer. My PHP App also needs to know the `ipAddress` of the database-layer server, so that the application can connect to MySQL. In both cases, the `ipAddress` flows along two relationships.

Since My PHP_App is originally a Chef component, it has a target server with a direct, *`hostedOn`*, defined relationship.



The *My PApp* Chef component has a direct *hostedOn* defined relationship that targets the Server. The relationship can be used for `ipAddress` mapping. However, you must manually replace the *hostedOn* relationship with the *webserver* relationship that targets *Apache*.

A Chef component accepts the *Server* properties using the *hostedOn* relationship. After the component is provisioned, output property values are set including (original input) values for `nodeName` and `nodeIpAddress` properties.

For Chef components, the `nodeName` and `nodeIpAddress` properties can be passed along a chain of multiple related components where each consumes output from the preceding one. In symbolic notation this configuration looks like this:

```
MyPhpApp.dbIpAddress = MyPhpApp.database.nodeIpAddress = MySqlDb.hostedOn.ipAddress
```

When using OO components, a value has to be referenced using multiple relationships or *multi-hop* mapping:

```
MyPhpApp.databaseIpAddress = MyPhpApp.database.hostedOn.ipAddress
```

You cannot configure this type of parameter mapping using the UI editor, as only single relationships are supported. However, you can use the public REST-full API for topology-component management to configure multi-relationship mapping. Once configured, the mapping works well on the backend.

The example below shows part of the PUT request JSON body used for a component update, giving the value flows and the *webServer* and *hostedOn* relationships. The red text shows notation for the *multi-hop* mapping:

```
...
"mapping" : {
       "input" : {
           ...
         "nodeIPAddress" : "@PROPERTY:{urn\\:x-hp\\:2013\\:software\\:ccue\\:designer\\:topology-
metamodel\\:chef}webServer7638f7c8-a4c1-478d-aaad-3844def94428.hostedOn_410ebb81-93a0-4c2e-b9c7-
1f63b56b5ecf.ipAddress",
           ...
       },
       "output" : {
           ...
       }
```

CSA content sources

This section describes the various content sources for CSA.

These instructions are for developers who develop and import content into CSA.

Component sources

CSA topology designs include components from the following sources:

Component	Description
Out-of-the-box (OOTB) Content	Content that is present in CSA after installation (VMware vCenter, Amazon EC2, Helion OpenStack components).
Operations Orchestration (OO)	OO links automated tasks to flows, then CSA wraps the result of a flow execution as a component, which can be used as a building block for more complex designs. CSA uses OO in several ways: • As a process executor for sequenced designs. • As a process executor for topology designs, where the OO master-flow is the implementation of the design execution plan. • As a component import source, where new topology components can be imported from existing OO flows.
Server Automation (SA)	SA provides complete automated lifecycle-management for enterprise servers using software policies, providing a scalable, heterogeneous solution across physical and virtual servers (including VM templates). SA policies can be used in CSA for application deployment on infrastructure provisioned by VMware vCenter. The software policies are wrapped into CSA components, which can be used in a topology design. During fulfillment of the design, SA deploys the associated policies on the infrastructure. For more information about importing SA components, see the *Import Components* topic in the CSA Management Console online help.
Chef	Chef is an automation framework that deploys servers and applications to any physical, virtual, or cloud location. Chef operates with *cookbooks* consisting of *recipes*. CSA leverages Chef cookbooks for installation of software on top of infrastructure, provisioned by providers such as VMware vCenter or Amazon EC2. The Chef cookbooks can be wrapped into CSA components that can be used for a topology-design composition.

OO Content

This section describes standard and custom OO content.

CSA automatically imports OO flows as components, then forms complex topology designs.

CSA – OO Integration

CSA must be configured to communicate with a running OO server (and vice versa) so that its out-of-the-box components can integrate with OO as needed.

To configure OO for CSA, log into the OO Central web interface to create an admin user:

Username: **admin**

Password: **cloud**

Roles: **ADMINISTRATOR**

Standard content and non-standard content

Standard content is content that conforms to the naming and structure conventions detailed in this section. Non-standard OO content is called *custom content* and does not have to conform. In standard content, flows representing a component must be placed in the same folder, and their paths must be stored in the content pack or OO server. For example:

```
Library > Integrations > HPE > Cloud Service Automation > Components > $ProviderTypeName > $ComponentType >
$Version > @Flows
```

The master flow content-pack path (`<Install Dir>\jboss-as\standalone\tmp\e2e-flows`) is located in the `csa.properties` file and is configurable.

The values beginning with the dollar sign (**$**) are placeholders for the **provider type name**, the **component type**, and the **version.** The **@Flows** value represents a list of individual flows in the folder. Optionally, there can be a sub-flow folder to gather additional entities used in the flows.

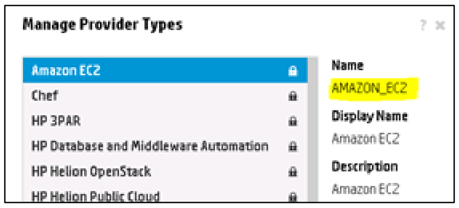
Provider types

CSA includes both custom-provider types and out-of-the-box (OOTB) provider types.

Custom provider-type names

The custom provider-type name must be:

- All uppercase with separating underscores, for example, CUSTOM_PROVIDER_TYPE.
- Used as the folder name for standard content.
- Available in the Provider Management area of the CSA Console, as the following figure shows:



Component type names

The component type name:

- Uses the placeholder `$ComponentType`.
- Is used as the folder name that appears in the flow path for the standard content (editable in the component).
- Should be a descriptive name, such as `OpenStackServer`, to differentiate the components and avoid naming conflicts.

Note: The provider-type parent folder does not affect the component name. In the following example, the component type name is `OpenStackServer`:

```
Library > Integrations > Hewlett-Packard > Cloud Service Automation > OPENSTACK -> Server ->
1.0
```

Versions

CSA allows you to import versioned standard content, but does not provide component version-management capabilities. A component is created with a version matching the leaf folder in the flow path.

Note: The component name and version provide a unique identification – you cannot import multiple components with the same name and version number, regardless of the related provider type.

Flows

Flows must be placed in the version folder.

Individual flow names:

- Are used automatically as an operation name in the new component.
- Is important for recognition of the lifecycle phase related to the operation.
- Must have an input property defined in the `LIFECYCLE_PHASE` constant, which consists of meta information not related to the flow logic, and marks the flow's purpose as it relates to the component lifecycle.

Note: Every OO flow that is used as an action in a component must have both success (✓) and failure (✗) outcome indicators.

The operation lifecycle phase has duplicate indicators, so the flow-name prefix and the `LIFECYCLE_PHASE` input-property value must be set as follows:

Flow	Flow-name prefix	LIFECYCLE_PHASE value
Deployment	Deploy or Create	`deploying`
Undeployment	Undeploy or Delete	`undeploying`
Modification	Modify	`modifying`
Undo a successful modification	Unmodify	`unmodifying`
Handle a failure during deployment	Deploy failure handler	`deploying_failure`
Handle a failure during undeployment	Undeploy failure handler	`undeploying_failure`
Handle a failure during modification	Modify Failure	`modifying_failure`
Custom public action executable on a deployed instance	Deployed	`deployed`

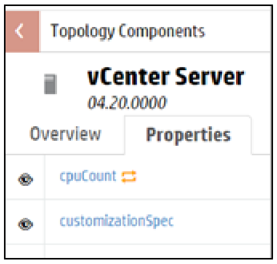
Input and output properties

The OO server automatically wraps all flows with a `result` output property. This property holds contextual messages, for example, detailed information about failure.

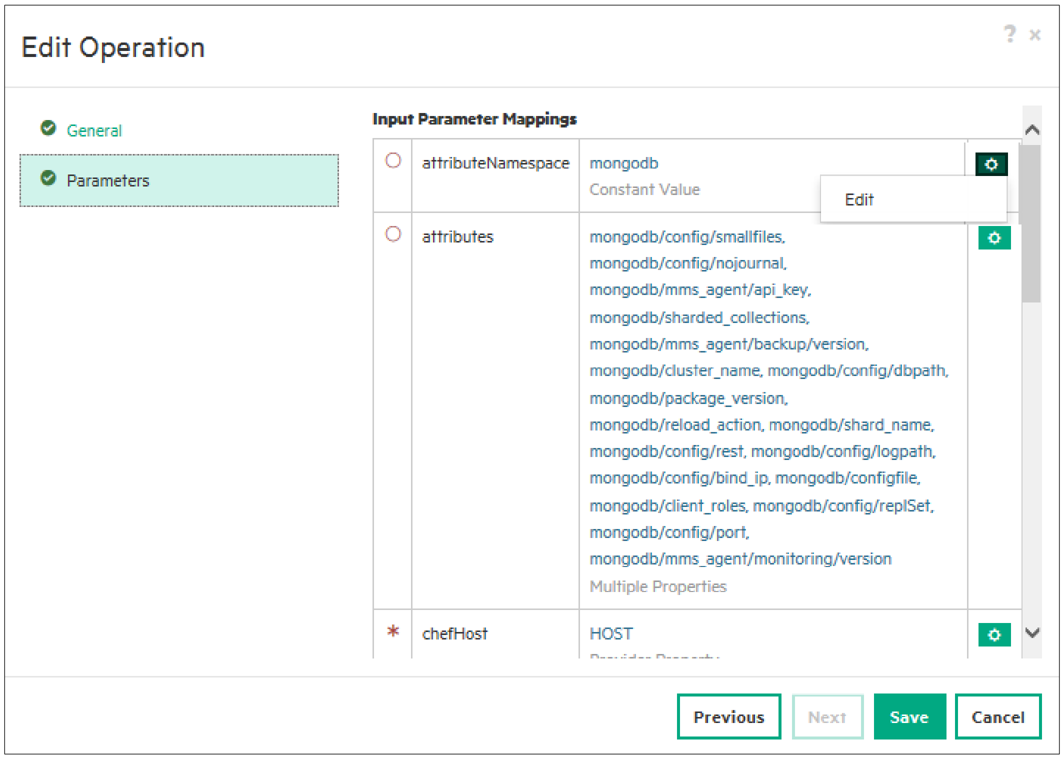
HPE suggests that you define flow outputs as explicitly as output properties.

Note: You no longer have to declare the output parameter **response** to your flow to indicate the outcome of an OO flow execution, but some of the OOTB components still use the property.

The Topology Components screen shows available properties and displays an icon (⇄) next to the property if the flow has requested previous values.



You can modify properties on an active subscription. CSA passes previous property values to modified OO flows by request, defining an additional flow input with a `prev_` prefix. For example, if the modification OO flow defines an input of `memorySize` and requires its previous value, you must also define an input of `prev_memorySize`.


Edit Operation

General

Parameters

Input Parameter Mappings

○	attributeNamespace	mongodb Constant Value	Edit
○	attributes	mongodb/config/smallfiles, mongodb/config/nojournal, mongodb/mms_agent/api_key, mongodb/sharded_collections, mongodb/mms_agent/backup/version, mongodb/cluster_name, mongodb/config/dbpath, mongodb/package_version, mongodb/reload_action, mongodb/shard_name, mongodb/config/rest, mongodb/config/logpath, mongodb/config/bind_ip, mongodb/configfile, mongodb/client_roles, mongodb/config/replSet, mongodb/config/port, mongodb/mms_agent/monitoring/version Multiple Properties	
*	chefHost	HOST	

Important: Only the un-prefixed flow inputs are displayed in the Properties field. The modification operation uses the `prev_` OO flow inputs and displays them in the Input Parameter Mappings section. Do ***not*** edit these inputs.

A component's **modify** flow places modification-status information, error codes, and other information in the `modifyReturnValue` output field. The **modify failure** flow receives this information in the `modifyFailureValue` input field, enabling the flow to correct errors in the **modify** flow intelligently. This sequence is particularly useful in complex, multi-step flows, where failures can occur at any step.

IMPORTANT: CSA automatically handles field mapping of fields that are already set on the OO flows. Do *not* modify these fields in the Component Editor.

Property names

You can create property names in input or output mode using topology components. Property names have the following characteristics:

- Any legal identifier.
- Maximum of 255 Unicode letters and digits.
- Begin with a character.
- Prohibited characters: _ (underscore) and . (dot) (using these characters can cause issues during property-value propagation from CSA to OO flows).

The OO content is imported into CSA and then encapsulated into components. These components can be used in topology designs, but require manual adjustment and fine tuning.

All CSA components have **properties**, **operations** and **relationships**.

Property values are the inputs that parameterize the component realization. Properties can also hold results from the realization that will be displayed or even used by other components.

There are two kinds of operations:

- Lifecycle operations, which are coupled with component deployment and undeployment.
- Custom operations, which can be exposed as public actions that a Subscriber can execute in the Marketplace Portal.

Relationships to other components can express both optional and required dependencies. During provisioning, a component can use the outputs of components upon which it is dependent.

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

See Import OO flows for more information.

Parameter Mapping

You can use predefined mapping or create a new mapping. Most of the components need input values for realization. These values come from user input through component properties, related components, or from the provider instance that is used for the realization.

To create a new parameter mapping to import components, see Create a new parameter mapping.

OO Import Limitations

The *standard* content import works seamlessly. *Custom* content (which may not follow the best practices for content) can be imported but has some limitations.

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. Unlike OO flows, OO operations cannot be imported as CSA 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.

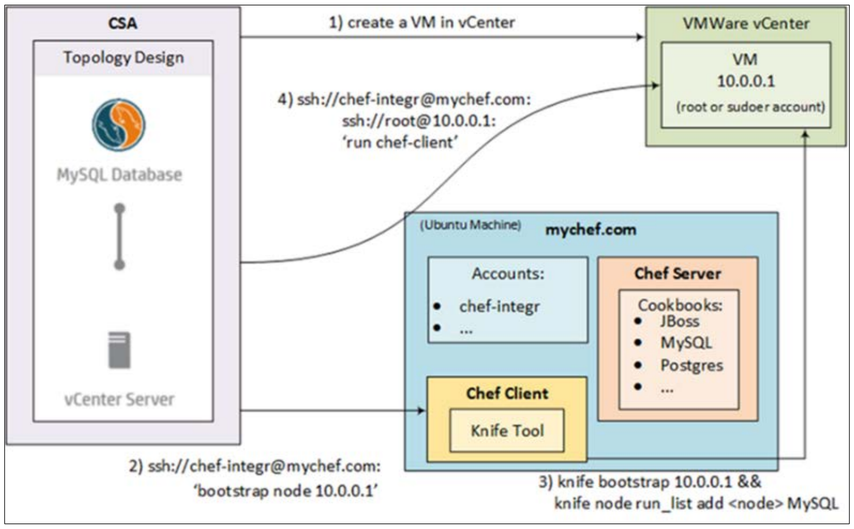
Chef content

Chef is an automation framework that deploys servers and applications to a physical, virtual, or cloud location. Chef contains **cookbooks**, which are folders that contain related **recipes** (installation scripts).

CSA uses Chef cookbooks to install software on top of infrastructure provisioned by providers such as VMware vCenter or Amazon EC2. Chef cookbooks are automatically encapsulated into CSA components, which can be used for topology-design composition. CSA communicates with Chef using public REST-full APIs, collects information about each cookbook selected for import, and creates a component based on the information collected.

You can define a design in the CSA Topology Designer. The Designer allows service developers to associate Chef components with the infrastructure components on which the Chef components will be provisioned.

The following figure shows CSA-Chef integration.



1. CSA provisions the server infrastructure used to host the Chef software.
2. CSA connects to the Chef host machine through SSH and invokes the knife command located there to bootstrap the infrastructure machine (such as vCenter VM). A user account, on the Chef host, must have access to the Chef configuration files, keys, and executables.
3. On the Chef host machine, the `knife bootstrap` and `knife node run_list` commands are invoked to put the infrastructure machine under Chef management and to select cookbooks to install the requested software.

4. CSA connects to the Chef host machine through SSH and opens another connection to the server created in step 1 to execute the `chef-client` script. The nested SSH call uses infrastructure-machine credentials in the topology design. CSA uses passwords and private keys for authentication. The `chef-client` script runs the installation script from the cookbook to install the software on the machine.

Cookbooks

Chef uses cookbooks that contain recipes, which are Ruby scripts automating the installation and configuration of software pieces, such as databases and application servers. During the import, Chef cookbooks are encapsulated into CSA components, which can be used for topology-design composition.

Imported Chef components are not bound to the original Chef instance in any way. So a design containing Chef components can be provisioned using any other compatible Chef server, as long as the required cookbooks are in place.

CSA communicates with the Chef server using public REST-full APIs, collects information about each cookbook selected for import, and creates a component based on the information collected. This information is presented to the user through the Component Import Wizard.

For more information on how to set up Chef to work with CSA, see Chef Setup.

Modification of Chef components

A Chef component is modifiable during the Modify Subscription operation, provided that the Chef component’s cookbook is idempotent (written in a manner that the same cookbook can be applied more than once). When a Chef component is embraced by importing a Chef cookbook, CSA assigns the default recipe to both the Modify lifecycle phase and the Deploying phase. Therefore, the same cookbook is used for both initial deployment and subsequent modification requests.

Unique lifecycle operations

CSA defines the following lifecycle operations:

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

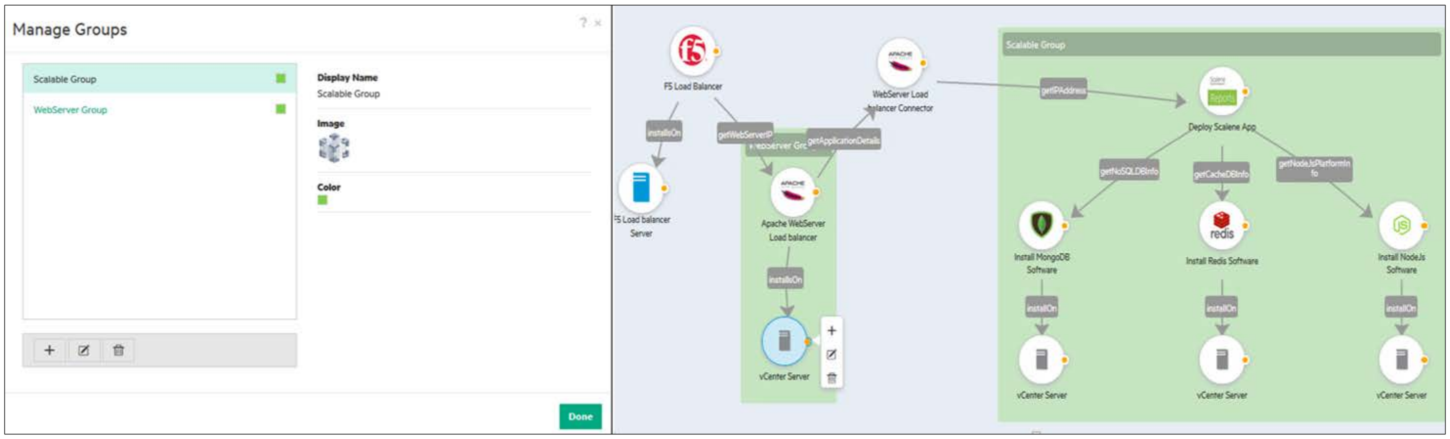
A component can have a single lifecycle operation of each type; for example, a single Deploy operation. So, when an operation is set to Deploy, any other operation that was previously configured as *Deploy* will have its lifecycle operation unset.

See Create a custom component, Add workflow, Configure input parameters, and Import properties for more information on component procedures.

Scalable groups

Use managed groups to add components within the topology design and to create scalable stacks of components during remediations when resources are scarce. For example, VMs can be added when CPU utilization increases past a defined threshold. The `Instance Count` group property indicates the number of instances of provisioned groups.

The following graphic shows how a unit of three identical components increases the number of provisioned instances of each component by scaling a stack up by 1, and increasing the `Instance Count` property value by 1.


Manage Groups

Scalable Group

WebServer Group

Display Name
Scalable Group

Image

Color

Done

Scalable groups do *not* support:

- Nested groups.
- Components belonging to more than one scalable group.
- Relationships between a scaling group and a component.

Capabilities

CSA topology components do not form a hierarchy (with the exception of Helion OpenStack components). Instead, CSA uses capabilities. A **capability** is a tag indicating what a component does. Capabilities can have properties and relationships.

CSA provides out-of-the-box capabilities (server, platform, database server, application server, and web server). The following table lists each type of capability and its properties:

Capability	Properties
Server	`ipAddress`, `hostname`, `username`, `password`, `privateKey`, and `instanceId`. **Note:** If a server component is used at design time, decide which authentication provisioning method should be used. For example, if you use `username` + `password`, by default CSA uses the `privateKey` field unless the field is empty.
Chef	`hostedOn`, which is automatically created after importing content, and is used to obtain server properties for the component deployment.

Lifecycle operations

Provisioned CSA components, including topology components, go through lifecycle states and operations.

Stages

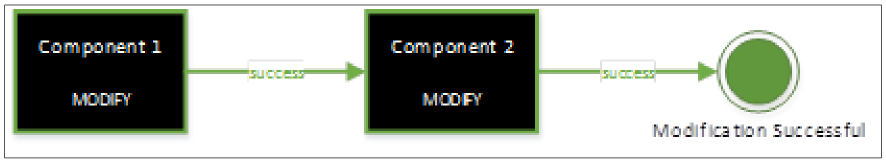
The following topology-component states are displayed in the CSA Operations Console (Management Console) and in the Marketplace Portal:

- Green boxes – stable success states
- Red boxes – stable failure states
- Faded gray and pink boxes – execution states (transition to the next state is triggered once the execution is finished)
- Yellow boxes – transitions from a stable state trigger when a component operation is explicitly invoked
- Other transitions automatically triggered.

Component modification scenarios

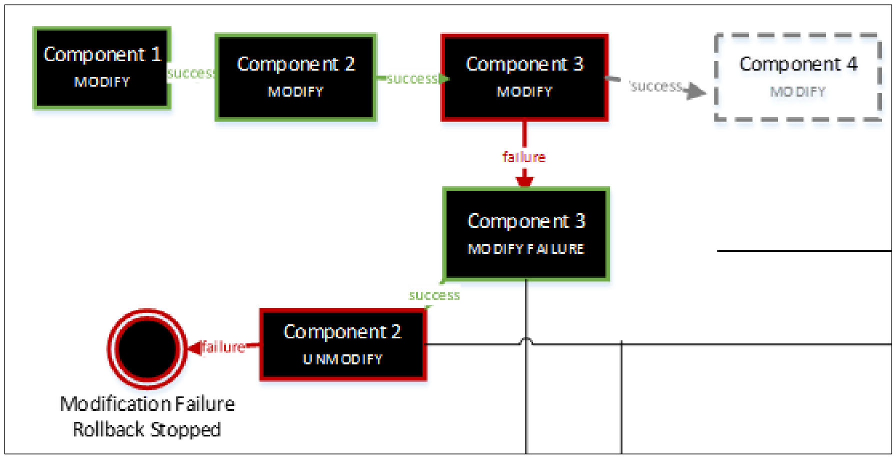
Scenario one: Successful modifications

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



Note: Typically, Modify component operations are successful.

Scenario two: Successful modifications followed by a failed modification



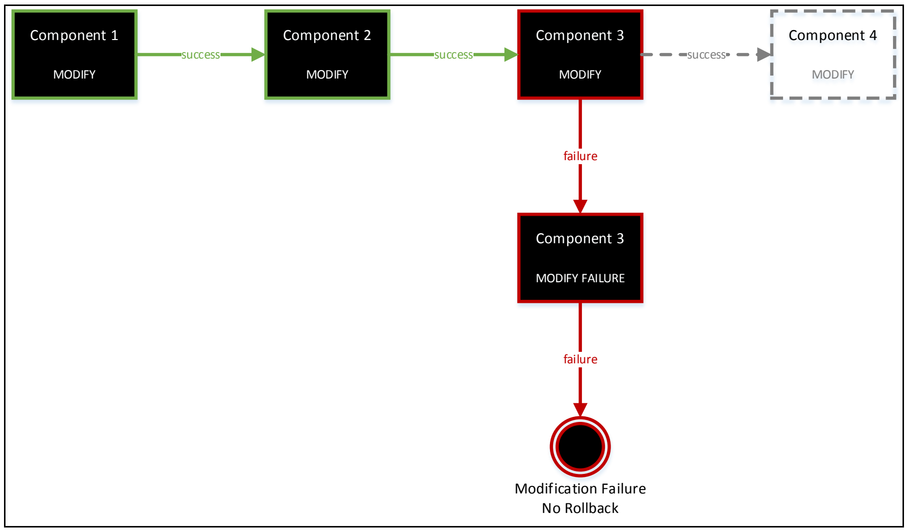
If the Modify operation for a particular component fails, its corresponding Modify Failure Handler is invoked. The Modify Failure Handler provides an opportunity for cleanup. The `modifyReturnValue` output from the Modify operation feeds into the `modifyFailureValue` input of the Modify Failure Handler. Information must to be communicated from the Modify operation to its corresponding Modify Failure Handler to provide additional contextual information about where the failure occurred. The nature of the failure is *modify* action so that the Modify Failure Handler can do intelligent cleanup.

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

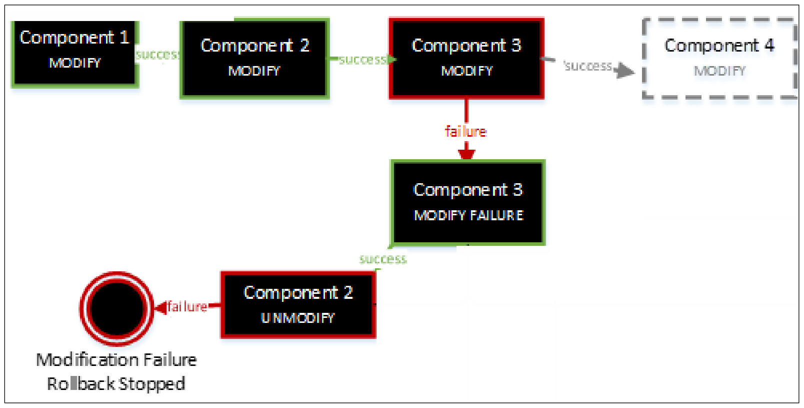
Additionally, the options selected when *Modify Subscription* was initiated are reverted if the rollback is successful. Subscribers are free to make different selections while resubmitting their previously failed modification request.

The Modify Failure Handler operation must always set the value of response parameter to failure. In the example below, the **failure** response indicates that the modification failed for Component 3. A successful Unmodify operation returns a response of **noop**, indicating that the rollback for that component was successful. A failed Unmodify operation returns a response of **failure**.

Failures may occur while executing the Modify Failure operation or Unmodify operations, as the following examples show.



A rollback is either not initiated or stopped as soon as a Modify Failure Handler or an Unmodify operation fails. In both these scenarios, the Subscriber may not change options that were previously selected for a subsequent resubmission. The Operator might have to intervene.



The following lifecycle operations can be defined on a component:

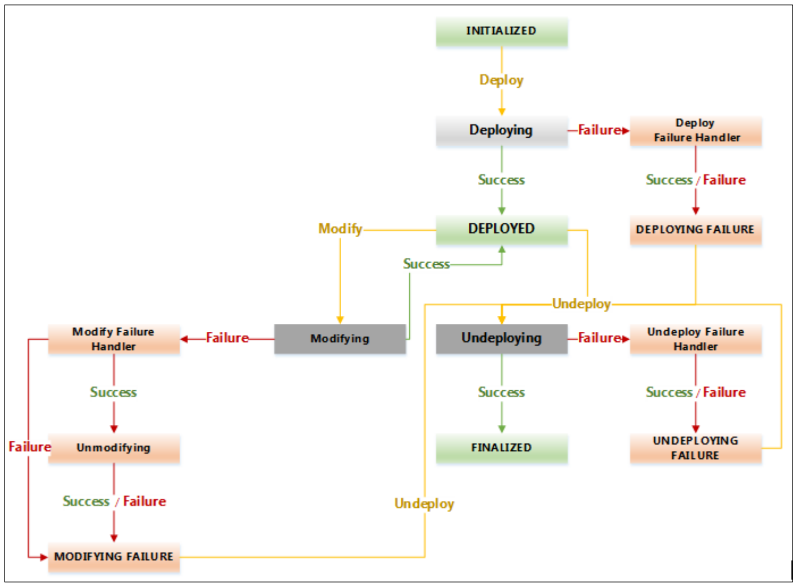
- **Deploy** – triggered when a new request is submitted.
- **Deploy Failure Handler** – triggered when a deploy request fails.
- **Undeploy** – triggered when an existing subscription is canceled.
- **Undeploy Failure Handler** – triggered when an undeploy operation fails.

Note: Failure to deploy or undeploy automatically invokes the corresponding failure handlers.

- **Modify** – triggered when a request to modify a previously deployed subscription is made. When the modification operation is successful, the component moves back to the Deployed state.
- **Modify Failure Handler** – triggered as a result of a modification failure, and operates on the component whose modify-action request failed.
- **Unmodify** – triggered after a Modify Failure Handler request. This operation rolls back all the operations executed so far in the order in which they occurred, achieves a clean rollback of the modification transition, and reverts the components to the state prior to the modification attempt. However, if the un-modification is not successful, the component's properties are retained and the Consumer can resubmit the last modification. Resubmitting the last modification might result in failure, so the Operator might have to intervene.

Note: During modification of a service instance, the modify action is invoked on *all* the components in the service instance, regardless of whether or not the modification request attempted to change component property values. However, the modify operation is passed with the previous value and the new value for the modifiable properties, so a Component Developer can decide whether any modification should be performed by comparing the two values (previous and new) and request a noop response, if no change was made to the component. Check whether the underlying resource currently exists before trying to modify or unmodify that resource.

The following graphic illustrates these topology operations:



Define a lifecycle operation

To define a lifecycle operation, edit the component, select an operation, and select the appropriate value from the dropdown menu.

Note: Lifecycle operations are automatically recognized if the component has been created using a Chef cookbook or standard OO content.

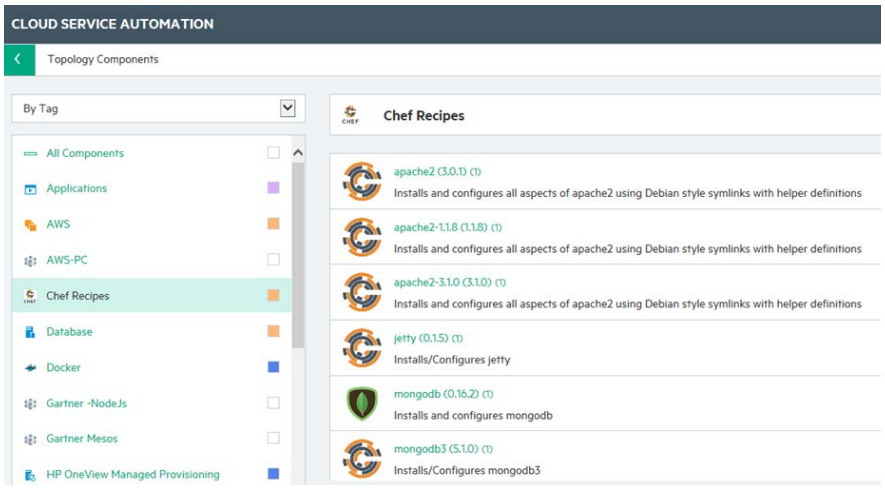
Topology-related procedures

This section contains procedures you should follow to set up CSA topology.

Import OO flows

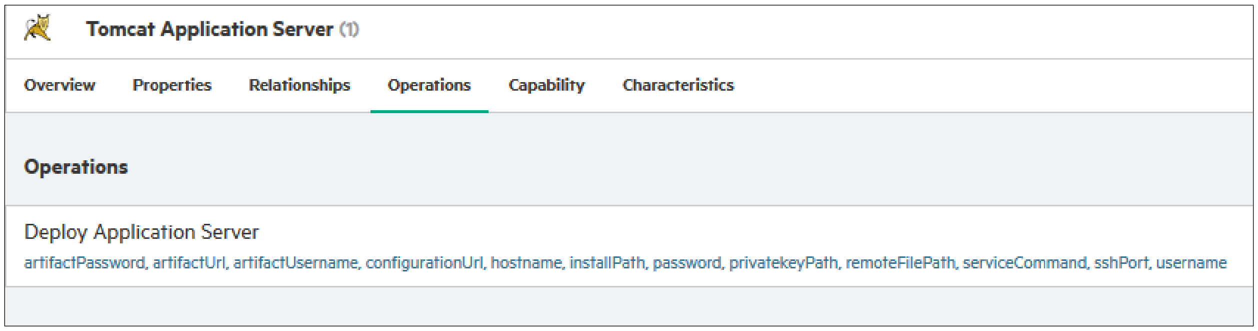
To import OO flows:

1. In the Cloud Service Management Console, navigate to **Designs** > **Topology** > **Components**.



apache2 (3.0.1) (1)
Installs and configures all aspects of apache2 using Debian style symlinks with helper definitions
apache2-1.1.8 (1.1.8) (1)
Installs and configures all aspects of apache2 using Debian style symlinks with helper definitions
apache2-3.1.0 (3.1.0) (1)
Installs and configures all aspects of apache2 using Debian style symlinks with helper definitions
jetty (0.1.5) (1)
Installs/Configures jetty
mongodb (0.16.2) (1)
Installs and configures mongodb
mongodb3 (5.1.0) (1)
Installs/Configures mongodb3

2. In the left panel:
   a. Choose to sort By Tag.
   b. In the All Components section, select the top-level component. In this example, Chef Recipes.
3. In the right panel, click the subcomponent that will be the flow recipient. In this example, *Tomcat Application Server*.

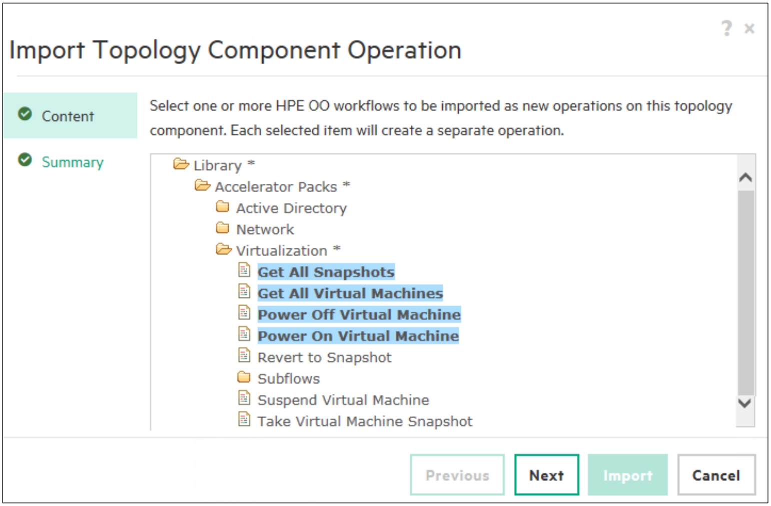

Tomcat Application Server (1)

Operations

Deploy Application Server

artifactPassword, artifactUrl, artifactUsername, configurationUrl, hostname, installPath, password, privatekeyPath, remoteFilePath, serviceCommand, sshPort, username

4. In the subcomponent screen, click **Operations**.
5. On the right side of the screen, click **Import**.
   The Import Topology Component Operation screen is displayed.


Import Topology Component Operation

Select one or more HPE OO workflows to be imported as new operations on this topology component. Each selected item will create a separate operation.

- Library *
  - Accelerator Packs *
    - Active Directory
    - Network
    - Virtualization *
      - Get All Snapshots
      - Get All Virtual Machines
      - Power Off Virtual Machine
      - Power On Virtual Machine
      - Revert to Snapshot
      - Subflows
      - Suspend Virtual Machine
      - Take Virtual Machine Snapshot

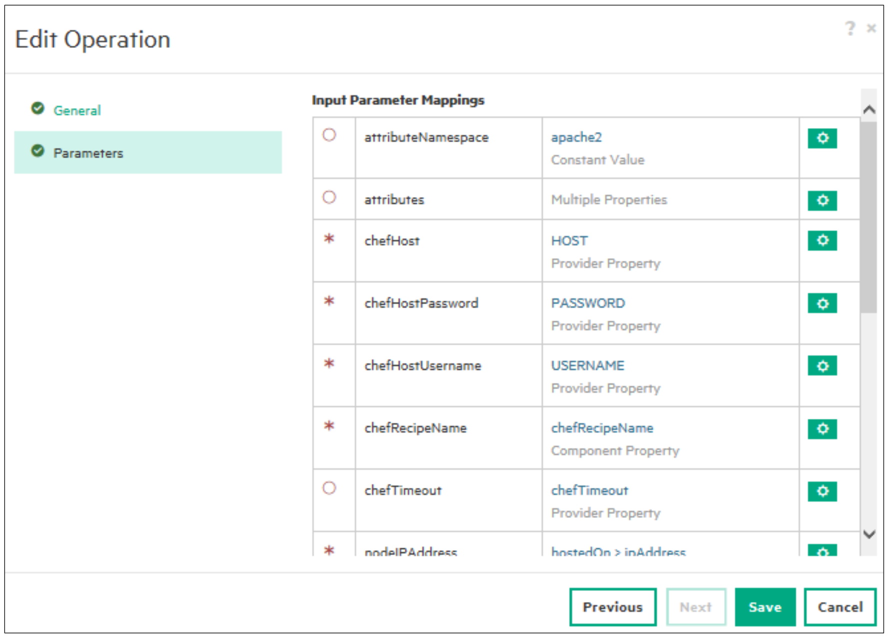
6. Select the flows to import and click **Next**.
7. Make sure the chosen flows appear in the list.
8. To add more flows, click **Previous**.
9. To accept the flows, click **Import**. The flows appear on the screen.
10. To add more flows, click **Import** again.

You can define additional content operations using the **Import** button.

Create a new parameter mapping

To create a new parameter mapping for the purpose of component import:

1. In the Cloud Service Management Console, navigate to **Designs** > **Topology** > **Components**.
2. Select a component.
3. Click **Operations**.
4. Click your operations Action Menu (⚙), and choose **Edit.**

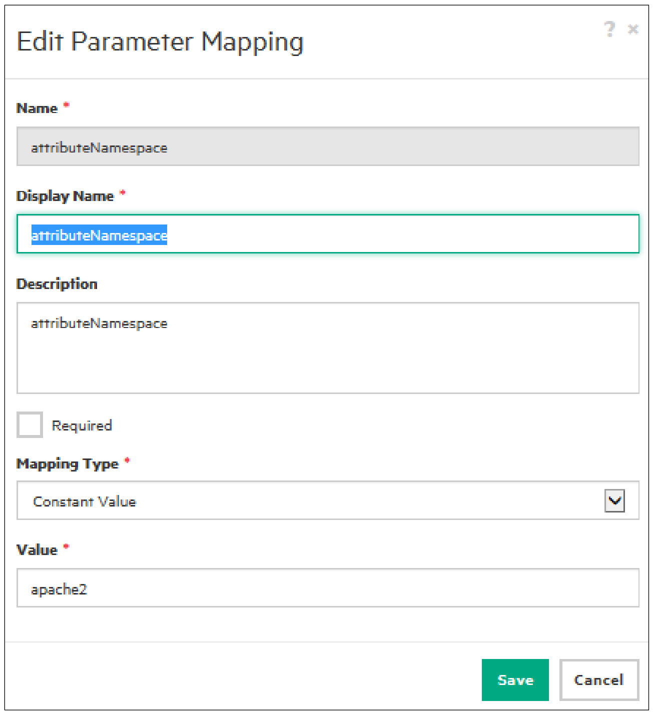


Edit Operation

Input Parameter Mappings

○	attributeNamespace	apache2 Constant Value	⚙
○	attributes	Multiple Properties	⚙
*	chefHost	HOST Provider Property	⚙
*	chefHostPassword	PASSWORD Provider Property	⚙
*	chefHostUsername	USERNAME Provider Property	⚙
*	chefRecipeName	chefRecipeName Component Property	⚙
○	chefTimeout	chefTimeout Provider Property	⚙
*	nodeIPAddress	hostedOn > ipAddress	⚙

5. In the Edit Operations screen, select Parameters.
6. In the Input Parameters Mapping section, identify the provider related to the provider instance.
7. In the parameter Action Menu, choose **Edit**.



Edit Parameter Mapping

Name *: attributeNamespace

Display Name *: attributeNamespace

Description: attributeNamespace

☐ Required

Mapping Type *: Constant Value

Value *: apache2

8. In the Edit Parameter Mapping screen, add mapping entries corresponding to a map of a component property to a provider property.
   For example, add the Default VMware vCenter Parameter Mapping maps-provider properties to flow input properties as follows:

Flow Input Parameter	Provider Property
`datacenterName`	`DATACENTERNAME`
`vCenterHost`	`HOST`
`vCenterPassword`	`PASSWORD`
`vCenterPort`	`PORT`

vCenterProtocol	PROTOCOL
vCenterURI	URI
vCenterUsername	USERNAME

Note that provider properties come from the provider service-access point information configured during resource-provider configuration (for example, **HOST**, **PASSWORD**, or **URI**). Additional properties (for example, **DATACENTERNAME**) must be manually defined as a property on the resource provider using the **Properties** tab in the Resource Provider configuration area. In the case of *standard content* import, one or more components are created and all use the selected parameter mapping.

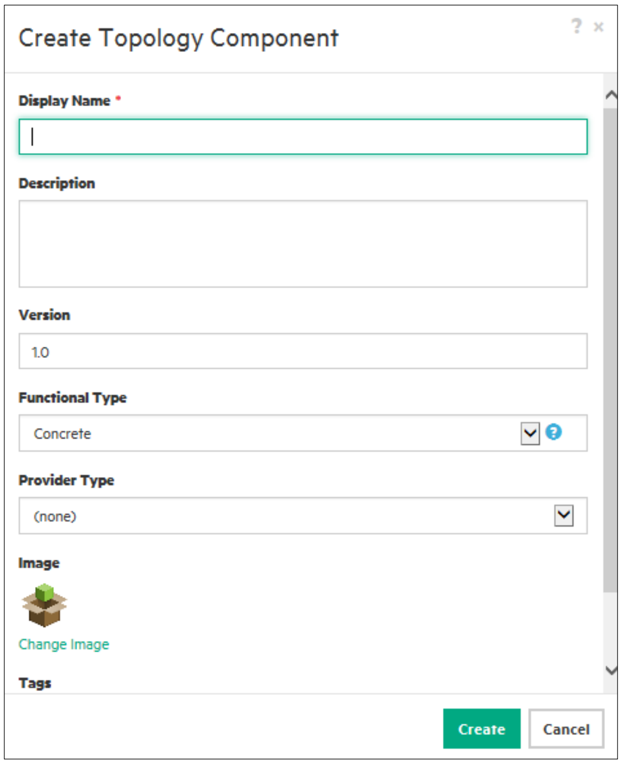
9. Click **Finish** to create your new component.
   Depending on whether the component has been created using *standard* or *custom* content, a different level of configuration will been done automatically. Typically, a few more adjustments are required to make the component fully operational.

Create a custom component

Custom components import workflow that does not follow standards. The component can be associated with out-of-the-box or custom resource providers.

To create a custom component:

1. In the Cloud Service Management Console, navigate to **Designs** > **Topology** > **Components**.
2. In the Topology Components screen, click the Action Menu ⚙ and choose Create Component.



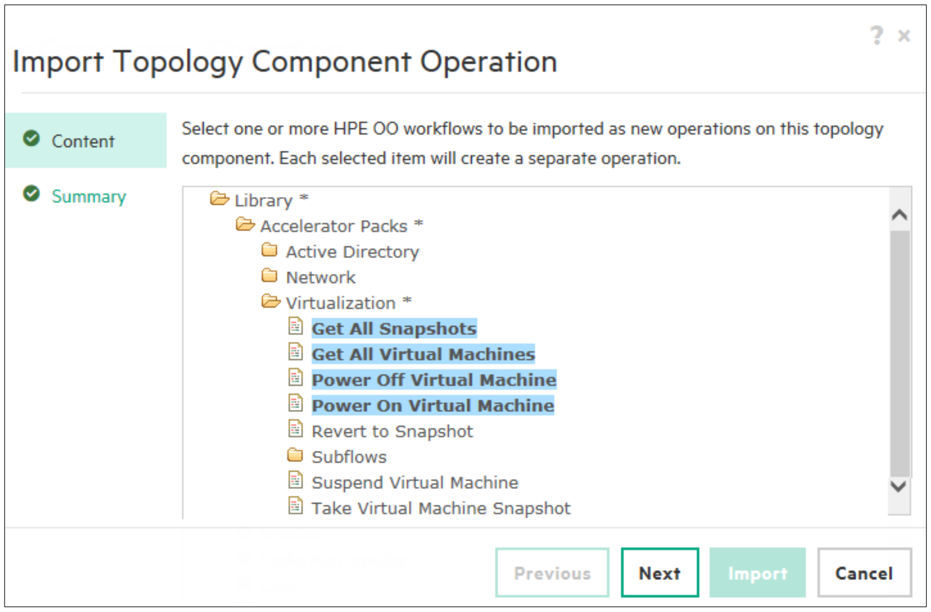
3. In the Create Topology Component dialog box, enter a:
   a. Name.
   b. Description.
   c. Version.
   d. Functional type:
      i. Concrete
         If you make this choice, choose a provider type from the list.
      ii. Capability
   e. Provider type.
4. Click **Create**.
   Note: The properties are imported automatically when the workflow is imported.

Add workflow

To add workflow to a new component:

1. In the Cloud Service Management Console, navigate to **Designs** > **Topology** > **Components**.
2. In the components section, select your new component.
3. Click the **Operations** tab.
4. On the right side, click **Import**.

   The Import Topology Component Operation screen is displayed.


Import Topology Component Operation

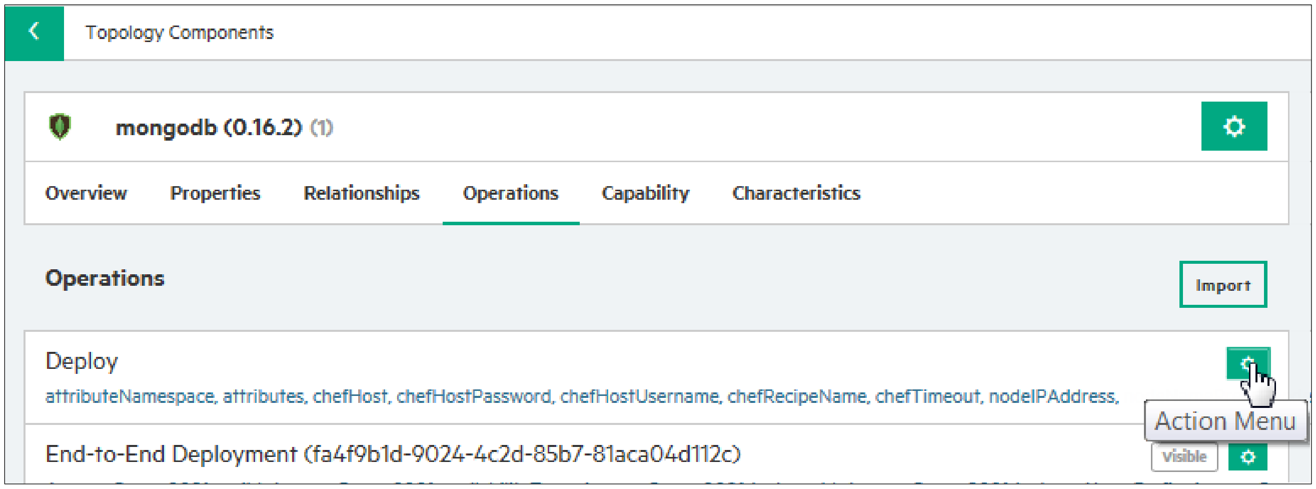
Select one or more HPE OO workflows to be imported as new operations on this topology component. Each selected item will create a separate operation.

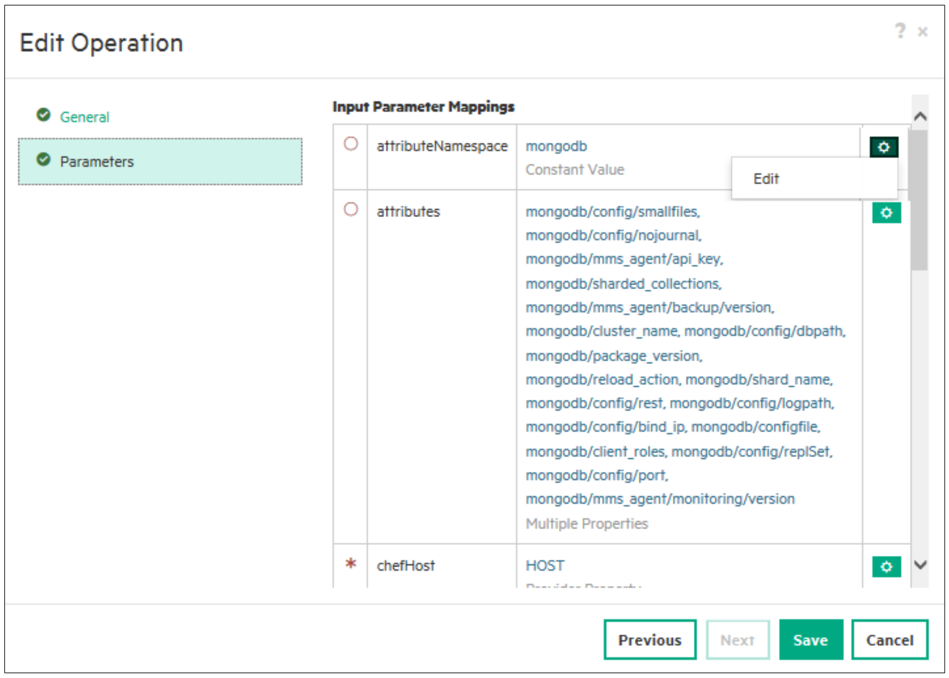
5. Choose workflows to import to your new component.
6. Click **Next**.
7. Click **Import**.

Configure input parameters

To configure input parameters for your new component:

1. In the Cloud Service Management Console, navigate to **Designs** > **Topology** > **Components**.
2. Click the **Operations** tab.
3. Choose an operation, then choose **Edit** from its **Action Menu**.


Topology Components
mongodb (0.16.2) (1)
Overview
Properties
Relationships
Operations
Capability
Characteristics
Operations
Import
Deploy
attributeNamespace, attributes, chefHost, chefHostPassword, chefHostUsername, chefRecipeName, chefTimeout, nodeIPAddress,
Action Menu
End-to-End Deployment (fa4f9b1d-9024-4c2d-85b7-81aca04d112c)
visible


Edit Operation

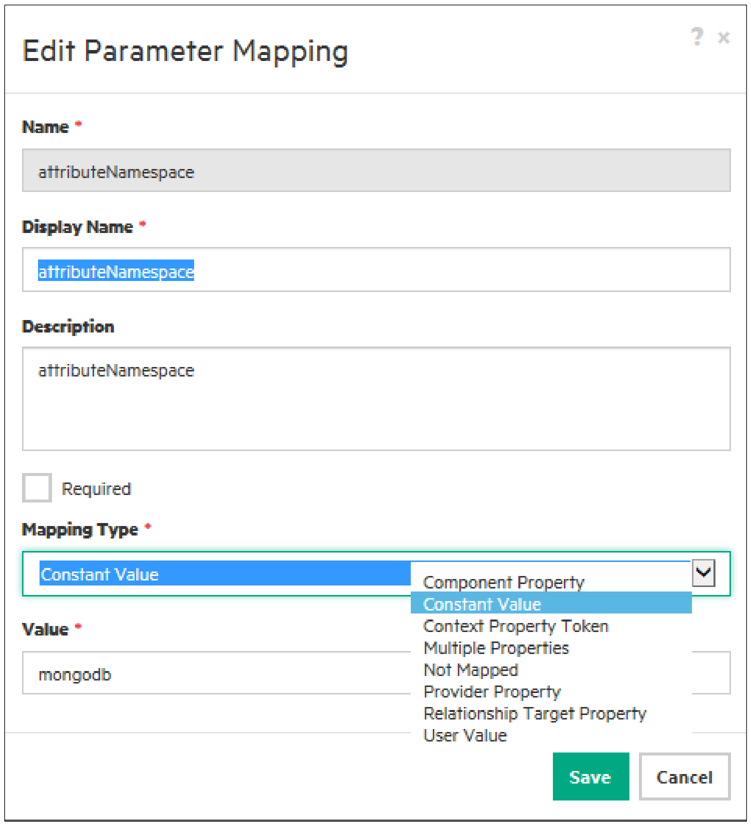
General

Parameters

Input Parameter Mappings

○	attributeNamespace	mongodb Constant Value	Edit
○	attributes	mongodb/config/smallfiles, mongodb/config/nojournal, mongodb/mms_agent/api_key, mongodb/sharded_collections, mongodb/mms_agent/backup/version, mongodb/cluster_name, mongodb/config/dbpath, mongodb/package_version, mongodb/reload_action, mongodb/shard_name, mongodb/config/rest, mongodb/config/logpath, mongodb/config/bind_ip, mongodb/configfile, mongodb/client_roles, mongodb/config/replSet, mongodb/config/port, mongodb/mms_agent/monitoring/version Multiple Properties	
*	chefHost	HOST	

4. In the Edit Operations screen, select **Parameters**.
5. To add a parameter, click its **Action Menu** and choose **Edit**. The Edit Parameter Mapping screen is displayed.


Edit Parameter Mapping

Name *
attributeNamespace

Display Name *
attributeNamespace

Description
attributeNamespace

☐ Required

Mapping Type *
Constant Value

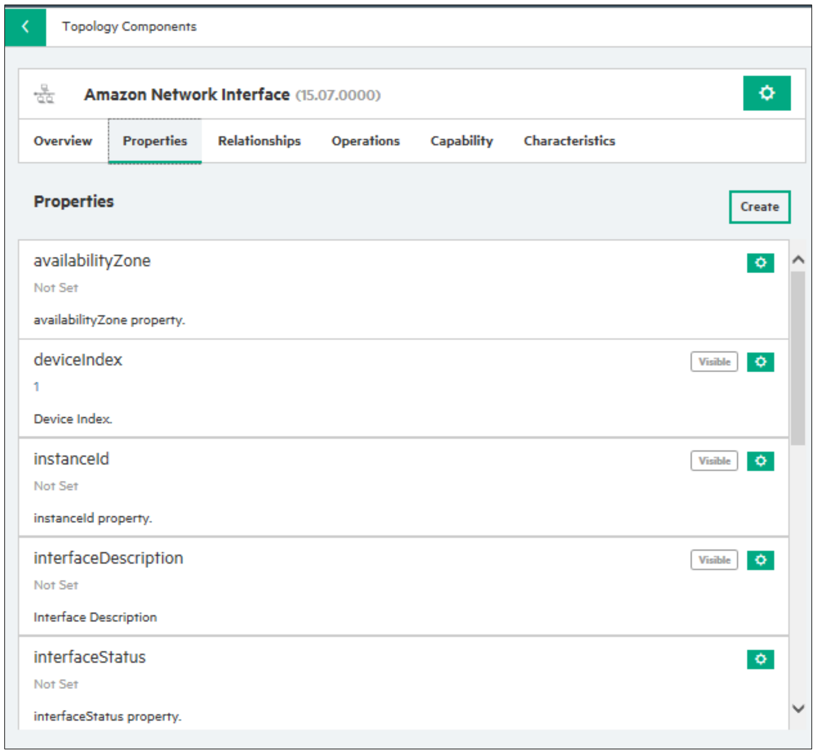
Value *
mongodb

6. If the parameter is required, check the **Required** box.
7. Choose a mapping type. See the CSA online help for more information about mapping types.
8. Click **Save**.
   Note: The values for the resource provider properties **HOST**, **PASSWORD**, **PORT**, **PROTOCOL**, and **USERNAME** are common to all resource providers, and are retrieved and set at run time.

Import properties

This example shows how to import for OO properties.

1. In the Topology Components screen, click a component.
2. Select the Properties tab.



Properties

availabilityZone
Not Set
availabilityZone property.

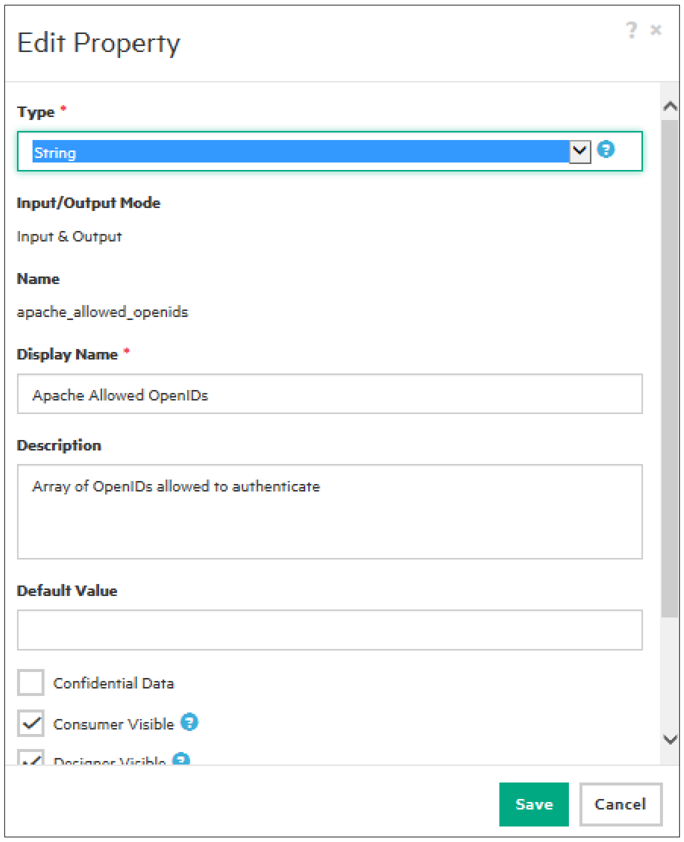
deviceIndex
1
Device Index.

instanceId
Not Set
instanceId property.

interfaceDescription
Not Set
Interface Description

interfaceStatus
Not Set
interfaceStatus property.

3. Choose a property, then select **Edit** from its Action Menu.



Edit Property

Type *
String

Input/Output Mode
Input & Output

Name
apache_allowed_openids

Display Name *
Apache Allowed OpenIDs

Description
Array of OpenIDs allowed to authenticate

Default Value

☐ Confidential Data
☑ Consumer Visible
☑ Designer Visible

4. In the Edit Properties screen, set the properties for your register service.
5. Click **Save**.

Chef setup

This section describes how to install and configure Chef to work with CSA.

Prerequisites

You must comply with a number of prerequisites before you install and configure Chef.

Note: For the most recent compatibility information, see the *CSA Support Matrix*.

1. Make sure you comply with supported configurations:

Software	Supported versions	Notes
CSA Server OS	Windows 2008 and 2012	See the CSA 4.2 Support Matrix
CSA Version	CSA 4.2	See the CSA 4.2 Installation Guide
Chef server OS	Ubuntu 12.04 LTS	See Install the Chef server for additional software requirements
Chef server software	Open-source Chef server, Version 11.01	Hosted and Enterprise Chef are not supported
Nodes provisioned by CSA and used with Chef	Ubuntu and RHEL/CentOS 6.X	Add SSH (by default, SSH is not installed on Ubuntu)

2. Install an Ubuntu Server 12.04 or later server.
3. Configure the Ubuntu server as follows:
   a. Fully-qualified domain name that can be recognized by the domain-name system (DNS).
   b. Same NTP time reference and time zone as the CSA server.
   c. Network connection to CSA-provisioned machines to install required software.
4. Install the following utilities:
   a. `apt-get install sshpass`
   b. `apt-get install netcat`
5. Create a *required* user account (**chef-integr**).
   CSA uses this account to:
   a. Initiate `ssh` connections.
   b. Access the `knife` tool and the tool's configuration file, key, and executables.
   c. Connect to the Chef server public API to list cookbooks.
   Note: User does *not* need to be **root** or **sudoer**.

Install the Chef server

IMPORTANT: Install the Chef client and Chef server on the same Ubuntu server.

To install the Chef server on the Ubuntu server:

1. Access the open-source Chef installation: http://getchef.com/chef/install/
2. Install the downloaded package:
```
sudo dpkg -i chef-server*.deb
```
3. Configure the Chef server:
```
sudo chef-server-ctl reconfigure
```
4. Check Chef server status:
```
chef-server-ctl status
```
5. Connect to the Chef server online:
   a. Access the server:
```
https://<chefservername>
```
   b. Use these initial credentials: **admin / p@ssw0rd1**.
   c. Enter a new password in the Message field.

IMPORTANT: Do *not* create a new user on the Chef server at this point in the installation.

Install Chef client

1. Access the Chef client-installation package: http://www.getchef.com/chef/install.
2. Install the package:
   ```
   sudo dpkg -i chef_*.deb
   ```
3. Configure knife:
   ```
   sudo knife configure –i
   ```
4. In the screens that follow, accept default values *except* in the following:
   a. Enter Chef server URL (FQDN): https://mychefserver.com
   b. Choose any new user name and password for the `chefuser` account. A private key for the new user is automatically generated.
   Note: You will need to use these credentials again.
5. Check that the new user appears in the Chef-server web interface, in the Users tab.
6. The **configure** command creates a `.chef/knife.rb` file containing all configuration. Change the `.chef` owner to the system user that is the integration user in CSA. Grant the knife `tool` executable rights to this user.
7. To add a proxy configuration (optional), open the `.chef/knife.rb` file and add the following line anywhere in the file:
   ```
   bootstrap_proxy  'http://your.proxy:8080'
   ```

Import and upload Chef cookbooks

To install common cookbooks:

1. Download the cookbook from the Chef site (usually a `*.tar.gz` file):
   ```
   knife cookbook site download apache2
   ```
2. Download other cookbooks that have a mutual dependency on your downloaded cookbook.
   If you skip this step, you might encounter errors. If that occurs, download the missing cookbook and upload it using the `knife` command.
   To upload the cookbook to a server, use the following command: `knife cookbook upload apache2 -o cookbooks`
3. Create subdirectory cookbooks.
4. Unzip the downloaded file into a subdirectory cookbook file.

Create a vCenter template

This section explains the process of creating and configuring a VMware vCenter template for use in a Chef environment. If you use Amazon for server provisioning, you must rely on templates provided by Amazon (however, configuration often works seamlessly in an Amazon environment). Chef cookbooks will be installed on server components that are provisioned by CSA using VM templates.

To create a vCenter template:

1. Create a standard VM machine in the vCenter client wizard.
2. Install the OS (recommended OSs are CentOS and Ubuntu) on vCenter or AWS.
3. Set up the proxy:
   a. If the machine is behind a proxy, add these lines to the `.bashrc` file in the `root` home directory:
   ```
   export ftp_proxy=http://your.proxy:8080/
   export http_proxy=http:// your.proxy:8080/
   export https_proxy=http:// your.proxy:8080/
   export no_proxy=localhost,127.0.0.1
   ```
   b. In the file `/etc/yum.conf` add this line to set up the proxy for yum:
   ```
   proxy=http://your.proxy:8080
   ```
4. Install perl and wget:
   ```
   Yum install perl
   Yum install wget
   ```
5. Set up the proxy for `wget` in the file `/etc/wgetrc`:
   ```
   https_proxy = http://your.proxy:8080/
   http_proxy = http://your.proxy:8080/
   ftp_proxy = http://your.proxy:8080/
   ```
6. Disable or configure the firewall:
   ```
   /etc/init.d/iptables save
   /etc/init.d/iptables stop
   chkconfig iptables off
   ```

7. Shutdown the VM.
8. Right-click the VM and select **Template** > **Convert to Template**.
9. Click Save.

You can also set proxies, and install and configure `yum/apt-get` and `wget`.

Note: Currently, Chef components on virtual machines are only supported on Linux templates. Theoretically, any Linux template should work if it provides the required tools and satisfies cookbook requirements.

Recipes in a cookbook

When a new component is created from a cookbook, CSA always uses the default recipe as the component implementation.

To use a non-default recipe, open the component's `Deploy` operation and change the value of the `chefRecipeName` parameter. The value can be any valid run-list expression `cookbook[@version][::recipe]`.

For example:

- `mysql`
- `mysql::client`
- `mysql@4.0.20`
- `mysql@4.0.20::client`

Different recipes from the same cookbook or multiple versions of one cookbook can coexist in CSA as separate components. Clone the component first, then change the recipe values for individual copies.

CSA will be unable to recognize how a cookbook's recipes and attributes should be used unless they are properly defined in the Chef cookbook's `metadata.rb` file.

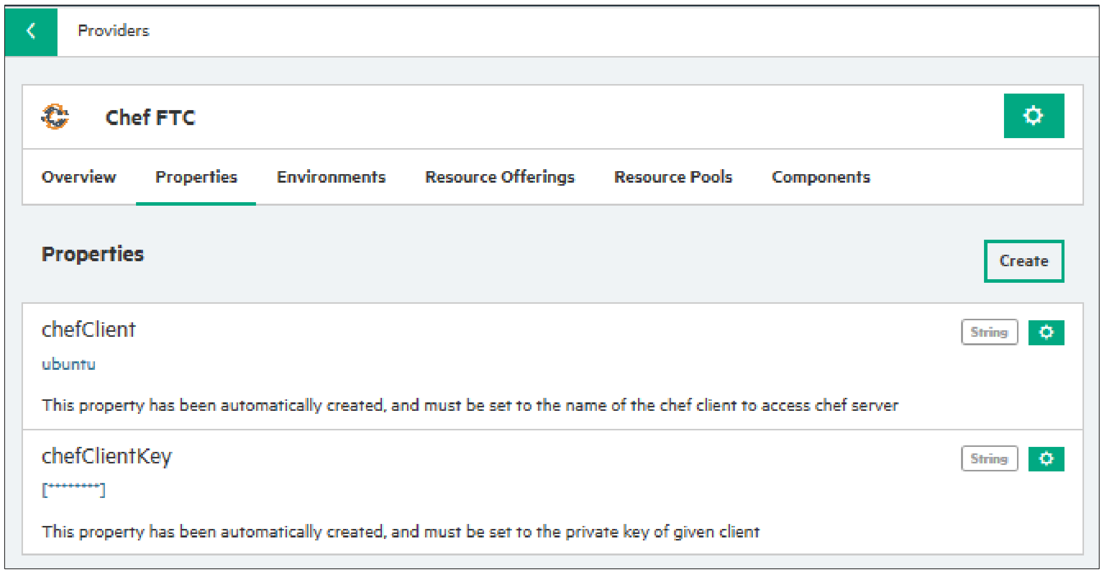
Customization Specification

Create a customization specification in the vCenter client, and specify the network setup, hostname convention, IP address, and name of the provisioned VMs. This configuration ensures that machines you create can be accessed from CSA and Chef servers. For more configuration and customization information, see VMware documentation.

The Chef provider in CSA

The Chef server must be registered as a resource provider in CSA.

1. In the Cloud Service Management Console, click Providers.
2. Click Chef FTC (for example).
3. Click the Properties tab.

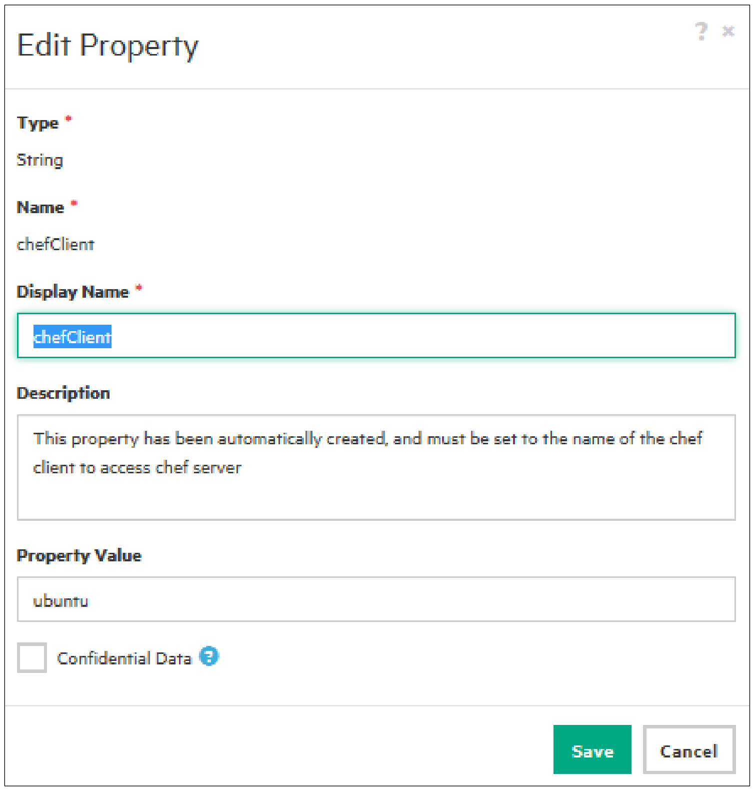


Properties

chefClient
ubuntu
This property has been automatically created, and must be set to the name of the chef client to access chef server

chefClientKey
[********]
This property has been automatically created, and must be set to the private key of given client

4. Click `chefClient` (client/user name configured on the Chef server, and used for access to the Chef API), then choose **Edit** from its Action Menu.


Edit Property

Type *
String

Name *
chefClient

Display Name *
chefClient

Description
This property has been automatically created, and must be set to the name of the chef client to access chef server

Property Value
ubuntu

☐ Confidential Data

5. In the Edit Resource Provider screen enter a:
   a. Display name.
   b. Description.
   c. Service Access Point (Chef-server end point (default port is 443), for example: `https://<hostname>:[port]`)
   d. User ID (the system-user account that exists on the Chef server machine (for example, `chef-integr`), and that connects to the machine using SSH.
   e. Password (corresponding to the User ID).
6. Click **Save**.
7. Click `chefClientKey` (private key associated with the `chefClient`), then choose **Edit** from its Action Menu.
8. Repeat steps 4 and 5.

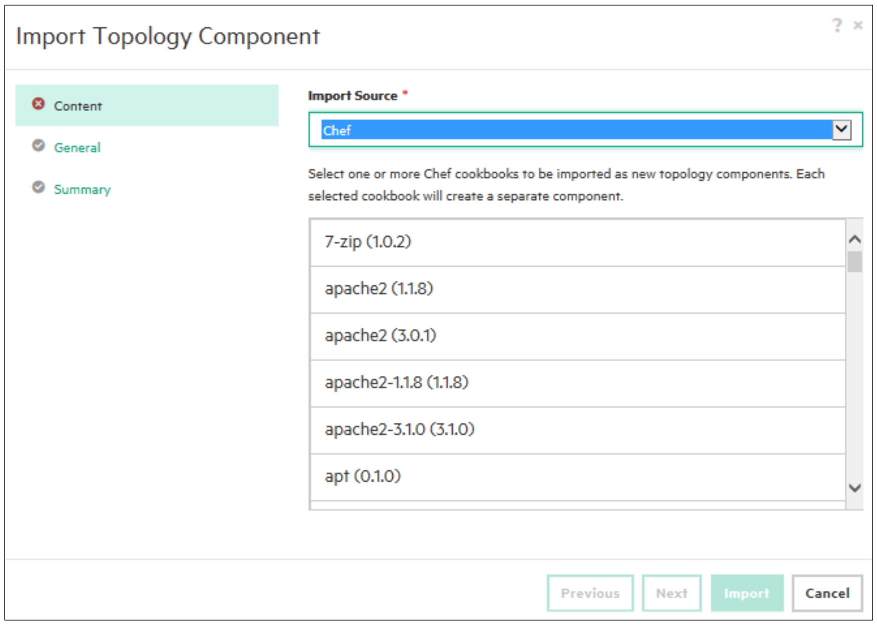
Note: The client name and key values are collected when a new user is created on the server. However, you can substitute the user created during Chef client configuration, or you can manually create a new user using the Chef-server UI.

Content Source

You can load Chef cookbooks from the Chef server, and configure Chef-server instances as resource providers in the Import Components screen.

To access the Import Components screen:

1. Navigate to **Designs** > **Topology** > **Components** in the CSA Console.
2. Click Chef from the left panel.
3. Click the Properties tab.
4. Choose Import Component from the Chef Action Menu.


Import Topology Component

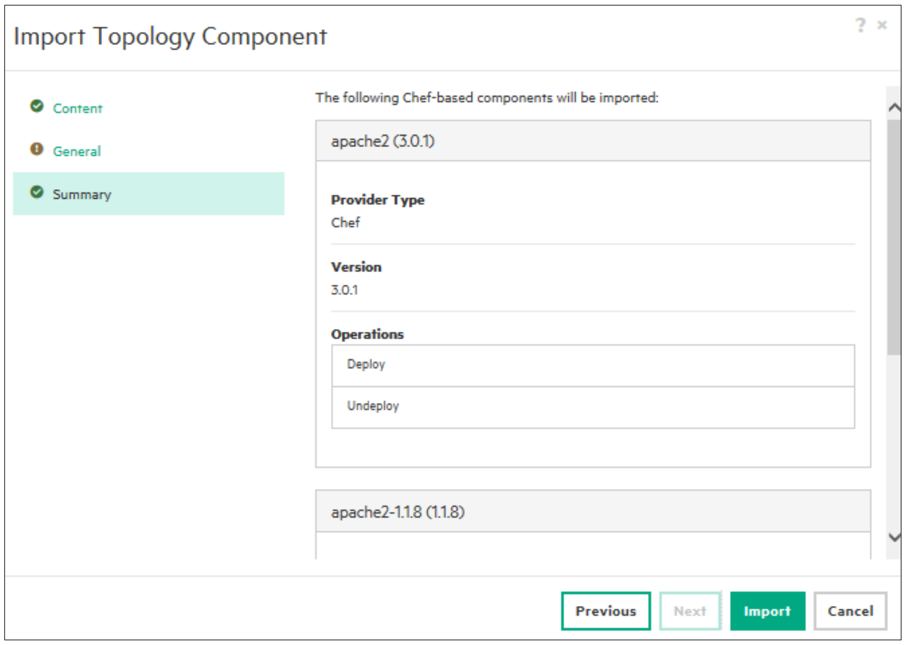
Import Source *

Chef

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

- 7-zip (1.0.2)
- apache2 (1.1.8)
- apache2 (3.0.1)
- apache2-1.1.8 (1.1.8)
- apache2-3.1.0 (3.1.0)
- apt (0.1.0)

5. In the Import Topology Component screen, choose an Import Source.
6. Select a Chef cookbook to import, then click **Next**.
7. Specify an image and tag (optional).
8. Click **Next**.


Import Topology Component

The following Chef-based components will be imported:

apache2 (3.0.1)

Provider Type
Chef

Version
3.0.1

Operations
- Deploy
- Undeploy

apache2-1.1.8 (1.1.8)

9. Review the Summary screen. To make changes, click **Previous**.
10. To import the components, click Import.

A cookbook version is displayed in parentheses after the name. The cookbook version is used as part of the new component name, so that multiple cookbook versions can be imported as different components. The cookbook version is not translated into the component version. The component version is automatically set to **version 1** and auto-incremented when the cookbook is imported repeatedly.

If you configure multiple Chef-resource providers, one must be selected.

Creating Topology Designs with CloudSlang Content

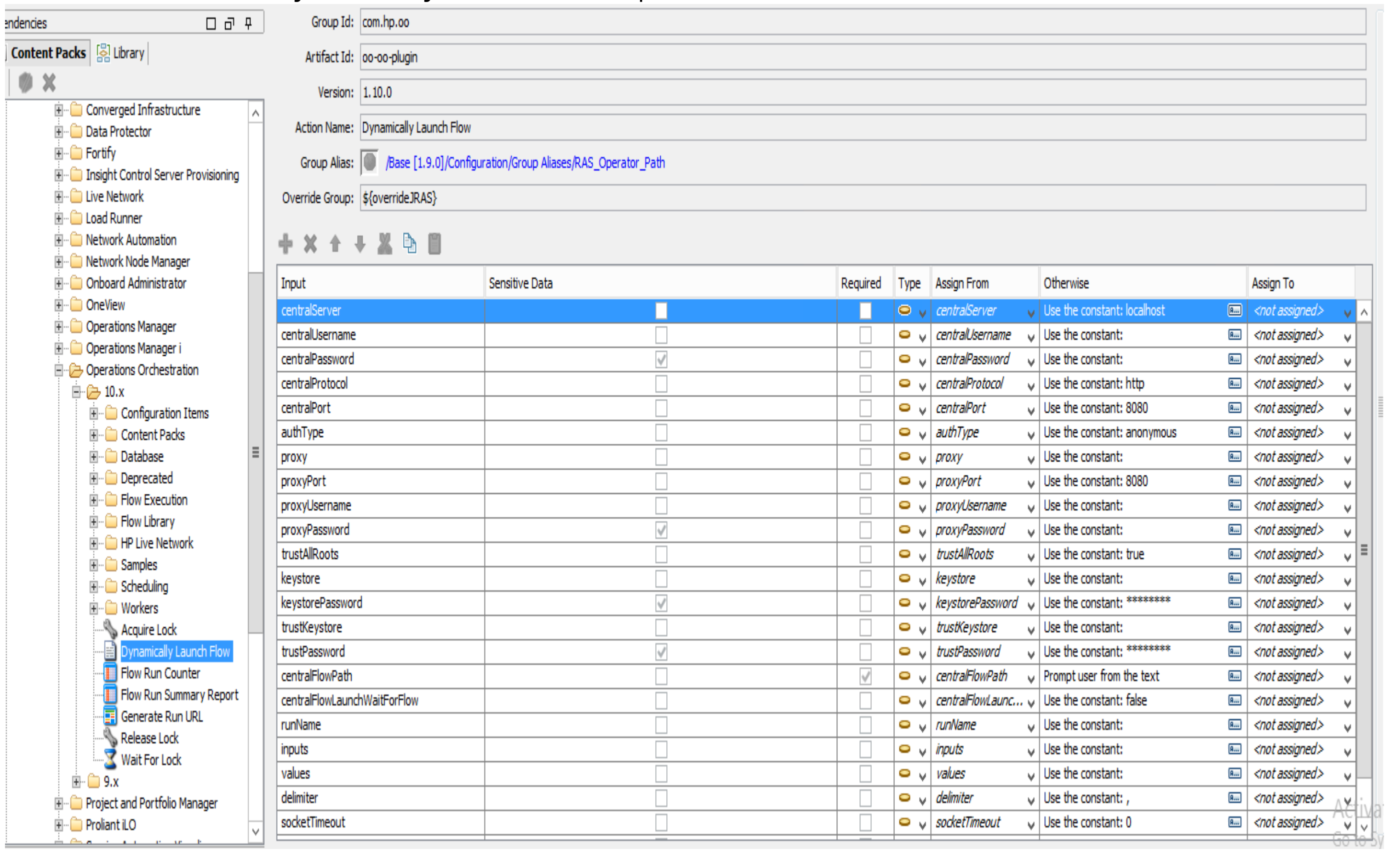
Currently there is no support for hybrid Cloud Slang-AFL flows in any combination.

The following procedure provides information on creating topology designs with CloudSlang content.

Pre-requisites:

Ensure cloudslang content is available in OO studio.

1. Ensure to get the Cloudslang flow path from the web designer studio that is required to be used as a component in CSA along with the inputs and values for the same.
2. Navigate to the OO studio and import the **HPE Solutions Content Pack**.
3. Create a workflow using "`/HPE Solutions [1.10.0]/Library/Integrations/Hewlett-Packard/Operations Orchestration/10.x/Dynamically Launch Flow`" operation.



Group Id: com.hp.oo
Artifact Id: oo-oo-plugin
Version: 1.10.0
Action Name: Dynamically Launch Flow
Group Alias: /Base [1.9.0]/Configuration/Group Aliases/RAS_Operator_Path
Override Group: ${overrideJRAS}

Input	Sensitive Data	Required	Type	Assign From	Otherwise	Assign To
centralServer	☐	☐		centralServer	Use the constant: localhost	<not assigned>
centralUsername	☐	☐		centralUsername	Use the constant:	<not assigned>
centralPassword	✓	☐		centralPassword	Use the constant:	<not assigned>
centralProtocol	☐	☐		centralProtocol	Use the constant: http	<not assigned>
centralPort	☐	☐		centralPort	Use the constant: 8080	<not assigned>
authType	☐	☐		authType	Use the constant: anonymous	<not assigned>
proxy	☐	☐		proxy	Use the constant:	<not assigned>
proxyPort	☐	☐		proxyPort	Use the constant: 8080	<not assigned>
proxyUsername	☐	☐		proxyUsername	Use the constant:	<not assigned>
proxyPassword	✓	☐		proxyPassword	Use the constant:	<not assigned>
trustAllRoots	☐	☐		trustAllRoots	Use the constant: true	<not assigned>
keystore	☐	☐		keystore	Use the constant:	<not assigned>
keystorePassword	✓	☐		keystorePassword	Use the constant: ********	<not assigned>
trustKeystore	☐	☐		trustKeystore	Use the constant:	<not assigned>
trustPassword	✓	☐		trustPassword	Use the constant: ********	<not assigned>
centralFlowPath	☐	✓		centralFlowPath	Prompt user from the text	<not assigned>
centralFlowLaunchWaitForFlow	☐	☐		centralFlowLaunc...	Use the constant: false	<not assigned>
runName	☐	☐		runName	Use the constant:	<not assigned>
inputs	☐	☐		inputs	Use the constant:	<not assigned>
values	☐	☐		values	Use the constant:	<not assigned>
delimiter	☐	☐		delimiter	Use the constant: ,	<not assigned>
socketTimeout	☐	☐		socketTimeout	Use the constant: 0	<not assigned>

4. Expose the following properties:
   centralFlowPath, **runname**, **inputs**, **values**, and **delimiter** and ensure to set **loglevel** as "**STANDARD**" or "**EXTENDED**".
   Following is a table with sample values for each attribute:

Attribute	Sample Value
centralFlowPath	Library/OOProject/CloudSlangFlow1
runName	CloudSlang1
loglevel	EXTENDED

5. Create a topology based component in CSA using the above flow and expose the mentioned properties.
6. Enter the actual values of the CloudSlang content flow as required, when the topology design is using this above component.

Troubleshooting

OO

CSA provides a test-run feature for topology designs, whose results include event details and links to the OO master-flow execution log. To test a component, use it in a design and test the entire design.

For a deeper level of debugging, use the OO studio, and load master flows that are generated when a test run is performed in CSA. These master flows are for provisioning a service, modifying a service, or de-provisioning a service. The content jar file for the generated master flows is saved in the folder: `<CSA_INSTALLDIR>\jboss-as\standalone\tmp\e2e-flows`.

Chef components

During Chef provisioning, CSA triggers the cookbooks execution only, and communicates issues to the user. For more detailed information on debugging, connect to the provisioned machine as the node managed by Chef and invoke the Chef client directly.

Possible debugging issues include:

- Connectivity – connections to the following are required:
  - CSA (and OO) connection to the Chef host
  - CSA connection to the infrastructure provider used for machine provisioning
  - CSA (and OO) connection to the provisioned machine
  - Chef host connection to the provisioned machine
- The cookbook does not work well on the operating system installed on the provisioned machine. For example, the cookbook works for a specific Linux distribution only.

The amount of fine tuning depends on the nature of the imported content and user requirements. For example, components often need relationship definition, lifecycle configuration for operations, or operation parameter mapping.

SSH IP validation issues

When server provisioning uses a limited set of IP addresses, SSH prohibits access if identity conflicts occur when a new VM uses the same IP address that a deleted VM used.

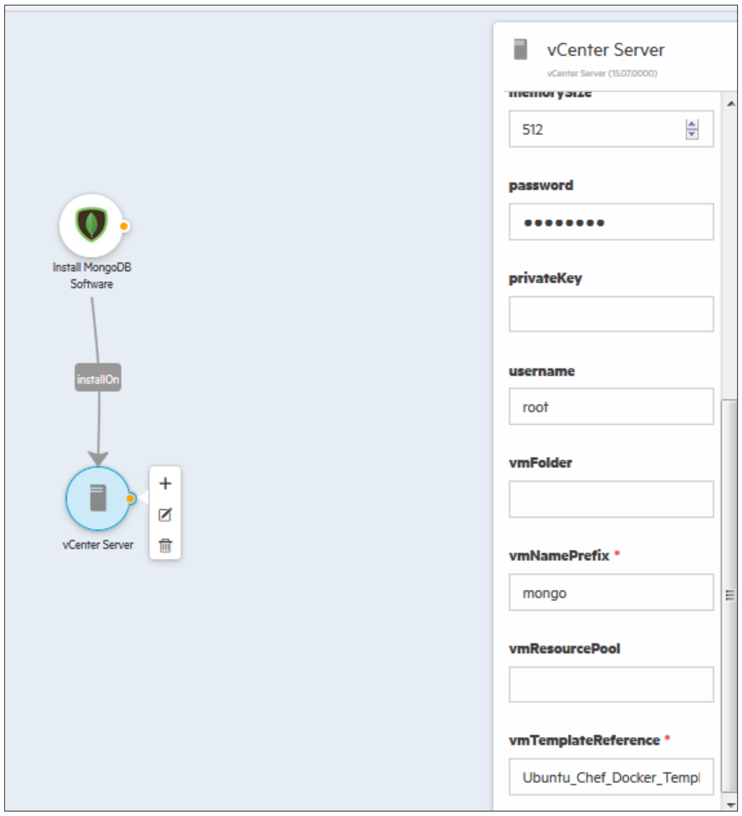
To resolve this, set the following option values in `/etc/ssh/ssh_config`:

`CheckHostIP` **no**
`StrictHostKeyChecking` **no**
`UserKnownHostsFile` **/dev/null**

Modifiable properties

During service creation or service modification, designers must explicitly mark as **modifiable** the topology-based service design properties. Modifiable properties must have a corresponding `prev_` input parameter in the underlying Modify flow.

In the vCenter example below, `cpuCount` is marked modifiable, but the designer did not choose to mark `memorySize` (which also qualifies) as modifiable). Other properties in the example (such as `vmTemplateReference`) are not marked modifiable because they do not have a corresponding `prev_` input parameter. Properties on various profiles can also be marked as modifiable during service creation and modification.



vCenter Server

vCenter Server (15.07.0000)

memorySize

512

password

••••••••

privateKey

username

root

vmFolder

vmNamePrefix *

mongo

vmResourcePool

vmTemplateReference *

Ubuntu_Chef_Docker_Templ

Terminology

The following terms are used in this document.

Capability

A capability is a special type of component that supports properties and relationships (but does not support operations or characteristics). When a concrete component supports a capability, the concrete component inherits the capability's relationships and must provide property mappings from the concrete component properties to the capability properties. Capabilities can be the target of relationships configured on a component. Capabilities can be included in a design, but for such a design to be successfully provisioned, another design must exist that contains a concrete component supporting the capability.

Chef cookbook

A Chef cookbook is a collection of grouped Chef recipes.

Chef recipe

A Chef recipe explains how Chef configures and manages server applications and utilities (such as MySQL).

Component (in Topology Designer)

A component represents one service-design element required to realize a service subscription. CSA provides a number of out-of-the-box components you can use for creating topology designs.

Component Properties (in Topology Designer)

Component properties provide a base set of attributes that can be used and edited when creating components in a service design. They represent configuration settings to be applied to the component during service-design provisioning. The value defined for a component property is the default value exposed in the service design.

Component Types

A base set of attributes that can be used and edited when creating service components in a service design.

Content (standard versus custom)

Content can be imported from OO as standard content or custom content. Standard content must follow specific directory and naming conventions to infer the provider type, component name, and version of a created component (with the flow and subflows located under a `Provider Type Name / Component Name / Version` directory). Custom content requires the user to specify the provider type and component name at import time.

Cloud Service Automation

Cloud Service Automation (CSA) is a unique platform that orchestrates the deployment of computer and infrastructure resources and of complex multi-tier application architectures. CSA integrates and leverages the strengths of several datacenter management and automation products, adding resource management, service offering design, and a customer portal to create a comprehensive service automation solution. The CSA subscription, service design and resource utilization capabilities address three key challenges:

The CSA Marketplace Portal provides a customer interface for requesting new cloud services and for monitoring and managing existing services, with subscription pricing to meet your business requirements.

The CSA graphical service design and content portability tools simplify developing, leveraging, and sharing an array of service offerings that can be tailored to your customers’ needs.

The CSA lifecycle framework and resource utilization features ease the complexity of mapping your cloud fulfillment infrastructure into reusable, automated resource offerings for on-time and on-budget delivery.

Cloud Service Management Console

Software that provides a CSA design and administration interface. The Cloud Service Management Console supports the following user roles: Administrator, Consumer Service Administrator, Resource Supply Manager, Service Designer, and Service Operations Manager.

Marketplace Portal

Software that delivers cloud services to subscribers (customers) by providing one or more service catalogs per organization. The Marketplace Portal is integrated into and shipped with CSA. The Marketplace Portal supports the following user roles: Consumer Organization Administrator, Consumer Organization Administrator, and Service Consumer.

Operations Orchestration

Operations Orchestration (OO) is a software product that coordinates communication between integrated products and managed devices. Customized OO flows are essential to implementing the CSA service lifecycle.

Operations Orchestration Flow

A run-book automation workflow composed of operations, subflows, and integrations that implement a discrete action. Flows are synchronized with CSA, and presented as actions, which can be directly attached to components. Operations Orchestration flows are created, modified, and saved using Operations Orchestration Studio. CSA includes a set of sample Operation Orchestration flows used by CSA’s sample service designs.

Server Automation

Server Automation (SA) is full-scale, virtual-server management automation software that delivers security, provisioning, policy management, deployment, and server compliance.

Lifecycle (in Topology Designer)

Each Topology component goes through a lifecycle transition: create, modify, and finally destroy (when not needed by the service anymore). These transitions throughout the life of the component are called the *lifecycle phase*.

Lifecycle Operations (in Topology Designer)

A topology component goes through various lifecycle phases: for example, Deploy, Modify, Un-deploy. Operations executed on these Lifecycle Phases are called *Lifecycle Operations*.

Parameter Mapping (in CSA)

Each Topology component defines a set of actions and each action typically declares a set of input parameters. A parameter mapping is defined to specify how the value should be passed for those input parameters when the action is executed. For example, value of an input parameter may be mapped to a component's property by creating a parameter mapping. In that case the value of the property is passed as the value of the input parameter when the action is executed.

Properties (modifiable properties)

Properties provide a base set of attributes that can be used and edited when creating components in a service design. They represent configuration settings to be applied to the component during service design provisioning. The value defined for a component property is the default value exposed in the service design.

Relationship

Relationships in topology designs define dependencies between components and also impact how a design is provisioned. For example, imported Chef components require a Server in order to be provisioned. Therefore, all imported Chef components are created with an Outgoing relationship to the Server capability, ensuring that a Server is provisioned before the Chef component.

Resource Provider

A management platform that provides either Infrastructure-as-a-Service (IaaS) or Software-as-a-Service (SaaS) to the cloud. For example, a provider of Matrix Operating Environment services provisions infrastructure and basic applications, while a provider of SiteScope services monitors applications.

Topology Design

Topology designs specify components, relationships, and properties. In contrast to sequenced designs, which more explicitly define the provisioning order and the sequence of actions that will run, topology designs are declarative in nature and do not include explicit actions or sequencing. The provisioning sequence is inferred by the relationships that exist between components in a topology design. Use topology designs for Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) deployments that are enabled using Chef, SA, and OO flow-based components.

Each topology design component binds to a single provider for fulfillment automation. Topology designs delegate component lifecycle provisioning to providers.

Send documentation feedback

If you have comments about this document, you can send them to clouddocs@hpe.com.

Legal notices

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: http://hpe.com/software/csa.

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 athttps://www.hpe.com/us/en/support.html.