The Orchestration process

Table of contents

1 Overview

When a system sends a request to use the **orchestration-service** service, there are four ways how the request can be handled based on the content of the orchestration form:

- If `externalServiceRequest` flag is set to true, then this means the orchestration is part of an inter-cloud orchestration and the request comes from the Gatekeeper Core System on behalf of a system from an other cloud. See details in the external service request description.
- If `triggerInterCloud` flag is set to true, then this means a modified version of dynamic orchestration in which the process skips the local part of the orchestration and immediately try to find matching providers in other clouds. See details in the inter-cloud orchestration process description.
- If `overrideStore` flag is set to true, then this means the dynamic orchestration (searching for a suitable provider in the whole Service Registry instead of the Orchestration Store). See details in the dynamic orchestration process description.
- Otherwise, orchestration will be handled by using the store rules. Currently, there are two kinds of stores: fix store and flexible store. The administrator can choose between them when they configure the system; but the two stores can't work simultaneously. Also, if the Orchestrator uses the fix store, the orchestration process is different whether the orchestration form contains a service requirement (normal fix store orchestration) or not (top priority store orchestration). See details below.

The rest of this document is organized as follows. In Section 2, we describe the dynamic orchestration. In Section 3, we describe the inter-cloud orchestration which also covers the case when `triggerInterCloud` flag is set to true. In Section 4, we describe the external service request case. In Section 5 and 6, we describe normal fix store orchestration and top priority store orchestration, respectively. Finally, in Section 7, we describe the flexible store orchestration.

2 Dynamic orchestration process

The following section describes the orchestration process when the dynamic option is selected. This means that the `overrideStore` flag is set to true in the orchestration form.

2.1 Checking the orchestration form

The first step is to make sure that the orchestration form is valid and contains everything we need for the process.

- If the `onlyPreferred` flag is set to true, preferred provider list cannot be empty. Please note that if the preferred provider list only contains foreign systems (systems from an other cloud) and the inter-cloud orchestration is not allowed, then this check still gives an error, since there is no way to provide any results.
- If requester wants exclusive rights (`enableQoS` flag is true and `qosExclusivity` command is used), then they have to set `matchmaking` flag to true, too.
- Requester system has to specify its name, address and valid port.
- Requester system has to specify a valid service definition requirement in the embedded service query form.
- If the preferred provider list is specified, all provider systems must have a name, address and valid port. If the preferred system is in an other cloud, cloud name and operator also have to be specified.
- If the Orchestrator supports Quality-of-Service (QoS) and requester wants exlusive rights (`enableQoS` flag is true and `qosExclusivity` command is used), the specified exclusivity time must be a positive integer that is lower than the maximum reservation duration (which is configurable).

2.2 Using the Service Registry

The Orchestrator calls the Service Registry's **query** service with the specified service query form (that is part of the orchestration form), but before that it can make some changes on that form:

- `pingProviders` flag of the orchestration form overrides the value in the service query form;
- if `metadataSearch` flag is false, the metadata requirements in the service query form is ignored;
- provider address type requirements can be overrided if any of the related orchestration flags is set;

If the Service Registry's **query** returns an empty list, the process can do two things:

- if inter-cloud orchestration is possible (orchestrator is configured to work with the Gatekeeper Core System and `enableInterCloud` flag is set in the orchestration form), the process starts an inter-cloud orchestration (see below).
- otherwise returns an empty result.

2.3 Using the Authorization

If the Service Registry's **query** produces some providers, the next step is checking the access rights of the requester for every provider using the Authorization Core System **authorization-control-intra** service.

We remove every provider from the list that the requester cannot use because it has no permission.

If the remaining provider list is empty, the process can do two things:

- if inter-cloud orchestration is possible, the process starts an inter-cloud orchestration (see below).
- otherwise returns an empty result.

2.4 Handling `onlyPreferred` orchestration flag

If the `onlyPreferred` flag is set, we remove any providers from the remaining provider list that is not in the preferred providers list. After that, if the remaining provider list is empty, the process can do two things:

- if inter-cloud orchestration is possible and there are non-local providers in the preferred providers list, the process starts an inter-cloud orchestration (see below).
- otherwise returns an empty result.

2.5 Orchestration warnings

This step calculates some orchestration warnings based on the service instance end of validity data (`TTL_UNKNOWN`, `TTL_EXPIRING`, `TTL_EXPIRED`).

2.6 Filtering reserved providers (1)

This step only occurs if the Orchestrator supports QoS. We removes every provider from the list that is reserved by someone else.

If the remaining orchestration result list is empty, the process can do two things:

- if inter-cloud orchestration is possible, the process starts an inter-cloud orchestration (see below).
- otherwise returns an empty result.

2.7 Temporary reservation

If QoS is supported and the requester wants exlusivity, then in this step the process locks all providers in the remaining orchestration result list. This is just a short time lock to make sure a parallel orchestration process does not "steal" providers from us while this orchestration process is running.

2.8 Quality-of-Service requirements

If QoS is supported and the requester specifies QoS requirements (and set the `enableQoS` flag), in this step we remove any providers from the remaining orchestration result list that not meet those requirements. If requester wants exclusivity for a specified duration, that also can shrink the size of

the list because providers can specify a *"recommended"* exclusivity duration in their service instance level metadata and too much difference can remove the provider from the list.

If the remaining orchestration result list is empty, the process can do two things:

- if inter-cloud orchestration is possible, the process starts an inter-cloud orchestration (see below).
- otherwise returns an empty result.

2.9 Filtering reserved providers (2)

We do this step again, because during the previous step other orchestration processes can "steal" some providers from our list. We can skip this step if the requester requires exclusivity because in this case we locked all potential providers previously.

If the remaining orchestration result list is empty, the process can do two things:

- if inter-cloud orchestration is possible, the process starts an inter-cloud orchestration (see below).
- otherwise returns an empty result.

2.10 Token generation (1) and Filtering reserved providers (3)

If QoS is supported we may generate tokens in this step. It is only needed if there is at least one provider that uses `TOKEN` security type for the specified service. In this case, we call the Authorization Core System's **token-generation** service to generate the tokens, which then are integrated into the orchestration results.

The best place for token generation is when the orchestration result list's size is minimal (if `matchmaking` is enabled, the size is one at the most). But if the Orchestrator supports QoS, it means that the result list can always lose providers because of other processes' reservations. Calling an other system's service is a slow operation so we want to make sure that everything is ready before the potentional matchmaking to minimalize the chance that someone "steal" that one provider.

Because the previously mentioned slowness, we have to filter reserved providers again (but only if we did not lock the providers before).

If the remaining orchestration result list is empty, the process can do two things:

- if inter-cloud orchestration is possible, the process starts an inter-cloud orchestration (see below).
- otherwise returns an empty result.

2.11 Matchmaking

If `matchmaking` flag is set, the orchestration process selects exactly one provider from the orchestration results (if any). This selection can achieved using various strategies. The only restriction is that the selection algorithm has to use the preferred provider list (if specified). Here some examples:

- Select the first preferred provider that is also present in the orchestration results. If no match is found or no preferred provider is specified, select the first orchestration result.
- Select the first preferred provider that is also present in the orchestration results. If no match is found or no preferred provider is specified, select a random orchestration result.

After the selection is done, and the requester wanted exclusivity, we extend the temporary lock on the selected provider and release all the others immediately.

2.12 Token generation (2)

If QoS is not supported we may generate tokens in this step. It is only needed if there is at least one provider that uses `TOKEN` security type for the specified service. In this case, we call the Authorization Core System's **token-generation** service to generate the tokens, which then are integrated into the orchestration results.

2.13 Returning results

The last step is to return the remaining orchestration results.

3 Inter-cloud orchestration process

The inter-cloud orchestration process is started in two cases:

- the dynamic orchestration process calls it, when no local provider meets the requirements and the `enableInterCloud` flag is set;
- or if the `triggerInterCloud` flag is set, we skip the dynamic orchestration and immediately calls the inter-cloud process. Please note that if the `triggerInterCloud` flag is set to true, the requester has to set `enableInterCloud` flag to true as well.

3.1 Checking the orchestration form

The first step is to make sure that the orchestration form is valid and contains everything we need for the process.

- If the `onlyPreferred` flag is set to true, preferred provider list cannot be empty.
- If requester wants exclusive rights (`enableQoS` flag is true and `qosExclusivity` command is used), then they have to set `matchmaking` flag to true, too.
- Requester system has to specify its name, address and valid port.
- Requester system has to specify a valid service definition requirement in the embedded service query form.
- If the preferred provider list is specified, all provider systems must have a name, address and valid port. Cloud name and operator also have to be specified.
- If the Orchestrator supports Quality-of-Service (QoS) and requester wants exlusive rights (`enableQoS` flag is true and `qosExclusivity` command is used), the specified exclusivity time must be a positive integer that is lower than the maximum reservation duration (which is configurable).

3.2 Global service discovery

The next step is to call the Gatekeeper Core System's **global-service-discovery** service with the proper form. The form contains three things:

- the service query form (from the orchestration form);
- preferred clouds (need to collect the clouds from the preferred provider list entries, if any);
- and a flag that informs the Gatekeeper whether we need QoS measurements, too, or not (it basically the value of the `qosEnabled` flag).

The Gatekeeper contacts known neighbour clouds, and asks them about the service our requester needs. The result contains how many suitable provider are available for each cloud (and if necessary, QoS measurements for each provider) and some other information.

If the global service discovery does not return any options, the orchestration returns an empty result.

3.3 Quality-of-Service requirements preverification

If QoS requirements are specified, we have to take this step before we selecting a cloud. The number of potential providers per cloud is affecting the selection and some foreign providers may not meet the necessary QoS requirements, se we have to remove them from the result of the global

service discovery.

If a foreign provider can be accessed directly, the verifier algorithm uses the measurement data that comes with the global service discovery result to make the necessary filtering.

However, when a foreign provider can only be accessed via the Gateway Core System, the measurements get an overhead (which is the sum of the communication overhead between the consumer and the Gateway and the communication overhead between the two Gateways). The Gateway-Gateway overheads are measured by the QoS Monitor Core System periodically. So in this case, the verifier asks the data about this overhead from the QoS Monitor and uses it to make the filtering.

The providers that meet the QoS requirements become preferred providers of the orchestration request. If `onlyPreferred` flag is set to true, we use the intersection of the original preferred providers and the QoS-verified providers, otherwise we override the original list with the QoS-verified providers.

3.4 Cloud selection

The orchestration process selects exactly one cloud from the global service discovery results (if any). This selection can achieved using various strategies. The only restriction is that the selection algorithm has to use the preferred cloud list (derived from the preferred provider list, if specified).

Here some examples:

- Select the first preferred cloud that is also present in the global service discovery results. If no match is found or there is no preferred cloud, select the first cloud in the results (or returns empty response if `onlyPreferred` flag is set).
- Select a random preferred cloud that is also present in the global service discovery results. If no match is found or there is no preferred cloud, select a random cloud in the results (or returns empty response if `onlyPreferred` flag is set). The random selection takes into consideration the number of potentional providers of the cloud, so a cloud with more potentional providers has more chance.

3.5 Inter-cloud negotiation

The orchestration process creates an inter-cloud negotiation form that contains the service query form, the target cloud (selected in the previous step), the prefrerred systems from the target cloud (if any), orchestration flags and commands, and some other information.

Then the process calls the Gatekeeper Core System's **inter-cloud-negotiations** service to do the orchestration in the other cloud.

If the service returns empty response, the orchestration process returns empty response too.

3.6 Quality-of-Service requirements verification

If QoS requirements are specified, we have to filter the result list using the requirements. This step can be skipped if

- provider reservation command is specified, or
- only one result is returned and that service instance only accessible via Gateway

because in these cases the provider can only come from the pre-verified providers.

3.7 Orchestration warnings

This step adds the orchestration warning `FROM_OTHER_CLOUD` to the results.

3.8 Matchmaking

The next to last step is to select one result from the result list if `matchmaking` flag is set and still has more than one results.

3.9 Returning results

The last step is to return the remaining orchestration results.

4 External service request

The external service request orchestration process is started when `externalServiceRequest` flag is set to true. Please note that an ordinary application system can not set this flag (it causes an authentication error), only the Gatekeeper Core System has permission to do that.

This case represents the orchestration process where the requester system is NOT in the local cloud.

This means that the Gatekeeper Core System made sure that this request from the remote Orchestrator can be satisfied in this cloud (Gatekeeper polled the Service Registry Core System and will poll Authorization Core System).

4.1 Checking the orchestration form

The first step is to make sure that the orchestration form is valid and contains everything we need for the process.

- If the `onlyPreferred` flag is set to true, preferred provider list cannot be empty.
- If requester wants exclusive rights (`enableQoS` flag is true and `qosExclusivity` command is used), then they have to set flag to true, too.
- Requester system has to specify its name, address and valid port.
- Requester cloud has to specify its name and operator.
- Requester system has to specify a valid service definition requirement in the embedded service query form.
- If the preferred provider list is specified, all provider systems must have a name, address and valid port.
- If the Orchestrator supports Quality-of-Service (QoS) and requester wants exlusive rights (`enableQoS` flag is true and `qosExclusivity` command is used), the specified exclusivity time must be a positive integer that is lower than the maximum reservation duration (which is configurable).

4.2 Using the Service Registry

The Orchestrator calls the Service Registry's **query** service with the specified service query form (that is part of the orchestration form). This query does not return empty response because the Gatekeeper Core System checked the Service Registry during the global service discovery just moments ago.

4.3 Handling `onlyPreferred` orchestration flag

If the `onlyPreferred` flag is set, we remove any providers from the remaining provider list that is not in the preferred providers list.

4.4 Filtering reserved providers (1)

This step only occurs if the Orchestrator supports QoS. We removes every provider from the list that is reserved by someone else.

4.5 Filtering on recommended service time

If QoS is supported and enableQoS flag is set and there is a provider exclusivity requirement, in this step the process removes any providers that not allows the specified duration of exclusivity. Please

note that all the other QoS requirements verification are happen elsewhere (see preverification and verification steps in the previous section).

4.6 Token generation (1) and Filtering reserved providers (2)

We generate tokens in this step. It is only needed if there is at least one provider that uses `TOKEN` security type for the specified service. In this case, we call the Authorization Core System's **token-generation** service to generate the tokens, which then are integrated into the orchestration results.

Because the token generation is a slow operation, we have to filter reserved providers again.

4.7 Returning results

The last step is to return the remaining orchestration results.

5 Fix store orchestration process

The following section describes the orchestration process if the Orchestrator Core System is set to use the original, fix store and the `overrideStore` orchestration flag is set to false (this is the default value). Please note that the fix store orchestration does not use Quality-of-Service requirements and commands.

5.1 Checking the orchestration form

The first step is to make sure that the orchestration form is valid and contains everything we need for the process.

- Requester system has to specify its name, address and valid port.
- Requester system has to specify a valid service definition requirement in the embedded service query form.
- The interface requirements list in the embedded service query form must contain exactly one interface.

5.2 Getting the consumer system data

In the next step, the process calls the Service Registry's **query-by-system** service to check the consumer registration and acquire its database id.

5.3 Getting the related store rules

Using the data from the previous query and the service query form, the process loads the related rules from the orchestration store ordered by priority.

If the entry list is empty then the process returns empty result.

5.4 Creating cross-check list for local entries

This step only need if you have store rules that contains local providers. If every entry are related to foreign providers, we can skip this step.

The Orchestrator calls the Service Registry's **query** service with the specified service query form (that is part of the orchestration form), but before that it can make some changes on that form:

- `pingProviders` flag of the orchestration form overrides the value in the service query form;
- if `metadataSearch` flag is false, the metadata requirements in the service query form is ignored;
- provider address type requirements can be overrided if any of the related orchestration flags is set;

Then we checks the access rights of the requester for every provider (returned by the Service Registry) using the Authorization Core System **authorization-control-intra** service.

The created list contains all the providers that offer the specified service and the consumer has the permission to use it.

5.5 Selecting the provider with the highest priority that currently available

The process checks every store rule from the highest priority (which is 1) to the lowest to find a provider that is currently available (in the sense that its service is in the Service Registry and the consumer has the necessary permission to use it).

We have to handle two cases:

5.5.1 Local rule

If the rule contains a local provider, we use the list that is created in the previous step to check whether this provider is available for the consumer or not.

If the provider matches our requirements we checks its service instance's security type. In case of `TOKEN` security type, we call the Authorization Core System's **token-generation** service to generate the token, which then is integrated into the orchestration result.

5.5.2 Foreign rule

If the rule contains a provider from an other cloud, the process uses the inter-cloud negotiation to check the provider's availability.

The orchestration process creates an inter-cloud negotiation form that contains the service query form, the target cloud (comes from the store rule), the prefrerred system from the target cloud (comes from the store rule) and orchestration flags (`matchmaking` flag is automatically set to true).

Then the process calls the Gatekeeper Core System's **inter-cloud-negotiations** service to do the orchestration in the other cloud. If the request returns a valid orchestration response that means that the provider from the rule is available for the consumer.

After the selection step we have an orchestration response. If that is empty, that means there is no suitable provider for the consumer in the store. Otherwise, the response contains the access information (including tokens) for the consumer to use the service.

5.6 Filtering reserved providers

Although the store orchestration does not support the QoS requirements and provider reservation this does not mean that a store orchestration request returns already reserved providers. So if the Orchestrator itself supports QoS, in this step we drop the result if that provider is reserved by someone else.

5.7 Returning the result

The last step is to return the orchestration result or an empty result if we can not find available provider.

6 Top priority orchestration process

There is a variation of the fix store orchestration process. In this case, the requester only specify its own database id (as a system) and the Orchestrator returns the top priority provider for every service that is specified in the orchestration store if it can.

The same results can be achieved by starting a normal fix store orchestration with an orchestration form that contains a requester but not a requested service.

Please note that this variation does not support inter-cloud rules so you can not use it if your top priority providers are in other clouds.

6.1 Getting the consumer system data

In the first step, the process calls the Service Registry's **query-by-id** service to check the consumer registration and acquire its system details.

6.2 Getting the related store rules

Using the specified consumer id the process loads the related rules from the orchestration store. Please note that in this step we loads only the top priority (1) rules for different service/interface combinations.

If the entry list is empty then the process returns empty result.

6.3 Cross-checking the list for local entries

As previously mentioned, this variant does not support foreign providers, so in this step we remove any foreign providers from the store rules list.

Next the Orchestrator calls the Service Registry's **query** service for the specified service defintion name (and including the interfaces as interface requirements).

Then we checks the access rights of the requester for every provider (returned by the Service Registry) using the Authorization Core System **authorization-control-intra** service.

The result list contains all the providers that offer one of the specified services (in the rules) and the consumer has the permission to use it.

If the result list is empty then the process returns empty result.

6.4 Filtering reserved providers

Although the top priority orchestration does not support the QoS requirements and provider reservation this does not mean that a top priority orchestration request returns already reserved providers. So if the Orchestrator itself supports QoS, in this step we drop the any orchestration result if the provider is reserved by someone else.

6.5 Token generation (1) and Filtering reserved providers (2)

We generate tokens in this step. It is only needed if there is at least one provider that uses `TOKEN` security type for the specified service. In this case, we call the Authorization Core System's **token-**

generation service to generate the tokens, which then are integrated into the orchestration results.

Because the token generation is a slow operation, we have to filter reserved providers again.

6.6 Returning the results

The last step is to return the orchestration results or an empty result if we can not find any available provider.

7 Flexible store orchestration process

The following section describes the orchestration process if the Orchestrator Core System is set to use the newer, flexible store and the `overrideStore` orchestration flag is set to false (this is the default value). Please note that the flexible store orchestration does not support inter-cloud rules, Quality-of-Service requirements and commands.

This mode also skips permission checks by the Authorization Core System currently.

The rules in this mode are more flexible than the fix rules:

- you can specify the consumer name as text, the provider name as text and the service definition as text, so you can create rules before registering a provider, a consumer or even a service.
- in version 4.4.0 we introduced metadata at system level, and you can use this feature to create wildcard rules.

For example:

- you can tell, this rule is applied to any consumer that has these metadata key-value pairs;
- you can tell, this consumer can use any provider, who provides this particular service AND has these metadata key-value pairs (in system level);
- and combine these.

7.1 Checking the orchestration form

The first step is to make sure that the orchestration form is valid and contains everything we need for the process.

- If the `onlyPreferred` flag is set to true, the preferred provider list cannot be empty. Please note that if the preferred provider list only contains foreign systems (systems from an other cloud), then this check still gives an error, since inter-cloud orchestration is not supported in this mode so there is no way to provide any results.
- Requester system has to specify its name, address and valid port.
- Requester system has to specify a valid service definition requirement in the embedded service query form.
- If the preferred provider list is specified, all provider systems must have a name, address and valid port.
- The requester can not use some of the orchestration flags: `enableInterCloud`, `triggerInterCloud` and `enableQoS`.

7.2 Getting the consumer system data

In the next step, the process calls the Service Registry's **query-by-system** service to check the consumer registration and acquire its system details.

7.3 Collecting the matching rules

In this step we collect all the store rules that can be used with this particular consumer. This is a multistage process.

First, we get all the rules from the store where the service definition name and the consumer name match with the specified ones (name based rules). If interface requirements are specified, we remove those rules that do not meet these requirements. Please note that a rule without any interface matches any interface requirement.

Next, we creates the current consumer system metadata. This can be done by merging the consumer system metadata from the orchestration form and the consumer system metadata from the Service Registry. If both metadata contains the same key, the value from the former overrides the value from the latter.

After that, we get all rules from the store where the service definition name matches with the specified one and has consumer metadata requirements (metadata based rules). If interface requirements are specified, we apply those. A metadata based rule matches with the consumer if the consumer's current metadata contains all the key-value pairs from the consumer metadata requirements of the rule. Please note that a metadata based rule can also contains consumer name requirement and in this case both the names and the metadata must match.

We collect the remaining name based and metadata based rules (and removing any duplicates), then sort them by priority and database id.

If no related rules are found, the process returns an empty orchestration result.

7.4 Querying the Service Registry

The process create a service query form for each rule using the following algorithm:

- service definition requirement comes from the original request;
- interface requirement comes from the rule;
- metadata requirement comes from the rule (from service metadata) by default, but if `metadataSearch` orchestration flag is set to true and metadata requirements are specified in the original request, then we create a merge of the two metadata requirements, where the original metadata requirements take precedence if there are key collisions;
- security requirement comes from the original request;
- version requirements comes from the original request;
- ping provider requirement comes from the orchestration flag.

Then the Orchestrator calls the Service Registry's **query-multi** service with these service query forms, then create a structure which assigns a response list to the store rule from which the service query form is created.

7.5 Filtering service query results by provider requirements

In this step, the process removes any service query result that does not match to the provider requirements (coming from the rule and/or orchestration request). For each rule, we check the appropriate service query results and remove any provider

- that is not in the preferred providers list (only if `onlyPreferred` flag is set);
- that has a name that not match with the provider name requirement (if the rule has a provider name requirement);
- whose system metadata does not match with the provider metadata requirements (if the rule

has provider metadata requirements). A provider metadata matches with the rule's provider metadata requirements if the provider's metadata contains all the key-value pairs from the provider metadata requirements of the rule.

Please note that the process maintained the sorting in all previous steps, so the providers that belongs to the more important rule are precedes those that belongs to a less important rule. The algorithm also make sure that a provider only appears in the list once.

If no providers are remained, the process returns an empty orchestration result.

7.6 Orchestration warnings

This step calculates some orchestration warnings based on the service instance end of validity data (`TTL_UNKNOWN`, `TTL_EXPIRING`, `TTL_EXPIRED`).

7.7 Matchmaking

If `matchmaking` flag is set, the orchestration process selects exactly one provider from the orchestration results (if any). This selection can achieved using various strategies. The only restriction that the selection algorithm has to use the preferred provider list (if specified). Here some examples:

- Select the first preferred provider that is also present in the orchestration results. If no match is found or no preferred provider is specified, select the first orchestration result.
- Select the first preferred provider that is also present in the orchestration results. If no match is found or no preferred provider is specified, select a random orchestration result.

7.8 Token generation

We may generate tokens in this step. It is only needed if there is at least one provider that uses `TOKEN` security type for the specified service. In this case, we call the Authorization Core System **token-generation** service to generate the tokens, which then are integrated into the orchestration results.

7.9 Returning results

The last step is to return the remaining orchestration results.