Science Gateway Use Cases

August 22, 2019

Version 1.1

These use cases describe how research communities use community computing resources to power their "science gateways," supporting the specialized needs of research fields, communities of practice, and joint initiatives. Science gateways are applications—most often web-based—that are used by groups of researchers with similar needs. Each gateway is developed and operated by one or more leaders in the research field who applies for an allocation to serve the community. Gateways can also help researchers who have their own allocations by providing a more customized, user-friendly interface.

SGW-01: Acquire credentials that allow use of community resources

A **science gateway developer** needs to enable a gateway to acquire credentials that allow the gateway to use community resources and/or services. A community resource is a resource provided to the community by a service provider (e.g., a compute cluster, a storage system). A community service is a service that adds value to the community's resources (e.g., single sign-on, data transfer, user support).

We assume the following things.

1. The gateway is registered with the community and possesses its own credentials that enable it to securely use the community's authentication service.
2. The gateway has a mechanism for authenticating individual gateway users and it maintains its own list of authorized gateway users.

In most cases, the developer expects it to work as follows.

1. First, a gateway user logs into the gateway.
2. Then, the gateway uses its own credentials, the gateway user's identity, and the community authentication service to acquire credentials that allow it to use community resources and/or services.
3. Then, the gateway combines the acquired credentials as needed into a security context (a set of verifiable credentials).
4. Finally, the gateway uses the security context to securely interact with community resources and/or services.

We'll take any solution, as long as the following are true.

1. In Step 1, the gateway developer is responsible for the mechanism(s) by which the gateway logs gateway users in.
2. In Step 2, the interaction with the community authentication service takes 15 seconds or less.
3. In Step 2, the API provided by the community authentication service should be available in Java, Python, PHP, and C++ (in order of priority, highest to lowest).
4. In Step 3, it takes less than 5 seconds to create a security context.
5. The community authentication service is available 99.9% of the time. (No more than 8.77 hours downtime per year.)
6. The gateway can securely store its own credentials and any credentials obtained from the community authentication service.

SGW-02: Transfer files to and from a community resource

A **science gateway developer** needs to enable a gateway to transfer files to or from a community resource. A community resource is a resource provided to the community by a service provider (e.g., a compute cluster, a storage system).

We assume the following things.

1. The gateway is registered with the community and belongs to a project that has received an allocation to use the resource.

2. The gateway has already acquired credentials that allow it to access community resources and/or services. *(This is described in use case SGW-01.)*
3. The files to be transferred are located on (or must be transferred to) the gateway's host system. (See alternate scenarios below.)
4. The gateway's host system has a public IP address.

In most cases, the developer expects it to work as follows.

1. First, a gateway user logs into the gateway and does something that requires file transfers to or from a community resource.
2. Then, the gateway accesses a community file transfer service ("transfer service") and requests the file transfers.
3. Then, the gateway monitors the status of the transfer request using the transfer service's interface.
4. While the transfer status is pending or active, the gateway monitors the transfer status and can provide status information to the gateway user.
5. When the transfer status changes to "completed" (the specific value may vary), the gateway notifies the gateway user of success or failure.

It will always work like this unless one of the following is true instead.

1. The files to be transferred are on (or must be transferred to) the gateway user's system. In that case, the gateway user may first need to install and configure a software application on the local system to enable and allow the transfer service to access the files.
2. The files to be transferred are on (or must be transferred to) a third system (neither the gateway host nor the gateway user's system). In that case, the third system must be accessible by the transfer service and the gateway must be authorized to access the files and/or storage.
3. Use case SGW-04 describes a scenario where the gateway user transfers files from (or to) a community resource without logging into the gateway. (See use case SGW-04.)

We'll accept any solution as long as the following are true.

1. The transfer service is available 99.9% of the time. (No more than 8.77 hours of unscheduled downtime per year.)
2. In Steps 2-4, the gateway authenticates to the transfer service using credentials acquired as described in use case SGW-01 in a manner consistent with use case CAN-06.
3. In Steps 2-4, the API provided by the transfer service should be available in Java, Python, PHP, and C++ (in order of priority, highest to lowest).
4. In Step 5, if the transfer status is "completed" and the result is successful, then all files have been transferred accurately and completely without exception.
5. In Step 5, if the transfer status is "completed" but the result is not successful, sufficient diagnostic information is provided to enable the gateway operator to seek help from the community's user support service.
6. When files are transferred to a community resource, the resulting files are owned by the gateway's account on that resource.

7. When files are transferred to or from the gateway user's system (first alternate scenario), any software that the gateway user must install to enable the transfers has the following characteristics.
    a. The software is as easy to install as any other application.
    b. The software has a license allowing the gateway user to use it.
    c. The software does not enable anyone/anything other than the gateway (via the transfer service) to transfer files. (The gateway user may authorize other gateways or other users, but only if the gateway user so chooses.)
    d. The gateway user is able to configure which files and/or storage the software is able to access on the local system.
8. When files are transferred to or from the gateway user's system (first alternate scenario), the gateway user's system can have a dynamic IP address and can be behind a NAT.

SGW-03: Run an application on a community resource

A science gateway developer needs to enable a gateway to run an application on a community resource. The application may be: an individual job (a single application run that produces a single set of results); a suite of jobs (parameter sweep, ensemble, or workflow); or an interactive job that accepts input while it runs, producing a stream of results.

We assume the following are true.

1. The gateway is registered with the community and belongs to a project that has received an allocation to use the resource.
2. The gateway has already acquired credentials that allow it to access community resources and/or services. *(This is described in use case SGW-01.)*
3. The application is already installed and validated on the community resource. (*Use case HPC-01 describes the process for installing and validating applications on resources.)*
4. The gateway formulates the inputs and options for the application run, either programmatically or by soliciting them from the gateway user.

In most cases, the developer expects it to work as follows.

1. First, a gateway user logs into the gateway and does something that requires the gateway to run an application on the resource.
2. If necessary, the gateway transfers input files and/or scripts to the resource as described in use case SGW-02.
3. Then, the gateway submits the job to the resource.
4. While the job is queued and/or active, the gateway monitors the job status.
5. While the job is queued and/or active, the gateway may cancel the job.
6. When the job completes, the gateway uses the status code to determine whether or not the job was successful.
7. If necessary, the gateway retrieves output files from the resource as described in use case SGW-02.
8. Finally, the gateway uses the results of the job to respond to the gateway user's interaction.

It will always work like this unless one of the following is true instead.

1. If the application is interactive, the gateway may connect to the job when the job begins running on the resource and use that connection to receive streaming results and to provide input to the job, possibly involving the gateway user in this interaction. The details of this connection are beyond the scope of this use case.
2. If the application is a suite of jobs, the gateway might use a workflow manager application to manage the jobs. Use case HTC-04 (alternate scenario) describes how this would work.

We'll accept any solution as long as the following are true.

1. In Step 3, any job submission options provided by the resource are available to the gateway.
2. In Step 4, the gateway is able to monitor the job's status via periodic poll or asynchronous notification.
3. In Steps 3-6, the API provided by the job submission interface should be available in Java, Python, PHP, and C++ (in order of priority, highest to lowest).
4. In Steps 3-6, the error rate for jobs submitted by the gateway is the same or better than the error rate for the same jobs submitted manually by a human.
5. In Steps 3-6, errors returned by the resource's queueing system are returned to the gateway in full fidelity. Errors returned by any intermediary interfaces (remote submission service, middleware) are clearly distinguished and sufficient to allow the developer to get help from the community's user support service.

SGW-04: Enable gateway users to transfer files to or from a community resource

A science gateway developer needs to enable gateway users to transfer files between a community resource and the gateway user's local system without passing through the gateway host. These gateway users aren't members of the gateway's community project and don't have access to the gateway's allocation on the resource. The files and storage on the resource are owned by the gateway.

We assume the following are true.

1. The gateway is registered with the community and belongs to a project with an allocation to use the community resource.
2. The gateway has already acquired credentials that allow it to access community resources and/or services. *(This is described in use case SGW-01.)*
3. The gateway user is registered with the gateway.
4. The gateway user is registered with the community's file transfer service ("transfer service").

In most cases, the developer expects it will work as follows.

1. First, the gateway user logs into the gateway and does something that requires file transfers to or from a resource on which the gateway has an allocation.
2. Then, the gateway obtains the gateway user's registered identity with the transfer service. (The gateway may need to ask the gateway user to enter this identity.)
3. The gateway uses the transfer service API to create a file share on the resource that includes the files to be transferred or the storage space into which files will be transferred.

4. The gateway uses the transfer service API to adjust the permissions on the file share to permit the gateway user to access the file share. (Either read-only for downloads or read-write for uploads.)
5. The gateway gives the gateway user an identifier or address that allows the gateway user to connect to the file share using the transfer service.
6. If necessary, the gateway gives the gateway user instructions for how to download software to make the gateway user's local system accessible by the transfer service.

In most cases, the gateway user expects it will work as follows.

1. First, the gateway user logs into the gateway and does something that requires file transfers to or from a resource on which the gateway has an allocation.
2. Then, the gateway asks for the gateway user's registered identity with the community file service and the gateway user enters it.
3. Then, the gateway gives the gateway user instructions for how to download and install software to make the gateway user's local system accessible by the transfer service.
4. The gateway user follows the instructions and installs the software, configuring it so the files to be transferred (or the storage into which the files will be transferred) are accessible. The software provides an identifier or address for use with the transfer service.
5. Then, the gateway gives the gateway user another identifier or address that allows the gateway user to connect to the file share on the resource.
6. Then, the gateway user accesses the transfer service and logs in with the identity provided to the gateway in Step 1 above.
7. Then, the gateway user enters the addresses provided by the local software in Step 3 and by the gateway in Step 4 to set up the transfer. The gateway user selects the files to be transferred and initiates the transfer. At this point, the gateway user may log out of the transfer service. The transfer will be managed by the transfer service until all of the requested files have been transferred.
8. When the transfer completes, the gateway user will receive an email notification.

We'll accept any solution as long as the following are true.

1. A single transfer can involve millions of individual files and hundreds of terabytes (TBs).
2. In the developer's Steps 3-4, the API provided by the transfer service should be available in Java, Python, PHP, and C++ (in order of priority, highest to lowest).
3. Between the gateway user's Steps 7 and 8, if the gateway user shuts down the local system, disconnects from the network, or quits the transfer software, the transfer is suspended and will restart automatically when connectivity is restored.
4. In Step 2 for both the gateway developer and the gateway user, the gateway user's registered identity with the transfer service can be a community identity or it can be an identity with another organization (campus identity, ORCID ID, Google ID, etc.).
5. The gateway user's system can have a dynamic IP address and can be behind a NAT.
6. If the gateway user must install software on the gateway user's local system to enable the transfers, the software has the following characteristics.
    a. The software is as easy to install as any other application.

b. The software has a license allowing the gateway user to use it.
c. The software does not enable anyone or anything other than the gateway user (via the transfer service) to transfer files. (The gateway user may authorize other gateways or other users, but only if the gateway user so chooses.)
d. The gateway user is able to configure which files and/or storage the software is able to access on the local system.

7. Folder/directory structures are preserved.
8. File metadata (e.g., filenames, timestamps) is preserved.

SGW-05: Obtain information about a community resource

A **science gateway developer** needs to enable a gateway to obtain detailed configuration or status information about a community resource so the gateway can use the resource efficiently. A community resource is a resource provided to the community by a service provider (e.g., a compute cluster, a storage system). Configuration information is information about how the resource is configured on a seldom-changing basis. (E.g., system architecture, job queue names, queue requirements, data transfer endpoints, storage systems.) Status information is the dynamic status of the resource. (E.g., current number of jobs in a queue, estimated queue wait times, current free storage.)

We assume the community has established a unique identifier for each community resource and a standard taxonomy/schema for resource configuration and status information.

In most cases, the developer expects it to work as follows.

1. First, the gateway developer specifies the community resources to be used by the gateway.
2. Then, whenever the gateway needs resource configuration or status information, it uses the community information service's search interface and specifies the resource's identifier and the required information. The interface returns the most recent information.
3. Then, the gateway adjusts how it uses the resource based on the information it received.

It will always work like this unless the gateway needs to be notified when configuration or status information changes. In this case, in Step 2, the gateway uses the community information service's subscription interface instead of its query interface. While the subscription is active, whenever the information changes, the community information service delivers a notification to the gateway with the updated information. The subscription remains active for a specified period unless the gateway cancels the subscription.

We'll accept any solution as long as the following are true.

1. In Step 2, the API provided by the information service should be available in Java, Python, PHP, and C++ (in order of priority, highest to lowest).
2. The information service is available 99% of the time. (No more than 3.65 days per year of downtime.)
3. For high-performance computing and high-throughput computing resources, the information service provides the following configuration information.
    - List of software (including modules that can be activated)
    - Resource limitations: total cores

- Node limitations: memory, cores
- queues limitation
- max # cores/nodes
- max walltime
- queue attributes required
- project/account name
- queue attribute options
- email notification (Y/N)
- Supported protocols/tools for job submission and their end points
- Supported protocols/tools for data transfer and their end points
- Storage mount points (home, work, scratch) and their limits
- Resource reservation requirements

4. Current availability status (up/down) for all community resources and services is available.
5. All information includes timestamps indicating when the information was generated or measured.
6. The information provided by the system is accurate.

SGW-06: Estimate when a job submitted to a community resource is likely to finish

A **science gateway developer** needs to enable a gateway to inform gateway users when to expect their computation tasks will finish. These tasks are submitted to community resources where either the gateway operator or the gateway user has an allocation. Tasks will likely not begin immediately due to competing uses of the resource and scheduling policies.

The developer prefers a software interface that can be built into the gateway application. The application can use this interface to obtain a prediction of when a given computation task on a given community resource will start running, how long the task is likely to take, and/or when it is expected to complete.

In most cases, the developer wants to experience it as follows.

1. First, the developer visits the community website and looks in the section for science gateway developers for something related to queue or compute time prediction. He/she should find documentation explaining the service, how to use it in science gateways, what to expect from the service (availability and quality expectations), and how to access an SDK for the service.
2. Then, the developer integrates the SDK into his/her gateway application by configuring the SDK and adding application code calling the SDK's interface. The developer expects to call the SDK when the gateway has formulated a computation task, immediately before and/or immediately after the task has been submitted to the resource. SDK calls are expected to include information about the task similar to those included with a task submission: target resource, queue, requested computation resources, expected execution duration etc. The response should be an estimate of the queue time and execution time (or completion time) for the task.
3. If the developer has difficulty or technical questions during Step 2, he/she will submit a ticket to the community's user support service and expect a helpful and timely response.

4. If the SDK does not behave as expected during the gateway's operation, the developer (or the gateway's operator) will submit a ticket to the community's user support service and expect a helpful and timely response.

We'll take any solution, as long as the following are true.

1. The documentation in Step 1 is openly available without requiring the developer to identify or authenticate him/herself.
2. The development environments for which SDKs are available include Python and Java.
3. The SDK is freely available for academic and research use.
4. When tasks are submitted as described in SDK requests, the actual times correspond reasonably well to the estimated times provided by the SDK.

History

	Version	Date	Changes	Author
Entire Document	0.1	08/23/2012	First Version submitted to A&D	ECSS-Gateway Team
Refined UCSGW 1.0 and UCSGW 2.0 and UCSGW 3.0.	0.2	08/30/2012	Addressed Architect Comments on v0.1 and clearly scoped the use cases.	ECSS-Gateway Team
Refined UCSGW 1.0 and UCSGW 2.0 and UCSGW 3.0.	0.3	09/24/2012	Addressed Architect Comments on v0.2; reordered the use cases; added references to campus bridging use cases.	ECSS-Gateway Team, Craig Stewart
Added UCSGW 4.0 and UCSGW 5.0	0.4	10/25/2012	Added 4 and 5 use cases finishing the first phase of gateway use cases.	ECSS-Gateway Team
Entire document	1.1	08/22/2019	Added SGW-06; reformatted SGW-01 to -05 to XSEDE-2 format; standardized terminology with other use case areas; and removed unnecessary XSEDE terminology	L. Liming