The Art of Conversation with CrayPort

Bidirectional Record Management

Daniel Gens, Owen James, Elizabeth Bautista, Melissa Abdelbaky
National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory
Berkeley, California, USA
{dygens, o1james, ejbautista, mromanus}@lbl.gov

***Abstract* — The National Energy Research Scientific Computing Center (NERSC) is the primary scientific computing facility for the Office of Science in the U.S. Department of Energy. NERSC houses top-ranked Cray supercomputers, and staff works very closely with on-site Cray engineers, submitting on average 75 Cray cases each month. Until recently, submitting a Cray case was done by phone or email, and all subsequent updates were delivered manually. This process caused delays, errors during manual data entry, and increased incident processing and resolution time. In 2018, NERSC deployed an API-based bidirectional integration that allowed submitting and updating Cray cases directly from one Incident Management platform, thus streamlining 24x7 operations and enhancing communication between engineering teams.**

As the first site to implement this solution, we will share our development and deployment experience, as well as some report data. This solution is extendable to any Incident Management platform.

Keywords-incident management; bidirectional; integration; Cray case; streamline; deployment; implement API

I. INTRODUCTION

The National Energy Research Scientific Computing Center (NERSC) is the primary unclassified computational facility for the Office of Science in the U.S. Department of Energy (DOE). Serving approximately 7,000 scientists working on at least 700 research projects across the 50 states and collaborations globally, NERSC accelerates scientific discovery through high performance computing and data analysis spanning a wide range of scientific disciplines. NERSC's current system is Cori, a ~12,000 node XC-40. Perlmutter, the upcoming system to be delivered in 2020 will have over 3 times the compute power of Cori and contain a mixture of CPU-only and GPU-accelerated nodes.

A key challenge of operating these large-scale computing systems is the number of potential system issues that can arise due to their size and complexity. Efficient management, response, and documentation of these issues is critical in delivering high-availability and functionality of the systems to NERSC users. Resolving them and minimizing their impact on other machine components, services, and users requires seamless communication and joint collaboration between NERSC and Cray engineering. In the current production environment, NERSC engineers work very closely with on-site Cray engineers, submitting an average of 75 machine support cases each month. At extreme-scales, the number of potential system issues and the rates at which they occur will increase tremendously, making coordination more challenging and driving the need for new service management tools capable of meeting the increased demand.

Existing tools for issue tracking and/or incident management in production IT environments, such as ServiceNow [1], Salesforce Service Cloud [2], Zendesk [3], Atlassian JIRA[4], etc., play a vital role in capturing system and data center issues. They provide a centralized view within a company or organization of what happened, what actions were taken, what systems or services are impacted, who responded to this issue, and more. At NERSC, employees from two organizations, i.e., Cray and NERSC, are involved in collaboratively resolving issues on the same high-performance system yet each organization maintains a separate platform for internal incident management — specifically, NERSC uses ServiceNow while Cray uses a Salesforce-based platform called CrayPort. Thus, the same issue results in two incident reports - one that a Cray engineer opens and can update in CrayPort and one that NERSC staff opens in ServiceNow.

Traditionally, updating these tickets was a manual process that employees at each respective organization would be responsible for supporting. Further, Cray provides limited external access to CrayPort by their customers for security and privacy purposes, so not all members of NERSC staff had the ability to create and update tickets on the CrayPort system. This created a number of challenges related to synchronizing and communicating relevant information between organizations, as well as coordination of resolution strategies. It is also subject to a high probability of human-

[1] "ServiceNow." https://www.servicenow.com/. Accessed 12 Apr. 2019.
[2] "Service Cloud - Salesforce." https://www.salesforce.com/products/service-cloud/overview/. Accessed 12 Apr. 2019.
[3] "Zendesk." https://www.zendesk.com/. Accessed 12 Apr. 2019.
[4] "Jira – Atlassian." https://jira.atlassian.com/. Accessed 12 Apr. 2019.

error and miscommunication, leading to increased incident processing and problem resolution times. This method of coordinating between the two organizations is not scalable to larger systems with greater numbers of incidents and is not well-suited for tracking relationships between multiple incidents. It is important to note that these challenges are not unique to the relationship between NERSC and Cray but rather are relevant to Cray's relationships with any external customer that needs to synchronize incident management between CrayPort and their own internal incident management system.

To address these challenges, Cray announced plans to develop a RESTful API for the CrayPort platform that would enable customers to directly interact with the Cray incident management system via HTTP POST and GET requests and invited NERSC developers to participate in an open collaboration on desired features and mechanics from the alpha stage of the project. Based on this collaboration, NERSC developers created a second API to expose their incident management system, ServiceNow, by leveraging components built-in on the ServiceNow framework and customizing them in the context of the CrayPort API. The result was an API-based bidirectional integration between the NERSC ServiceNow and the CrayPort systems. The NERSC development process went hand-in-hand with the development of the CrayPort API, providing a unique opportunity to adapt and improve the code as Cray developers introduced additional features and new design ideas.

The first release of the NERSC ServiceNow API with CrayPort integration occurred in September 2018 and two subsequent releases have been deployed since then that include features developed through requests by staff who use the platform. Although the existing solution is specific to ServiceNow, and CrayPort, the technology stack and the integration architecture is extensible to other incident management solutions or API-capable databases, allowing other DOE organizations to use this integration in their workflows with modest modification to the code.

This paper describes the software design process of the NERSC ServiceNow API and explores the ways in which the collaboration with Cray drove the technology choices. Section II provides the motivation for the project and explains the challenges that exist in managing incidents across multiple organizations. The design and implementation of the API and the bidirectional integration between ServiceNow and CrayPort is discussed in Section III. Section IV highlights the key features of the integration, while Section V reports the results observed since the integration's deployment. A brief conclusion is included in Section VI.

II. Challenges and Background in Cross-Organization Incident Management

Customarily, in response to an issue, site-reliability engineers at NERSC created a ServiceNow ticket - this process involved manually filling out multiple fields, copying information from emails, files, and logs into the ticket, and submitting it to the ServiceNow database. Once a NERSC ServiceNow ticket number is generated, the incident is reported to Cray through the call center or through sending an email with a problem description, referencing the ServiceNow ticket number. The ServiceNow incident is updated with the Cray case number.

During the issue resolution process, NERSC and Cray staff who have access to both incident management systems had to update their own ticket first before copying the information into the other ticket system. Outside of both systems, when additional information came through email, logs, or any other format, that information would need to be copied into both tickets to sync the information. Figure 2.1 provides an illustration of this workflow.

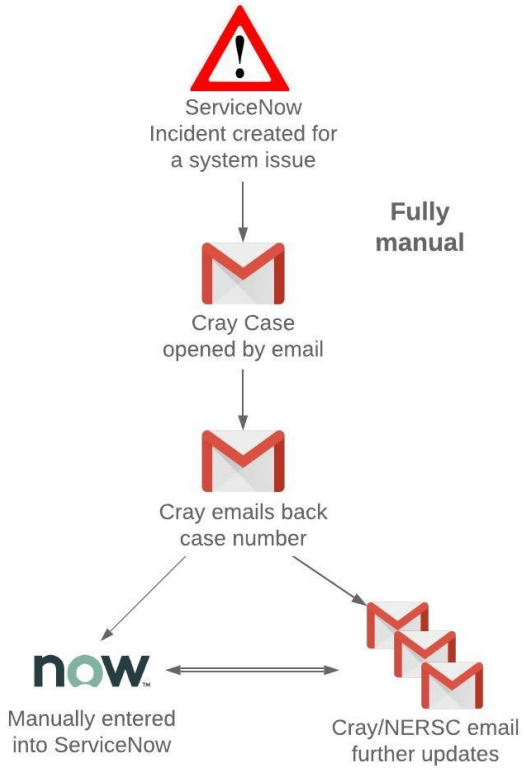

Figure 2.1 Illustration of the manual workflow

A. ServiceNow Incident Management System

ServiceNow is a Platform-as-a-Service (PaaS) offering that provides IT, customer, and employee workflow management tools to businesses around the globe. It is

highly-configurable for custom enterprise applications and built for scale. At NERSC, it is utilized for its IT service management tools, such as ticketing, asset tracking, managing some workflows and reporting.

Following a PaaS model, ServiceNow is focused on providing developers with the tools required to rapidly create and deploy custom applications that meet the needs of their organizations. Part of meeting this need is a strong focus on programmability and extensibility. Figure 2.2 provides a sample ecosystem of the ServiceNow Platform that can be modularly included into custom applications. The platform uses a MariaDB java driver and exposes the various components to developers via standard tools and languages, such as JavaScript, for building custom applications against it.

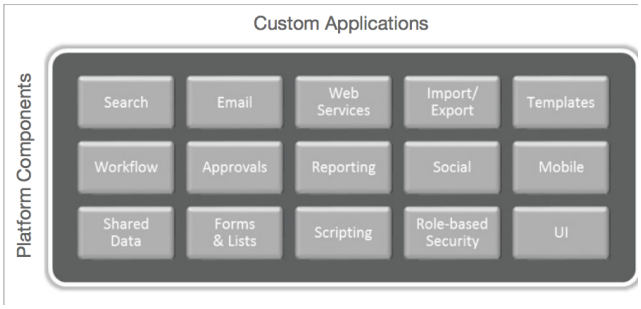

Figure 2.2 Sample of ecosystem showing commonly used components and capabilities[5]

B. CrayPort Incident Management System & the CrayPort API

The CrayPort Incident Management system is a Salesforce-based IT service management platform operated by Cray. CrayPort is the customer support portal for Cray and it is meant to service all of Cray's customers over the life cycles of their various machines. In CrayPort, incidents are called "cases" while specific machines are referred to as "assets." Recently, Cray created a web-based RESTful API, called the CrayPort API, that provides end-users with the ability to create, access, update, and manage CrayPort cases through HTTP POST and GET triggers.

The CrayPort API can be used to perform a number of tasks by allowing end-users to interact with the following objects:

1. Assets - to view asset information, such as the name, type, and serial number of the asset.
2. Attachments - to add attachments to Cases, and to get a specific attachment or a list of attachments for a specific case.
3. Cases - to view, create and update issues reported to Cray.
4. Comments - to viewing and adding comments to Cases.
5. Part orders - to view part order details, such as the status, quantity, and part description.
6. Shipping details - to view shipping details of part orders.

Attachment objects in CrayPort are represented as in the following model. The attachment body is converted to a Base64 encoded string.

```
CrayPortAttachment {
Id (string),
CaseNumber (string),
FileName (string),
Description (string),
MimeType (string),
Body (string): Base64 encoded string.
}
```

Supported HTTP verbs and endpoints are:

- GET by Attachment Id - Returns a specific attachment
- GET by Case Number - Returns a list of attachments for a specific Case
- GET Body - Returns the Attachment Body as Base64 encoded string
- POST - Creates an attachment

Case objects in CrayPort represent database records for issues reported to Cray, and interactions with Cases make up the majority of our calls to the CrayPort API. Case objects are described with the following model:

```
CrayPortCase {
   CaseNumber (string),
   Status (string),
   IsClosed (string),
   Subject (string),
   Description (string),
   ContactEmail (string),
   AssignedTo (string),
   Asset (string),
   Type (string),
   Priority (string),
   PriorityChangeReason (string),
   Product (string),
   Component (string),
   Version (string),
   BugType (string),
   BugNumber (string),
   BugFixedIn (string),
   ExternalReference (string)
}
```

Supported HTTP verbs for Case objects are:

- GET all cases - Return a list of all cases

[5] ServiceNow Platform Technical overview: https://community.servicenow.com/community?id=community_article&sys_id=573d2ee5dbd0dbc01dcaf3231f9619ac accessed April 11, 2019

- GET by Case Number - Returns a specific case
- POST - Creates a case
- PATCH - Updates case fields for a specific case

Comment objects in CrayPort API are described with a lean and simple model, where the comment is identified by a unique string:

```
CrayPortComment {
   Id (string),
   CaseNumber (string),
   Body (string)
}
```

CrayPort API supports the following HTTP requests for the Comment objects:

- GET by ID - Returns a specific comment
- GET by Case Number - Returns a list of comments for a specific case
- POST - Creates a comment

III. NERSC SERVICENOW <-> CRAYPORT INTEGRATION

The goal of the collaboration between Cray and NERSC was to provide a solution for the challenges faced in sharing information between separate organizations with different incident management systems. Toward this goal, Cray created the CrayPort API, which served as a catalyst in enabling this work. While the CrayPort API provided the means for NERSC staff to interact with CrayPort via web requests, it did not solve the issue of a human-in-the-loop having to trigger these requests.

This work aims to make the process of sharing information between the two incident management systems automatic, thereby ensuring that information is synchronized between NERSC and Cray and allowing engineers to focus on joint resolution of the issue. To achieve this goal, NERSC created the ServiceNow API and an architecture for the bidirectional integration of this API with the CrayPort API. This section details the ways in which the CrayPort API influenced the design of the bidirectional integration, introduces the NERSC ServiceNow API, and presents the architectural overview of the bidirectional integration.

A. NERSC ServiceNow API

The NERSC ServiceNow API leverages the capabilities provided by the CrayPort API, as well as those from the ServiceNow platform. Specifically, ServiceNow provides extensive integration capabilities through Web services, including the ability to build and expose custom API endpoints. In addition to utilizing the custom API endpoints, we were also able to leverage ServiceNow's default backend database for the client and server functions. The ServiceNow Platform backend is a MySQL database that uses GlideRecord - a special Java class that can be used in JavaScript exactly as if it was a native JavaScript class. GlideRecord is used for database operations instead of writing SQL queries and is essentially an ordered list containing records and their fields. Client software was written in JavaScript / jQuery, with web pages styled using HTML+CSS. Server-side development was performed in JavaScript, using extendable components of ServiceNow PaaS. [2]

1) Interacting with the CrayPort API Webhooks

Using the GET and POST HTTP methods exposed by the CrayPort API, NERSC developers implemented corresponding event-driven requests for these endpoints on the ServiceNow instance for specific actions on the instance. Similarly, CrayPort API developers also set up a number of webhooks for outgoing updates from CrayPort. Webhooks enabled us to receive notifications when a monitored CrayPort event occurs, such as when a new comment or an attachment is added to an existing Cray case, or when an existing Cray case is updated with new Priority, Subject, or Description. When a CrayPort event triggers a webhook to fire, an HTTP POST request will be sent to each configured callback URL. A webhook would be triggered when any existing CrayPort API object is updated or when any new CrayPort API object is created.

2) NERSC ServiceNow API endpoints for CrayPort incoming webhooks

To process the messages coming from CrayPort webhooks, we configured separate callback URLs per CrayPort API object on NERSC ServiceNow instance and implemented logic to process the requests. We currently have no requirement to monitor Case creation, only updates to existing cases, so our API endpoints accept and process requests with specific payloads for three objects:

1) CrayPortCase
   a) Update
2) CrayPortComment
   a) Create
   b) Update
3) CrayPortAttachment
   a) Create
   b) Update

3) NERSC ServiceNow webhooks

To develop the ServiceNow webhooks, NERSC engineers leveraged a ServiceNow capability, called Business Rules. Business Rules allow developers to trigger actions when incidents in ServiceNow are queried, updated, inserted, or deleted, according to a series of developer-defined rules. As part of the API development, we developed Business Rules for server-side events, and Client Scripts and UI Policies for client-side events. If the rule condition is met, the code associated with it is executed. For example, when a ServiceNow Incident linked to a Cray case has a new public

comment, a Business Rule called "Update Cray case with new Comment" triggers associated code that submits the new comment to CrayPort through the CrayPort API.

In the NERSC ServiceNow API, there are two types of webhooks: fully automatic and user action triggered.

The fully automatic webhooks are as follows:

- Update Cray case with new Comment:
  - Condition: When ServiceNow Incident linked to a Cray case has a new public comment
  - Code: Calls `addComment()` to submit the new comment to CrayPort through the CrayPort API
- Update Cray case with new Attachment:
  - Condition: When the ServiceNow Incident linked to a Cray case has a new attachment
  - Code: Calls `addAttachment()` to submit the new comment to CrayPort through the CrayPort API
  - Condition: User clicks "Change Cray Case priority" UI button in the Incident form
  - Code: Calls `changePriority()` to update the Cray case record in CrayPort through the CrayPort API. User is required to submit a Priority Change Reason.

See Appendix B for the source code for user-triggered webhooks.

B. Bidirectional Integration Architecture

Figure 3.1 shows the architecture of the bidirectional integration and the interaction between the two incident management systems as facilitated by the two new APIs. Authentication between the two endpoints is discussed in the following section.

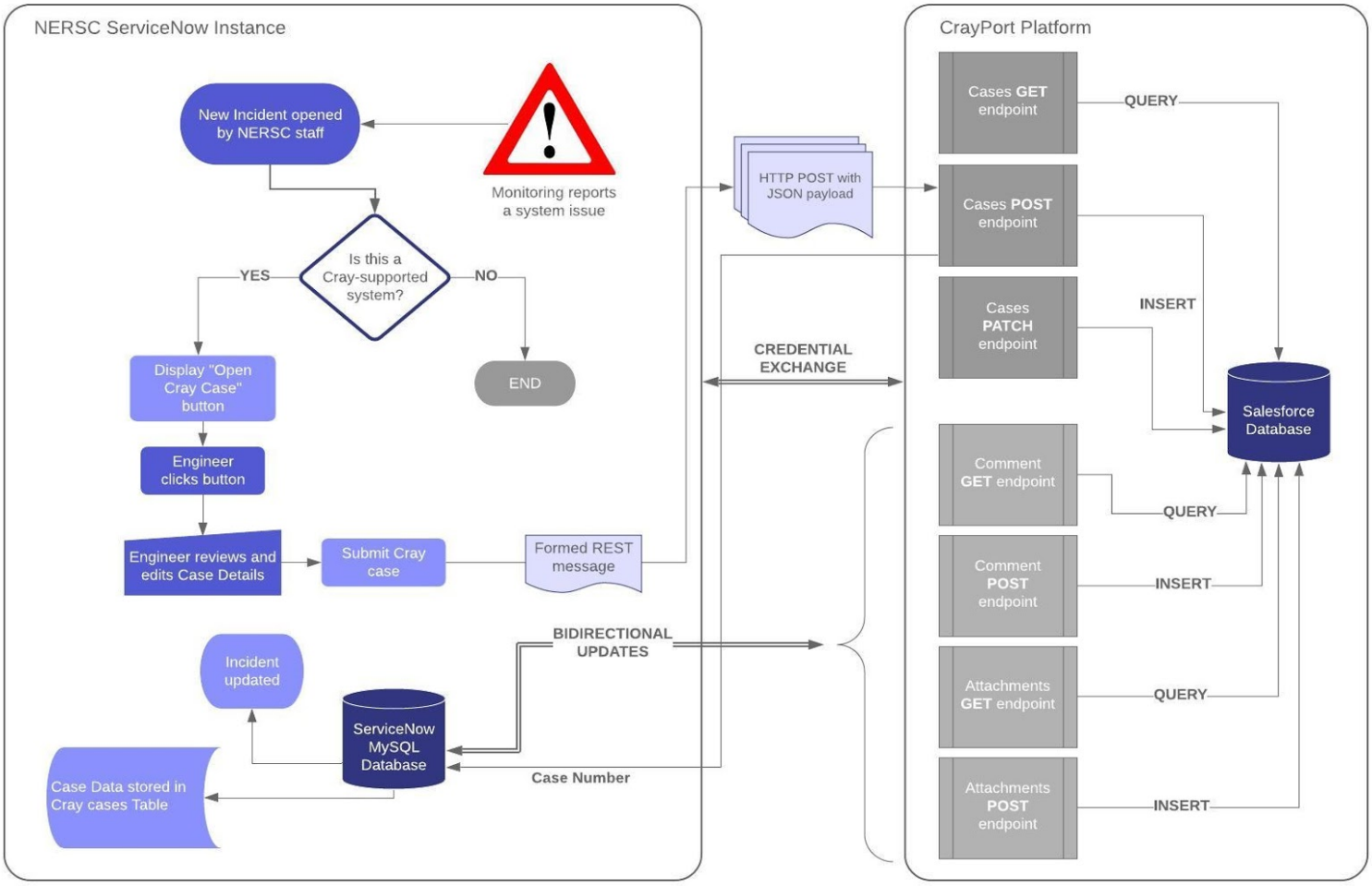

Fig. 3.1 Architectural Overview of Opening a Cray case through ServiceNow using CrayPort API

The current features enabled by this architecture include the ability to perform the following from the ServiceNow platform:

- Open a new Cray case and automatically link both records
- Link an existing Cray case to an existing ServiceNow record
- Change a Cray case priority (e.g., escalation to Critical case)
- Perform shared bi-directional updates between ServiceNow and CrayPort

1) *Authentication*

The CrayPort API uses Basic Access Authentication to ensure that all the submitted requests are authenticated and authorized. HTTP Basic authentication (BA) implementation enforces access controls to web resources. It does not require cookies, session identifiers, or login pages; rather, it uses standard fields in the HTTP header, removing the need for handshakes. [4] When a request is coming from NERSC ServiceNow, it uses the credentials for a specially created NERSC API User in CrayPort.

NERSC ServiceNow API endpoints also use Basic Access Authentication but in combination with built-in role-based security that is more restrictive than a normal NERSC staff user. All this category of user can do is send HTTP requests to ServiceNow API endpoints. Even if the credential set is compromised, the attacker would not be able to access any information on the ServiceNow Platform.

C. *Adaptability and Extensibility*

Although the existing solution is specific to the two platforms, the technology stack and architecture of the integration could be adapted for other incident management solutions or an API-capable database, allowing other DOE organizations to use this integration in their workflows with modification to the code.

First, organizations would need to expose their own API endpoints to act as callback URLs for CrayPort webhooks. The callback URLs can be configured as one URL for all CrayPort API objects, or as a separate URL per CrayPort API object. To define and configure callback URLs, Cray suggests that organizations work with the CrayPort API administrator. The source code of how NERSC ServiceNow API processes the incoming CrayPort webhooks is in Appendix C.

Second, modifications will need to be made in order to interact with a database that does not support GlideRecord. The query conditions can remain the same, but the way these queries are made will have to change for databases outside of ServiceNow. Also, developers will need to map field names to the ones used in their organization.

Third, the REST message framework in ServiceNow is not pure JavaScript, but uses the Platform's RESTMessageV2 class. However, JavaScript can be easily used to connect to an API with native classes, for example XMLHttpRequest. All REST calls will need to be adapted to use a chosen JavaScript / jQuery / AJAX framework.

The full code can be found here: http://tinyurl.com/nersc-sn-crayport. Inquiries can be sent to *sn-crayport-dev@lbl.gov*.

IV. Key Features of the NERSC ServiceNow – CrayPort Integration

In this section, we provide an overview of the key features enabled by this work. The current features of the integration provide a seamless and secure API-based workflow for incidents that require joint troubleshooting with Cray field engineers. If the issue requires a Cray case to be opened, a NERSC engineer can click a custom User Interface (UI) button in the Incident form, fill out minimal required fields and submit the Cray case.

NERSC's custom API exposed specific endpoints of the ServiceNow instance to CrayPort. Submitting a Cray case through the ServiceNow interface generates a REST call to an exposed CrayPort API endpoint, passing the necessary information from ServiceNow in a JSON payload. The CrayPort API will respond with a Case Number and other information directly populated in predetermined fields in the ServiceNow Incident. After this initial exchange, any updates from the ServiceNow Incident will be submitted as a comment to the Cray case using REST calls to CrayPort API endpoints. See Appendix D for the sample source code.

Conversely, any updates made to the Cray case within CrayPort triggers an HTTP request to the configured webhooks to ServiceNow. Thus, all updates are to be synchronized and shared bidirectionally between ServiceNow Incidents and Cray cases. This new API-based automated workflow allows NERSC engineers to submit and continuously update Cray cases through a single interface without duplicating information. The API also allowed us to implement new features to improve incident management with Cray, such as sharing attachments, updating severity levels, and closing cases. See Figure 4.1 for a graphic of the new submission workflow.

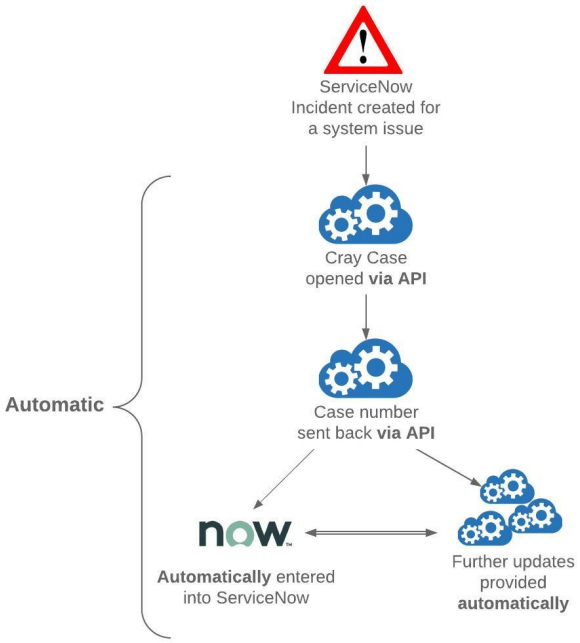

Figure 4.1 Demonstrates the new submission workflow.

Additional Features Beyond Incident Reporting

After the Cray case is linked to a ServiceNow Incident by either opening a new Cray case or linking an existing Cray case to a ServiceNow Incident, certain updates between them are shared. From CrayPort, this includes Priority, Subject and Description updates, new case Comments, and new Attachments. From ServiceNow, it includes new Incident public comments and case Priority updates.

1) Changing Cray Case Priority from ServiceNow

NERSC engineers are able to change Cray case Priority via the "Change Case priority" UI button, and the new priority is sent to CrayPort API via an HTTP PATCH request.

2) Changing Cray Case Priority in CrayPort

When Cray engineers change Cray case Priority, the new priority and the Priority Change reason are sent to a ServiceNow API endpoint, saved in the Cray case record and posted in the ServiceNow Incident.

3) Case details updated in CrayPort are propagated to ServiceNow

When Cray engineers make changes to case Subject or Description in CrayPort, this information is sent to a ServiceNow API endpoint, saved in the Cray case record and posted in the ServiceNow Incident.

4) Bidirectional Comment Updates (public comments)

Once a ServiceNow Incident is linked to CrayPort case, all public comments are propagated across both platforms. If the Cray case is opened from within ServiceNow, the first comment becomes the case description and all subsequent comments becomes comments in the Cray Case. If a ServiceNow incident is linked to an existing Cray case, all comments are sent to CrayPort as comments.

5) Linking an existing Cray case to a ServiceNow Incident

If a Cray case has been created directly in the CrayPort web interface, it would not be associated with a ServiceNow Incident automatically. A NERSC engineer needs to use the "Link Cray case" UI button in the ServiceNow platform to make that connection by entering the Cray case number in a pop-up window. The Cray case number goes through a validation script: first to check that the case is not a duplicate one and a record with the same number does not already exist in the ServiceNow Cray cases table; second to send an HTTP GET request to the CrayPort API to check that the case exists. Then, if the case number is valid, the Cray cases table is updated with that ServiceNow Incident number and details received in the response to the GET request during the validation stage.

V. RESULTS

This new workflow has demonstrated the following results:

- Streamlined and minimized the need for human-based data entry and duplicating of information which is a process that took the most time and is the most error-prone. In 2018, NERSC opened 674 Cori cases and 230 Edison cases. Based on 15 minutes to open and verify a case (open in ServiceNow, notify Cray call center and obtain a Cray case number) having this integration in place for the entire year could have saved 180 hours of staff time. It currently takes two minutes to open a new case because NERSC can provide the Cray case number prior to the phone call.

- Real-time, automated record updating to both platforms keeping them in sync.

- Reduced incident response time - A Cray engineer is notified of an incident twenty minutes sooner with the new workflow.

- Improved communication between engineering teams - Copy and pasting to ensure all communication (Slack, e-mail, phone, in-person) is

updated in two platforms can cause issues when not in sync. The automatic syncing process of the integration allows an engineer to only update one platform and be confident that they will be in sync preventing any misunderstanding of status, closed cases and tracking on-going issues.

- Facilitated reporting and incident review - Having a view into both platforms, allows us to correlate reports. For example, previously staff would need to engage with Cray to get monthly reports like how many critical cases were opened per month and for which asset. The integration allows us to create those reports from within the ServiceNow platform rather than getting them from Cray. Further, we can correlate them with other information available like incidents, Cray cases with priority, when they were reported, what is the current status.

- The collaborative nature of this solution has created a template for potential future interactions with Cray or other vendors on other areas moving forward as described in section III – adaptability and extensibility. When presented at the Cray Quarterly Business Review in January 2019, the software garnered interest from other data centers with Cray systems, as well as centers that work closely with third-party vendors that provide an API for their customer service platform.

VI. Conclusion

This innovative ServiceNow CrayPort integration allows real-time, automated record updating between two separate and disparate service management platforms. There is significant time reduction in reporting Cray cases, a more efficient process to keep information in sync, the process has improved communication between both organization's staff and there is a potential to share this new process with other organizations.

Acknowledgment

The authors would like to acknowledge the following staff at Cray, Inc.:

Development Team:

- Chris Solin, IT Developer
- Johnathan Miller, IT Developer
- Jason Flackey, Director, Application Systems & Architecture

Cray Site Team:

- Mark Green, Site Service and Support Manager
- Wendy Stresau, District Service Manager

This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02- 05CH11231.

References

[1] ServiceNow Platform Technical overview: https://www.servicenow.com/content/dam/servicenow-assets/public/en-us/doc-type/resource-center/white-paper/wp-sn-platform-technical-overview.pdf accessed April 11, 2019

[2] ServiceNow Glide documentation docs.servicenow.com/bundle/geneva-servicenow-platform/page/script/general_scripting/reference/r_GlideStack.htm Accessed April T1, 2019

[3] ServiceNow Platform Technical overview: https://www.servicenow.com/content/dam/servicenow-assets/public/en-us/doc-type/resource-center/white-paper/wp-sn-platform-technical-overview.pdf Accessed April 11, 2019

[4] ServiceNow Platform Technical overview: https://community.servicenow.com/community?id=community_article&sys_id=573d2ee5dbd0dbc01dcaf3231f9619ac Accessed April 11, 2019

APPENDIX A: NERSC SERVICENOW API SOURCE CODE FOR AUTOMATIC WEBHOOKS

```
addComment: function(caseNumber, comment) {
        // Trying to submit a CrayPortComment to CrayPort API
        try {
                // Create a new REST message for HTTP POST request
                var r = new sn_ws.RESTMessageV2('x_ners2_crayport.Comments', 'POST new  comment');
                // Check ServiceNow instance info - use debug values for test and dev
                this._checkInstance(r);

                // Populate case number
                r.setStringParameterNoEscape('case', caseNumber);
                // Populate comment body
                r.setStringParameterNoEscape('comment', this._escapeQuotes(comment));

                // Send the REST message
                var response = r.execute();
                // Get a JSON response and parse it to get response body
                var responseBody = response.getBody();
                // Parse response to get HTTP status code
                var httpStatus = response.getStatusCode();
                // If not “success”, log error
                if (!(httpStatus == "200" || httpStatus == "201"
                || httpStatus == "202" || httpStatus == "204"))
                gs.error(httpStatus + " - " + responseBody);
                // No need to do anything if “success”

        // If try fails, log exception and error message
        catch(ex) {
                var message = ex.getMessage();
                gs.error(message);
        }
},

addAttachment: function(caseNumber, mimeType, body, fileName, description) {
        gs.info("Adding attachment: " + fileName);
        // Trying to submit a CrayPortAttachment to CrayPort API
        try {
                // Create a new REST message for HTTP POST request
                var r = new sn_ws.RESTMessageV2('x_ners2_crayport.Attachments', 'POST new attachment');
                // Check ServiceNow instance info - use debug values for test and dev
                this._checkInstance(r);

                // Populate attachment payload
                r.setStringParameterNoEscape('number', caseNumber);
                r.setStringParameterNoEscape('mimetype', mimeType);
                r.setStringParameterNoEscape('body', body);
                r.setStringParameterNoEscape('filename', fileName);
                r.setStringParameterNoEscape('description', description);

                // Send the REST message
                var response = r.execute();
                // Get a JSON response and parse it to get response body
                var responseBody = response.getBody();
                // Parse response to get HTTP status code
                var httpStatus = response.getStatusCode();
                // If not “success”, log error
                if (!(httpStatus == "200" || httpStatus == "201"
                || httpStatus == "202" || httpStatus == "204"))
                gs.error(httpStatus + " - " + responseBody);
                // No need to do anything if “success”

                }
        // If try fails, log exception and error message
        catch(ex) {
                var message = ex.getMessage();
                gs.error(message);
        }},
```

APPENDIX B: NERSC SERVICENOW API SOURCE CODE FOR USER-TRIGGERED WEBHOOKS

```
changePriority: function() {
        // Get Cray case ID from the Cray cases table
        var case_sysID = this.getParameter('sysparm_case_sysID');
        // Get the new case priority
        var priority = this.getParameter('sysparm_priority');
        // Get Priority Change Reason
        var reason = this.getParameter('sysparm_reason');

        // Get the Cray case record by ID
        var cr = new GlideRecord('x_ners2_crayport_cray_case');
        cr.get(case_sysID);

        // Get the Incident record by ID
        var gr = new GlideRecord('incident');
        gr.get(cr.parent);

        try {
                // Create a new REST message for HTTP POST request
                var r = new sn_ws.RESTMessageV2('x_ners2_crayport.Cases', 'Update case PATCH');
                // Check ServiceNow instance info - use debug values for test and dev
                this._checkInstance(r);
                // Populate the new priority
                r.setStringParameterNoEscape('priority', this._priorityToString(priority));
                // Populate case number
                r.setStringParameterNoEscape('case', cr.cray_case_number);
                // Populate priority change reason
                r.setStringParameterNoEscape('reason', this._escapeQuotes(reason));

                // Send the REST message
                var response = r.execute();
                // Get a JSON response and parse it to get response body
                var responseBody = JSON.parse(response.getBody());
                // Parse response to get HTTP status code
                var httpStatus = response.getStatusCode();

                // If “success”
                if (httpStatus == "200" || httpStatus == "201"
                || httpStatus == "202" || httpStatus == "204") {

                        // Update case priority in the ServiceNow Cray case record
                        cr.priority = priority;
                        cr.update();
                        // Put a note in the linked Incident on priority change
                        gr.work_notes = "[code]Priority changed for <b>Cray Case #" +
                        gr.cray_case_number +
                        "</b>:<br /><br /><blockquote><b>New Priority:</b> " +
                        cr.priority.getDisplayValue() +
                        "<br /><b>Priority Change Reason:</b> " + reason +
                        "</blockquote>[/code]";
                        // Update the Incident record
                        gr.update();
                        // Show success message to user
                        gs.addInfoMessage("Successful priority change for case "+ cr.cray_case_number + ".
                        New Priority: " + cr.priority.getDisplayValue());
        }
               // If not “success”, log error message and response
                else {
                    gs.error(httpStatus + " - " + responseBody);
                    gs.addErrorMessage("ERROR: Cray case Priority was NOT changed!");
                }
        }
        catch(ex) {
                var message = ex.getMessage();
                gs.error(message);
                gs.addErrorMessage("ERROR: Cray case Priority was NOT changed!");
        }},
```

APPENDIX C: NERSC SERVICENOW API PROCESSES THE INCOMING CRAYPORT WEBHOOKS

```
 (function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {
       // Gathering information from the request body that comes from CrayPort
       var requestBody = request.body;
       var requestData = requestBody.data;
       var commentObj = requestData.CrayPortComment;
       var comment = commentObj.Body;
       var caseNumber = commentObj.CaseNumber;
       var author = commentObj.MetaData.CreatedByName;
       if (!(author.match(/nersc api/i) || comment.includes("Priority Change Reason") ||
comment.includes("Case Close Reason"))) {
              // Querying Incident table to find Cray case that the comment update is for
              var gr = new GlideRecord('incident');
              gr.addQuery('x_ners2_crayport_has_cray_case', 'true');
              gr.addQuery('u_vendor_sr', caseNumber);
              gr.query();
              while (gr.next()) {
                     // Publish the new comment and update the Incident record
                     gr.work_notes = "[code]" + author +
                            " updated <b>Cray Case #" + caseNumber +
                            "</b> with a new comment:<br /><br /><blockquote>" +
                            comment + "</blockquote>[/code]";
                     gr.update();
              }
       }
})(request, response);
```

APPENDIX D: NERSC SOURCE CODE FOR USING REST CALLS TO CRAYPORT API ENDPOINTS

```
submit: function() {
        // Get Incident ID from the Incident table
        var inc_sysID = this.getParameter('sysparm_inc_sysID');
        // Get Cray case ID from the Cray cases table
        var case_sysID = this.getParameter('sysparm_case_sysID');

        // Trying to submit a CrayPortCase to CrayPort API
        try {
                // Get the Incident record by ID
                var gr = new GlideRecord('incident');
                gr.get(inc_sysID);

                // Get the Cray case record by ID
                var cr = new GlideRecord('x_ners2_crayport_cray_case');
                cr.get(case_sysID);

                // Sleep for 10 seconds
                var util = new global.MyGlobalScopeUtils();
                util.sleep(10000);

                // Create a new REST message for HTTP POST request
                var r = new sn_ws.RESTMessageV2('x_ners2_crayport.Cases', 'POST new case');
                // Populate case subject
                r.setStringParameterNoEscape('subject', this._escapeQuotes(cr.cray_case_subject));
                // Populate case type
                r.setStringParameterNoEscape('type', 'General Inquiry');
                // Populate case asset (serial number)
                r.setStringParameterNoEscape('asset', cr.cray_asset.serial_number);
                // Populate case description
                r.setStringParameterNoEscape('description', this._escapeQuotes(cr.description));
                // Populate case priority
                r.setStringParameterNoEscape('priority',
cr.priority.getDisplayValue().toLowerCase());
                // Populate external reference field in CrayPort with SN Incident number
                r.setStringParameterNoEscape('inc', gr.number);

                // Get email of the user who clicked the Open Cray Case button
                var email = gs.getUser().getEmail();
                // If user is in Operations Technology Group (OTG)
                if (gs.getUser().isMemberOf('OTG'))
                        // Use operator email instead of the user's personal email for OTG
                        email = 'operator@nersc.gov';
                // Populate contact email
                r.setStringParameterNoEscape('email', email);

                // Check ServiceNow instance info - use debug values for test and dev
                this._checkInstance(r);

                // Send the REST message
                var response = r.execute();
                // Get a JSON response and parse it to get response body
                var responseBody = JSON.parse(response.getBody());
                // Parse response to get HTTP status code
                var httpStatus = response.getStatusCode();

                // If "success"
                if (httpStatus == "200" || httpStatus == "201"
                        || httpStatus == "202" || httpStatus == "204") {
                        // Get Case Number from the response
                        var caseNumber = responseBody.CaseNumber;
                        // Put it in the Cray case table
                        cr.cray_case_number = caseNumber;
```

```
                        // Update the case record in the Cray case table
                        cr.update();

                        // Populate fields in the Incident record
                        gr.u_vendor_sr = caseNumber;
                        gr.u_vendor_notified = cr.opened_at;
                        gr.x_ners2_crayport_u_cray_case = case_sysID;
                        gr.x_ners2_crayport_has_cray_case = true;
                        gr.work_notes = "[code]Opened <b>Cray Case #" + caseNumber +
                                "</b>:<br /><br /><blockquote><b>Subject:    </b> " +
cr.cray_case_subject +
                                "<br /><b>Asset:      </b> " + cr.cray_asset.name +
                                "<br /><b>Description:</b> " + cr.description +
                                "<br /><b>Priority:   </b> " + cr.priority.getDisplayValue() +
                                "</blockquote>[/code]";
                        gr.state = 10;
                        //Update the Incident record
                        gr.update();

                        // Show success message to user
                        gs.addInfoMessage('Cray case ' + caseNumber + ' has been submitted. Please
RELOAD the page!');
                }
                // If not "success", log error
                else {
                        gs.error(httpStatus + " - " + responseBody);
                        gs.addErrorMessage("ERROR: Cray case was NOT submitted!");
                }
        }
        // If try fails, log exception and error message
        catch(ex) {
                var message = ex.getMessage();
                gs.error(message);
                gs.addErrorMessage("ERROR: Cray case was NOT submitted!");
        }
},
```