BROADCOM®
MAINFRAME SOFTWARE

CA 7 REST APIs

A Comprehensive Guide to Streamlining Workload Automation

White Paper

Table of Contents

Chapter 1: Introduction

In today's dynamic IT environment, efficient workload management is crucial. CA 7™ Workload Automation Intelligence offers a powerful REST API capability that allows you to seamlessly integrate and manage workloads through custom applications or popular workflow and ticketing systems.

Integrating the CA 7 REST API with your applications provides a powerful toolset to efficiently automate and manage workloads. Optimize workflow automation and achieve seamless operations by understanding and implementing the various API categories, ensuring robust security measures, and handling technical nuances like terminal resource management and transaction key usage. The ability to manage jobs, datasets, resources, and Agents, coupled with secure and reliable API interactions, empowers you to create more resilient and efficient automation solutions. Embrace these capabilities to enhance your workload automation strategy and drive better business outcomes.

CA 7 REST APIs are versatile like Lego blocks—you can combine and configure them in different ways to automate a wide range of CA 7-related use cases. Popular use cases include the following:

- Incident Management
  - Use CA 7 REST APIs to create incidents in ITSM solutions.
  - Use CA 7 REST APIs to issue actions into CA 7 to manually or automatically resolve job-related incidents directly from ITSM solutions like ServiceNow.
- Scheduling Management
  - GUI-based approach: Use ServiceNow forms and workflows, or other no-coding/low-coding tools, to build a lightweight GUI powered by CA 7 REST APIs for your teams, enabling them to schedule existing jobs or define new jobs.
  - Jobs-as-Code approach: This approach fully automates the scheduling process. Jobs are defined in a YAML or JSON configuration file that can be versioned and rolled back if needed. The configuration file moves in the CICD pipeline like any other piece of code, and finally, the jobs are created or scheduled using CA 7 REST APIs.
- Monitoring
  - Monitor job execution events using either the CA 7 REST APIs or OpenTelemetry-based integration on the platform of your choice.

This document explores the CA 7 REST API from several different perspectives:

- Key features, getting started, and exploring with Swagger
- API categories and specific APIs that are available
- Integration and security
- Technical nuances of using the APIs to optimize operations and resource efficiency

Chapter 2: Key Features of the CA 7 REST API

The CA 7 REST API provides several key benefits:

- **Maintain CA 7 Object Definitions:** The CA 7 REST API enables you to manage the definitions of various CA 7 objects, such as jobs, datasets, and resources. This flexibility ensures you can tailor CA 7 to meet workload requirements.
- **Command Execution and Information Retrieval:** With the REST API, you can issue commands directly to CA 7 and retrieve information about the active workload. This capability allows you to manage and monitor workloads efficiently in real time, ensuring optimal performance and quick resolution of any issues.
- **Agent Management:** The API provides functionalities to retrieve a list of Agents along with their current statuses. Additionally, you can access logs from these Agents, providing a comprehensive view of the workload environment and aiding in troubleshooting and maintenance.

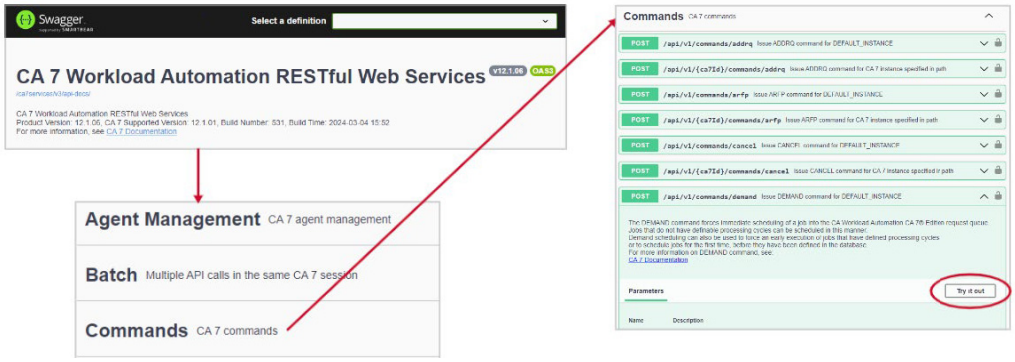
2.1 Getting Started

To begin using the CA 7 REST API, the best approach is to utilize Swagger, an interactive tool that simplifies API usage. Once the REST API is installed, Swagger becomes available, providing a user-friendly interface to explore and interact with the API endpoints.

The CA 7 REST API Server can be installed on Windows, zLinux/Linux, and UNIX System Services (USS). A CA 7 REST API Server may be installed for each CA 7 instance, or a single CA 7 REST API Server may be configured to communicate with multiple CA 7 instances.

2.2 Exploring APIs with Swagger

Figure 1: Example of CA 7 API Categories and an Expanded Category



With Swagger, APIs are organized into categories that allow you to quickly find the APIs you are looking for. Each category can be expanded to show the APIs related to that category, and each API can be expanded to show detailed information and usage instructions.

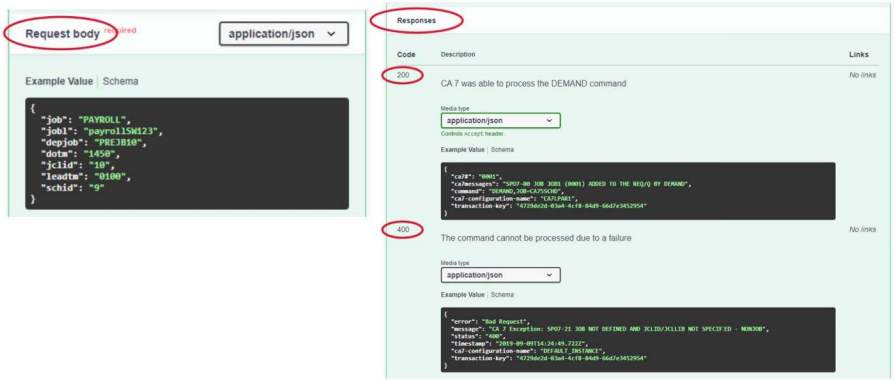
One of Swagger's most powerful features is **Try it Out**, which allows you to invoke an API to understand how it works. This interactive feature is invaluable for developers looking to test and understand API functionality in a real-world context.

Moreover, Swagger documentation is available offline, so you can see which API functions are available even before you install the REST API.

2.3 Understanding API Details in Swagger

Swagger provides comprehensive details about the input parameters that each API expects. For instance, if a request body is required, Swagger shows an example of the request body, including sample values for the properties. This helps developers understand how to effectively structure API calls.

Figure 2: Swagger Shows How to Use Each API



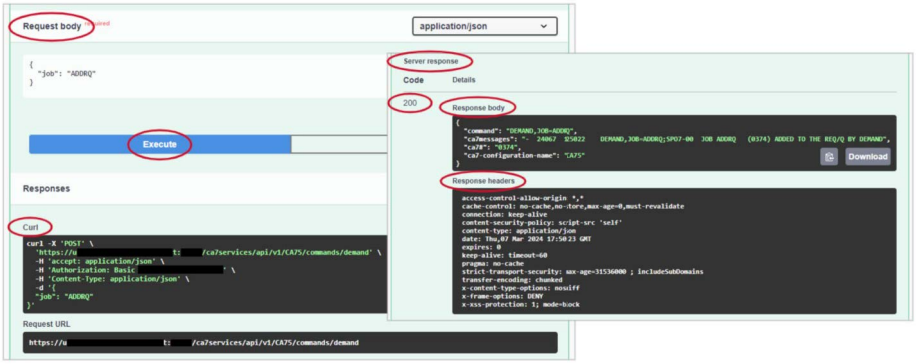
Swagger also displays sample responses that the application can expect to receive:

- A code 200 with a response body is returned if the request is successful.
- A code 400 with a response body containing an error message is returned if the request fails.

2.4 Using the Swagger Try it Out Feature

When you click the **Try it Out** button, Swagger allows you to specify the input parameters and execute the API request.

Figure 3: Invoke the API with Try it Out



In this example, the request body has been specified to include only the required demand property: job name.

When you click the **Execute** button, Swagger sends the request to the CA 7 REST API.

In the responses section, a sample Curl command contains all the information needed to invoke the API. This helps you understand how the application should construct the request.

Then you see the actual response from the REST API. The request was successful in this example, as shown by the 200 return code.

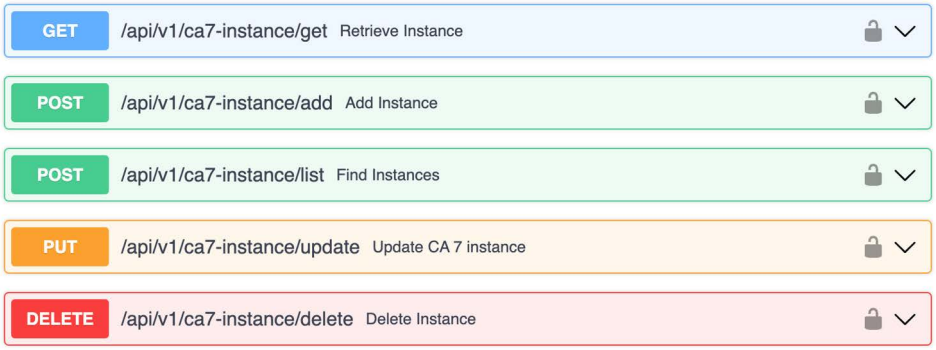
Chapter 3: CA 7 REST API Categories

NOTE: These APIs are for administrative users only.

3.1 CA 7 Instances

The **REST API CA 7 Instances** category contains APIs that allow you to maintain definition properties for the default CA 7 instance and any additional CA 7 instances you want to access. Security calls restrict these APIs to administrative users.

Figure 4: REST API CA 7 Instances Options



The CA 7 REST API can interact with one or more instances of CA 7. The following properties for the default instance are specified when the REST API is installed:

- Connection information for the CA 7 instance
- Connection information for the Datacom server
- Security settings
- Timeout values

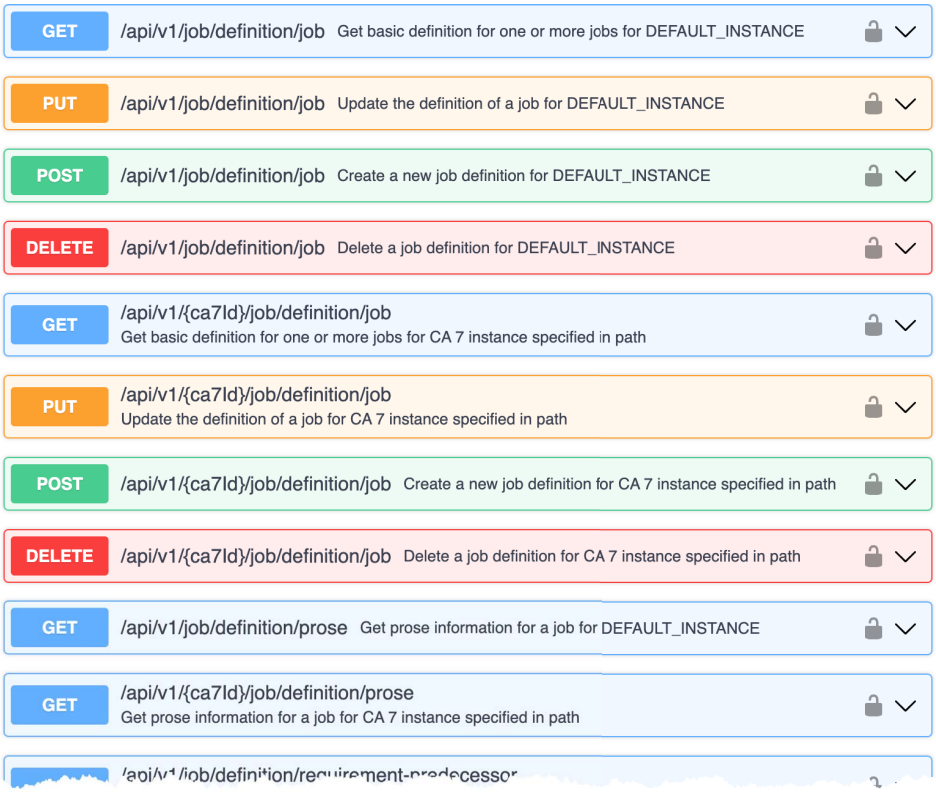
During installation, you can only configure one instance, the DEFAULT_INSTANCE. Additional CA 7 instances may be configured after installation using the ca7-instance endpoints category of REST APIs.

3.2 Job Definitions

Three API categories allow you to maintain object definitions in CA 7: job definitions, datasets, and resources. The job definitions category contains APIs that allow you to maintain CA 7 jobs, and includes the following:

- The job definition itself
- Definitions of predecessors and successors
- Definitions of triggers, schedules, resources, and more

Figure 5: Job Definitions

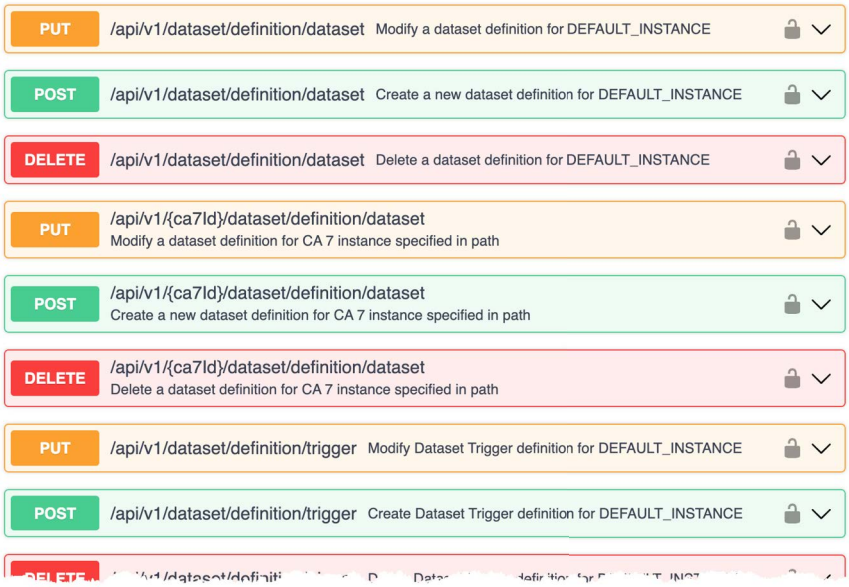


3.3 Datasets

This category contains APIs that allow you to maintain CA 7 datasets, including the following:

- The dataset definition itself
- Trigger definitions

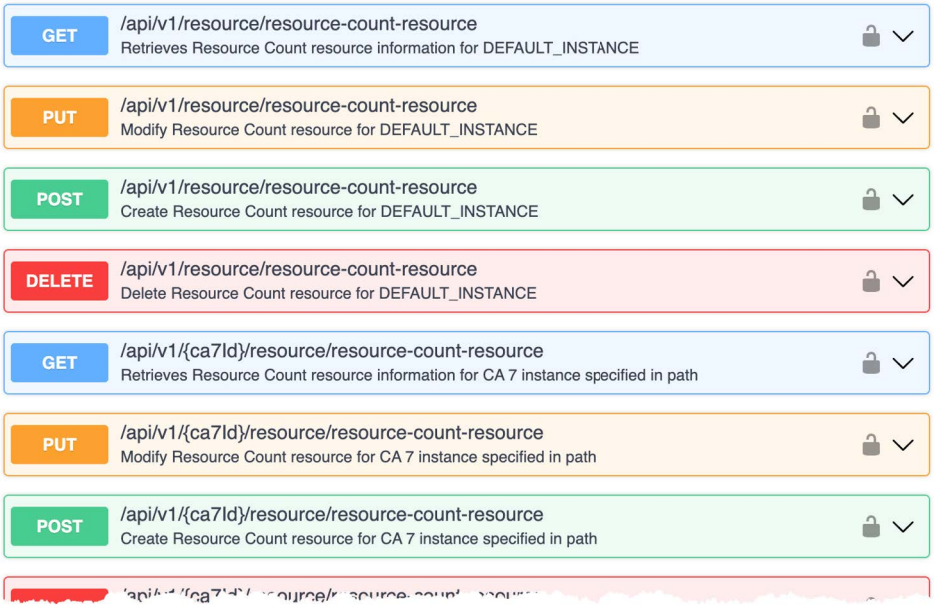
Figure 6: Datasets



3.4 Resources

This category contains APIs that allow you to maintain resource count resources.

Figure 7: Resources



3.5 Commands

The Commands category contains APIs that issue some common commands to CA 7 to manage the workload, and includes the following:

- **Demand:** Bring a job into the workload.
- **Cancel:** Cancel a job.
- **Hold:** Hold a job.
- **Release:** Release a job.
- **Restart:** Restart a job.

These APIs provide flexibility to manage a workload dynamically by issuing essential commands directly through the CA 7 REST API.

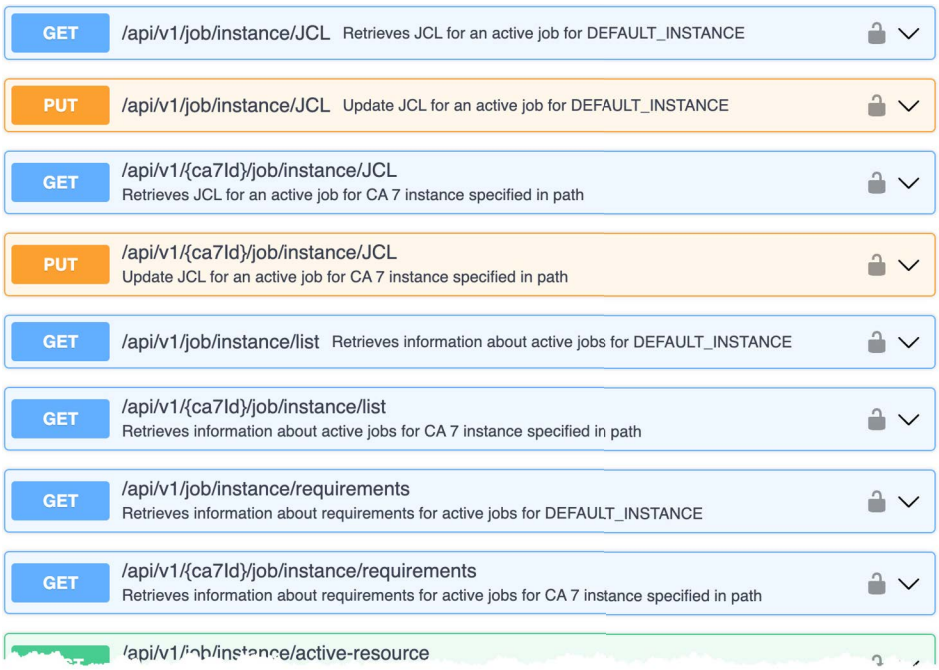
In addition to commands, the Job Instances and Job History categories round out the management of jobs in the active workload.

3.6 Job Instances

The Job Instances APIs provide the following capabilities:

- Retrieve information about the active jobs in the workload.
- Retrieve information about requirements for jobs in the active workload.
- Retrieve a job's JCL.
- Update a job's JCL.

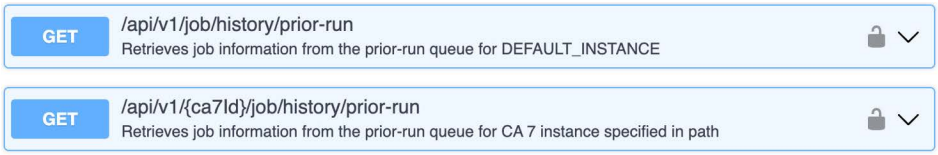
Figure 8: Job Instances



3.7 Job History

The Job History APIs enable you to retrieve job information from the prior-run queue. These APIs provide a comprehensive view of job execution history, aiding in monitoring and troubleshooting past job runs.

Figure 9: Job History

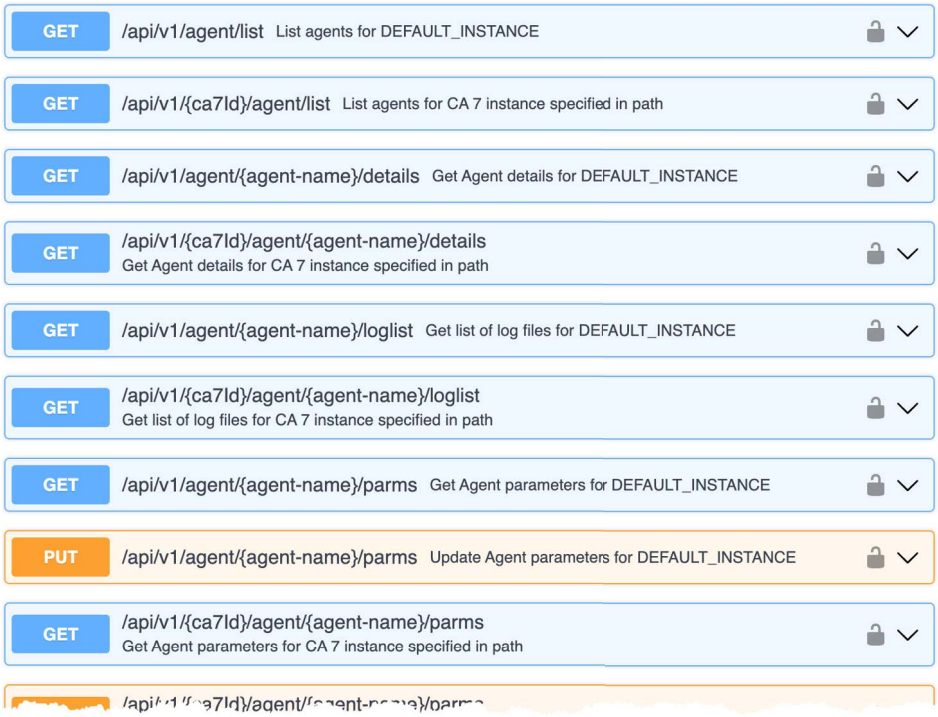


3.8 Agent Management

The Agent Management category of APIs enables applications in the following ways:

- Retrieve a list of the Agents defined to an instance of CA 7, including the Agent's version and status.
- Retrieve detailed information about the Agent, the Agent's parameters, and log files from the Agent.
- Take control of Agents to update the parameters and stop-start the Agent.

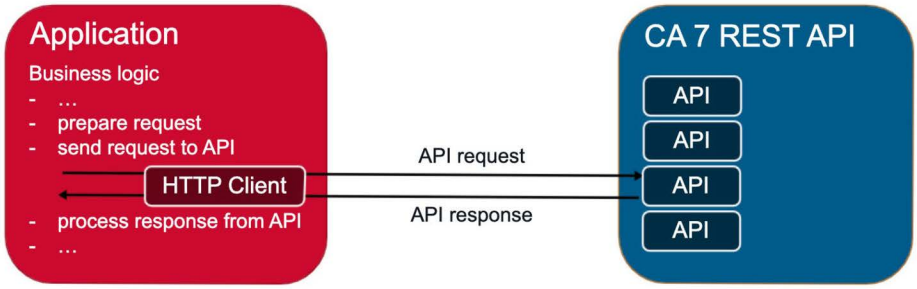
Figure 10: Agent Management



Chapter 4: CA 7 REST API Integration

Many applications today have built-in API integration points, and many developers possess the skills necessary to implement these integrations. Leverage the CA 7 REST APIs to enhance applications with powerful workload automation capabilities, streamlining business processes and improving efficiency.

Figure 11: CA 7 REST API Integration



Integrating the CA 7 REST APIs involves implementing business logic that aligns with specific requirements. This could include the following:

- Automating job scheduling and execution
- Monitoring job statuses and handling exceptions
- Managing datasets and resources dynamically
- Controlling Agents and retrieving detailed logs for troubleshooting

By embedding this business logic within your applications, you can create seamless workflows that reduce manual intervention and increase reliability.

4.1 Using the APIs

Let's look at how an application processes an API request.

Figure 12: API Request

```
curl -X 'POST' \
  'https://host:8443/ca7services/api/v1/commands/demand' \
  -H 'accept: application/json' \
  -H 'Authorization: Basic xxxxxxxxxxxxxxxxxxxxx' \
  -H 'Content-Type: application/json' \
  -d '{
  "job": "PAYROLL",
  "depjob": "PREJB10",
  "schid": "9"
}'
```

Using the CA 7 REST API is straightforward:

1. **Construct the API Request:** The application constructs the API request using an HTTP client. The request is comprised of the following parts:

- **The URI for the service**
- **An HTTP Authorization header**
- **Other HTTP request headers**, as required by the API
- **A request body**, if one is needed

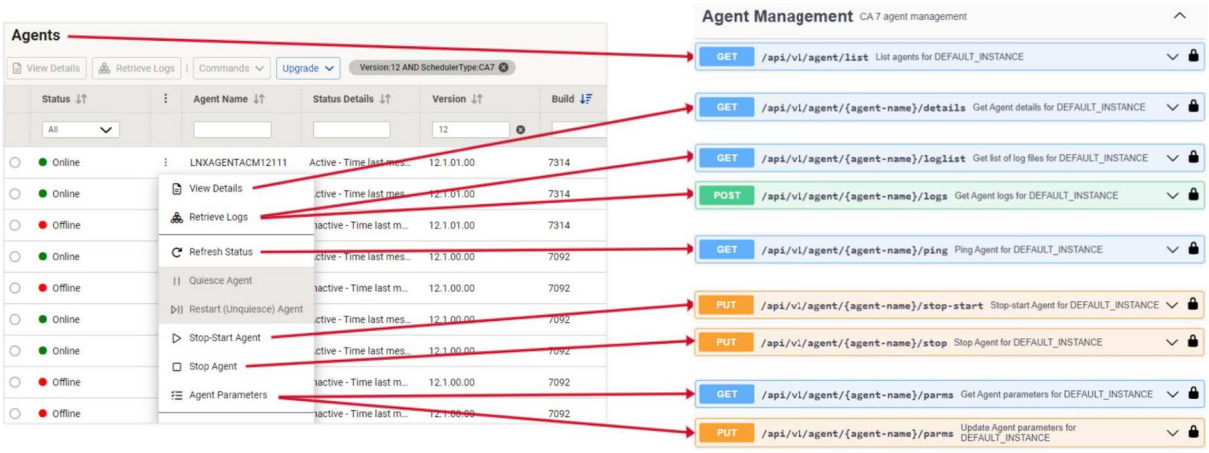
The Swagger documentation helps developers understand how to form the request. The Curl command displayed after using Swagger to execute an API request shows the components needed to construct the API request.

2. **Execute the API Request:** The application executes the API request using the appropriate HTTP method: GET, POST, PUT, DELETE, etc.
3. **Process the Response:** Once the response is received from the API, the application retrieves the response body and takes action based on the HTTP return code:
   - A code 200 indicates success and the application can proceed with the returned data.
   - A code 400 or other error codes indicate an issue; the application should handle it accordingly.

4.2 Rest API Integration

The Broadcom Agent Manager application can be used as an example. This application empowers users to monitor and manage the Agents connected to ESP and CA 7 schedulers.

Figure 13: Workload Automation Agent Manager



There is a one-for-one relationship between the functionality in Agent Manager and the Agent Management APIs made available by the CA 7 REST API. Here's how Agent Manager utilizes these APIs:

- **Retrieve the List of Agents:** When Agent Manager produces the list of Agents, it uses the Agent list API to retrieve the list. This allows users to see all the Agents defined to a CA 7 instance, along with their versions and statuses.
- **Display Detailed Information about an Agent:** When displaying detailed information about an Agent, the Agent Manager uses the Agent details API to retrieve the necessary details. This includes information about the Agent's parameters, log files, and other specifics.
- **Update Agent Parameters:** If users need to update an Agent's parameters, Agent Manager interacts with the relevant API to make those changes seamlessly.
- **Manage Agent Operations:** The Agent Manager uses corresponding API calls to control actions such as stopping and starting an Agent, ensuring that the Agents are managed efficiently and effectively.

By leveraging the CA 7 REST API, Agent Manager can provide comprehensive monitoring and management capabilities that align directly with the API functionalities.

4.3 Securing the APIs

When implementing the CA 7 REST API, it's crucial to prioritize security to protect data and maintain the integrity of the workload automation environment. Here are some best practices:

1. **Configure the REST API to Require HTTPS:** Always ensure that REST API endpoints are accessed over HTTPS to encrypt data in transit. Since HTTP Basic Authentication is used, HTTPS is essential to protect the credentials from being intercepted.
2. **Use Secure Connections to CA 7 (via AT-TLS):** Implement AT-TLS to secure the connections between the application and the CA 7 instance, ensuring that data is encrypted during transmission.
3. **Use Secure Connections to the Datacom Server (via AT-TLS):** Use AT-TLS to secure connections to the Datacom server, safeguarding the data flow between the application and the database.
4. **Use Certificates Signed by a Proper Certificate Authority:** Always use certificates signed by a trusted Certificate Authority (CA) to validate the authenticity of the servers and to prevent man-in-the-middle attacks.
5. **Implement Endpoint Authorization:** Ensure that access to API endpoints is restricted to authorized users only. Implement robust authentication and authorization mechanisms to control who can interact with the APIs.

For detailed instructions and additional security measures, refer to the following documentation topics:

- Secure the CA 7 REST API
- Implementing Endpoint Authorization
- Encrypted Communications for a CA 7 REST API

Ensuring these security measures are in place will help protect data and maintain the reliability of CA 7 REST API integrations.

Chapter 5: CA 7 REST APIs - Technical Nuances

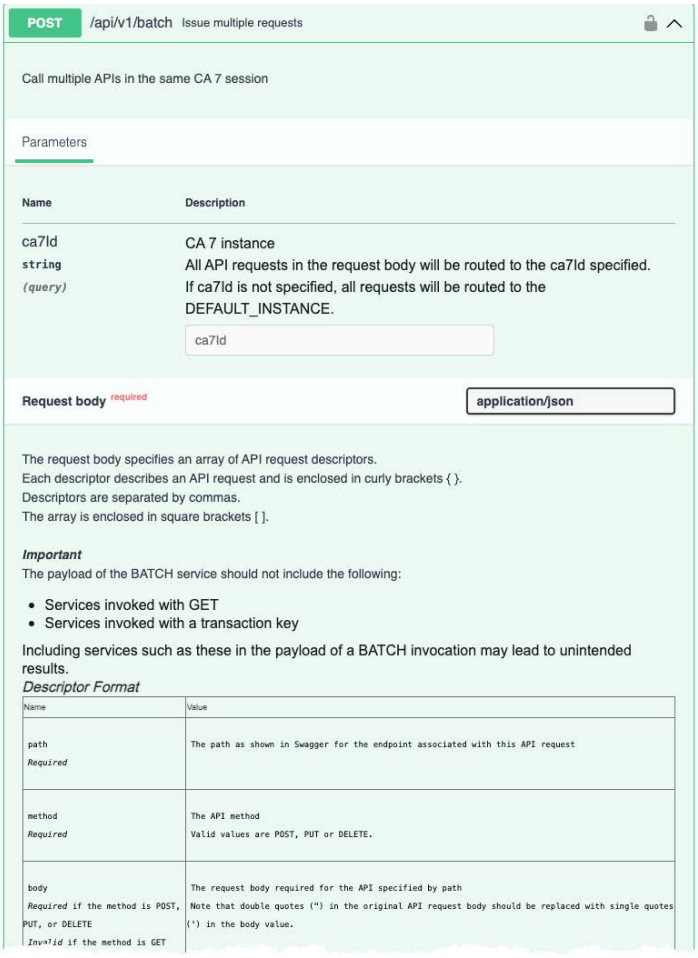
In this chapter, we delve into some technical nuances of using the CA 7 REST APIs to optimize operations and resource efficiency. These considerations will help manage CA 7 REST API integrations more effectively and ensure that applications can gracefully handle various scenarios:

- **CA 7 REST Terminal Resources:** Many APIs use CA 7 REST terminal resources. Ensuring there are enough terminals defined in CA 7 to handle API requests is essential. A lack of available terminals can lead to failures in executing commands.
- **Batch API Usage:** Consider using the Batch API to group multiple operations into a single request. This can help reduce the system's overhead and improve efficiency.
- **Handling Failures:** Failures do happen, so always code to handle failures gracefully. Never assume the API calls will be successful. Always check return codes and implement robust error handling to manage and log errors effectively.
- **Transaction Key API for Critical Commands:** For critical commands, use the Transaction Key API. This API ensures that critical operations are executed reliably and can be tracked accurately.

5.1 Batch API Usage

To efficiently manage and conserve CA 7 terminal resources, consider bundling API requests.

Figure 14: Batch API Requests to Conserve CA 7 Terminal Resources



Implementing this strategy helps maximize CA 7 terminal resources, ensuring workload automation processes run smoothly and efficiently:

- **Bundle a Set of API Requests:** By bundling a set of API requests, you can perform multiple operations within a single transaction, reducing the system's load
- **Use a Single CA 7 REST Terminal:** When you bundle API requests, they use a single CA 7 REST terminal. This approach minimizes the number of terminals needed for operations, conserving resources.
- **API Requests Made One by One:** Although the requests are bundled, they are executed one by one within the batch. This ensures that each operation is processed sequentially, maintaining the integrity and order of the tasks.
- **Useful for Bulk Operations:** Bundling is particularly useful for bulk operations, where multiple tasks need to be performed in a single execution cycle. This approach can significantly reduce overhead and improve efficiency.

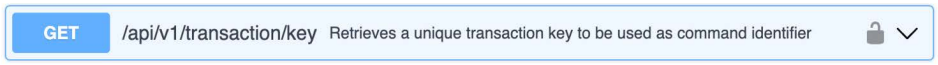
IMPORTANT: Do NOT include the following in the payload of the BATCH service. Including services such as these in the payload of a BATCH invocation may lead to unintended results:

- Services invoked with GET
- Services invoked with a transaction key

5.2 Utilizing the Transaction Key API

Using the Transaction Key API is another crucial aspect of managing CA 7 terminal resources and ensuring reliable operation. This API is beneficial for critical commands that must be issued only once.

Figure 15: Transaction Key API



- **Retrieve a Unique Transaction Key:** The Transaction Key API allows you to retrieve a unique transaction key. This key serves as a unique identifier for the command, ensuring that it is recognized and processed correctly by CA 7.
- **Valid for 60 Minutes:** The transaction key is valid for 60 minutes, providing a sufficient window to complete operations without frequently requesting a new key.
- **Allows Command Idempotence:** Using the transaction key ensures command idempotence. This means that even if the same command is issued multiple times, it will be processed only once, preventing duplicate operations and maintaining data integrity.
- **Useful for One-Time Commands:** This API is handy when a command must be issued only once. For instance, a **DEMAND** command can leverage this key to ensure they are not executed more than once.

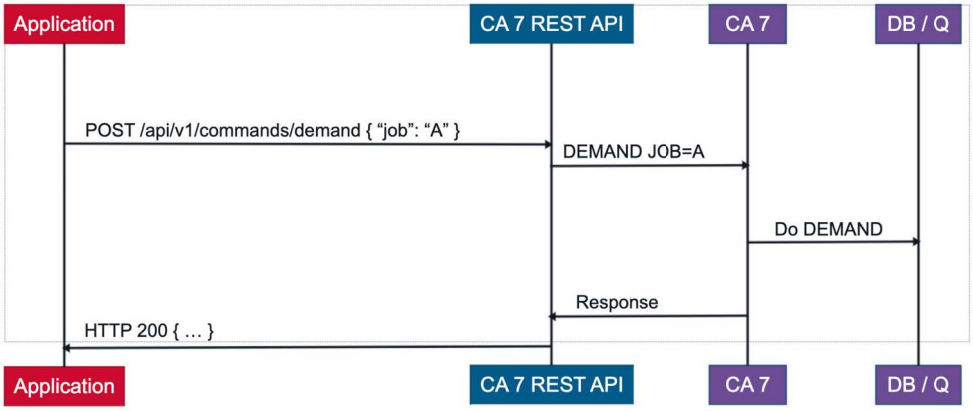
By incorporating the Transaction Key API into the workflow, you can enhance the reliability and control of CA 7 REST API interactions, ensuring that critical commands are executed accurately and efficiently.

The following sections detail scenarios with and without the Transaction API end point.

5.2.1 Scenarios: DEMAND without a Transaction Key

Figure 16 illustrates a successful **DEMAND** command without using a transaction key.

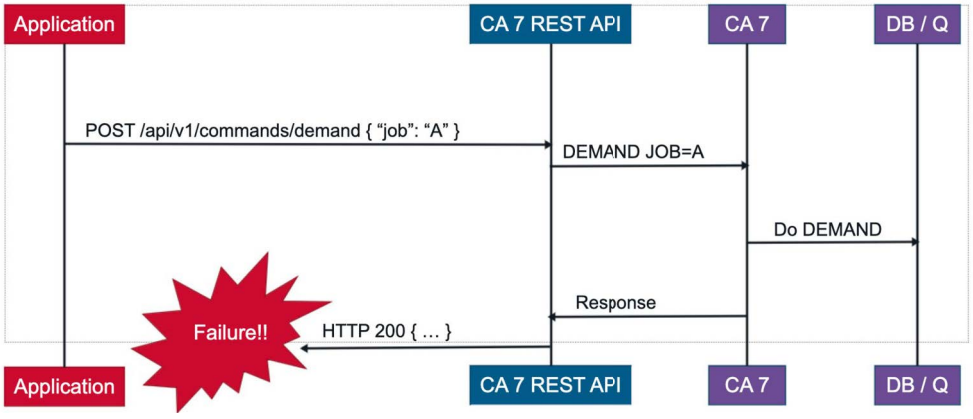
Figure 16: Successful DEMAND Command without Using a Transaction Key



The diagram illustrates issuing a **DEMAND** command without a transaction key. The application sends a POST request to the CA 7 REST API to demand job A. The CA 7 REST API forwards this command to CA 7, which then performs the demand operation. The CA 7 REST API receives the response and relays it back to the application, resulting in an HTTP 200 success code.

Figure 17 demonstrates a scenario where a **DEMAND** command without a transaction key fails.

Figure 17: Failed DEMAND Command without Using a Transaction Key



The application sends a POST request to the CA 7 REST API to demand job A. The CA 7 REST API forwards this command to CA 7, which attempts to perform the demand operation. Despite the API responding with an HTTP 200 success code, the actual demand operation fails.

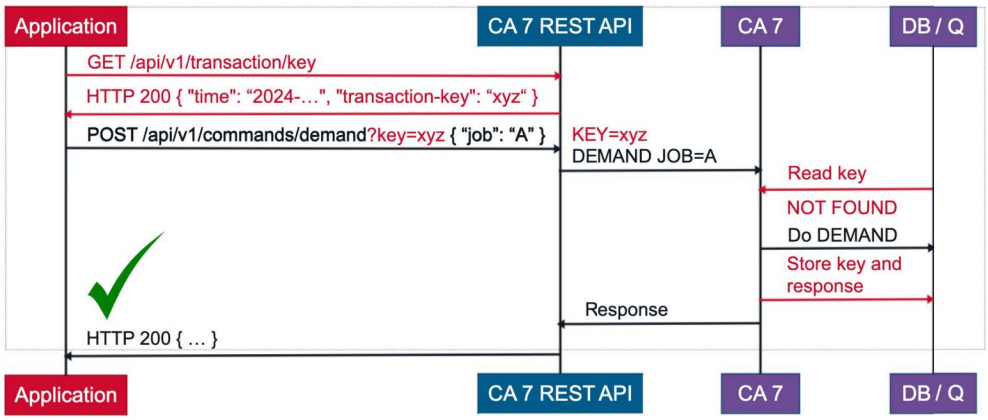
In such cases, simply reissuing the API request might not be feasible due to the risk of duplicating commands or other underlying issues that caused the initial failure. Using transaction keys for critical commands ensures that the operation is processed correctly and prevents duplicate actions, maintaining the integrity of your workload management.

Let's talk about the flow when we use a transaction key.

5.2.2 Scenarios: DEMAND with a Transaction Key

Figure 18 illustrates a successful **DEMAND** command using a transaction key.

Figure 18: Successful DEMAND Command Using a Transaction Key



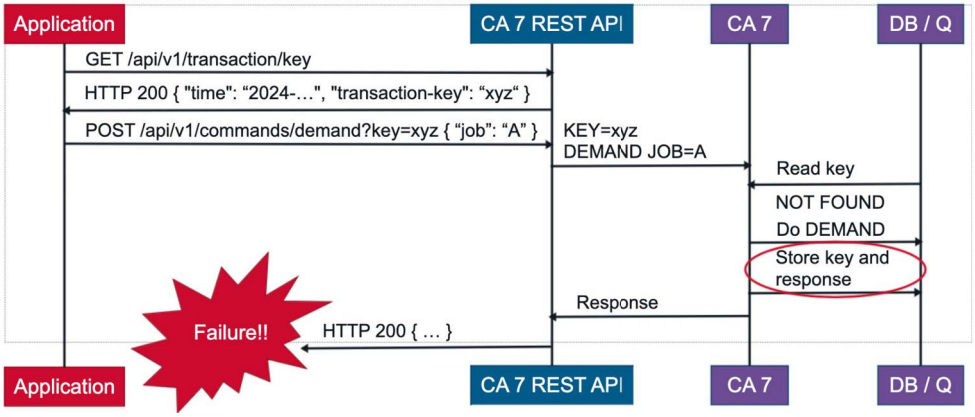
Here's the step-by-step process:

1. **Retrieve Transaction Key:**
   a. The application sends a GET request to `/api/v1/transaction/key` to retrieve a unique transaction key.
   b. The CA 7 REST API responds with an HTTP 200 status, providing the transaction key `"transaction-key": "xyz"`.
2. **Send Demand Command with Transaction Key:**
   a. The application then sends a POST request to `/api/v1/commands/demand?key=xyz` with the job details `"job": "A"`.
   b. The CA 7 REST API forwards this command to CA 7, including the transaction key.
3. **Process Demand Command:**
   a. CA 7 reads the transaction key and checks if it has been used before.
   b. Since the key is not found in previous records, CA 7 performs the **DEMAND** operation.
   c. The transaction key and response are stored to ensure idempotence. The storing of the transaction key and original response is what makes this work.
4. **Receive Success Response:**
   a. The CA 7 REST API receives the response from CA 7 and relays it back to the application.
   b. The application receives an HTTP 200 success code, indicating that the demanding job A was successfully executed.

Using a transaction key ensures that critical commands are executed only once, preventing duplicates and maintaining the operation's integrity.

Figure 19 illustrates a scenario where a **DEMAND** command using a transaction key fails.

Figure 19: Failed DEMAND Command Using a Transaction Key



Here's the step-by-step process:

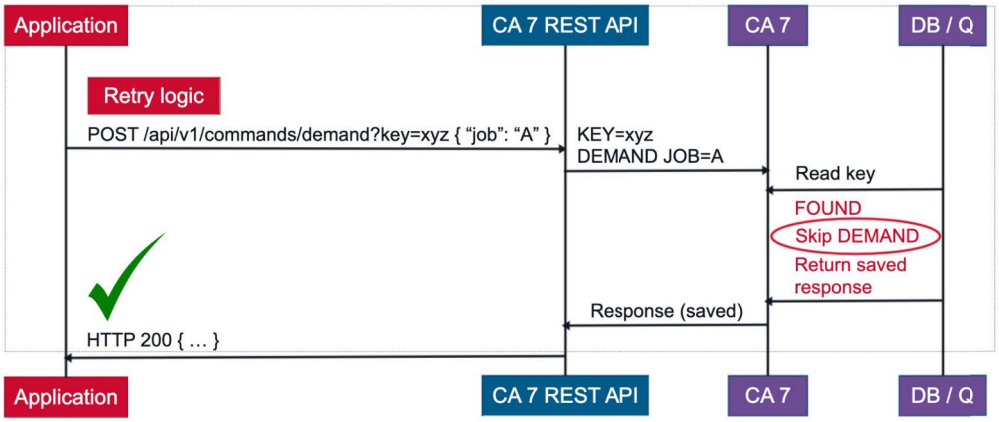
1. **Retrieve Transaction Key:**
   a. The application sends a GET request to `/api/v1/transaction/key` to retrieve a unique transaction key.
   b. The CA 7 REST API responds with an HTTP 200 status, providing the transaction key `"transaction-key": "xyz"`.
2. **Send Demand Command with Transaction Key:**
   a. The application then sends a POST request to `/api/v1/commands/demand?key=xyz` with the job details `"job": "A"`.
   b. The CA 7 REST API forwards this command to CA 7, including the transaction key.
3. **Process Demand Command:**
   a. CA 7 reads the transaction key and checks if it has been used before.
   b. Since the key is not found in previous records, CA 7 proceeds to perform the demand operation.
   c. The transaction key and response are stored to ensure idempotence. The storing of the transaction key and original response is what makes this work.
4. **Failure in Application:**
   a. The application receives an HTTP 200 success code from the CA 7 REST API, indicating that the command was processed.
   b. However, an issue occurs within the application, resulting in a failure. Despite the successful API response, the application flags a failure condition.

This scenario highlights the importance of handling and verifying responses correctly within your application. Even if the CA 7 REST API successfully processes the command, application-level errors can still occur. The use of transaction keys ensures that critical commands are not duplicated and maintains the integrity of operations.

5.2.3 Scenario: DEMAND with Transaction Key - Retry after Failure

Figure 20 illustrates a scenario where a **DEMAND** command using a transaction key is retried after an initial failure.

Figure 20: Retry after Failure DEMAND Using a Transaction Key



Here's the step-by-step process:

1. **Initial Failure Handling:**
   a. Initially, the application attempted to send a **DEMAND** command with a transaction key `"transaction-key": "xyz"`.
   b. The CA 7 REST API processed this request, and CA 7 stored the transaction key and the response.
2. **Retry Logic:**
   a. After the initial failure in the application, the retry logic kicks in. The application sends the POST request again to `/api/v1/commands/demand?key=xyz` with the job details `"job": "A"`.
3. **Check Transaction Key:**
   a. The CA 7 REST API forwards the command to CA 7, including the transaction key.
   b. CA 7 reads the transaction key and finds it in its records, indicating that this command has already been processed.
4. **Skip Redundant DEMAND:**
   a. Since the transaction key is found, CA 7 skips re-executing the **DEMAND** command to avoid duplicate operations.
   b. Instead, CA 7 returns the previously saved response.
5. **Receive Success Response:**
   a. The CA 7 REST API receives the saved response from CA 7 and relays it back to the application.
   b. The application receives an HTTP 200 success code, indicating that the demanding job A was successfully executed, even though it was not re-executed.

This retry mechanism ensures that the demand command is not duplicated, maintaining the integrity of operations. Using transaction keys allows the application to handle retries gracefully, providing reliability and preventing unnecessary actions.

Chapter 6: Additional Information

Broadcom is here to help, so please don't hesitate to reach out to the team if you have any questions or need assistance:

- Refer to mainframe.broadcom.com/workload-automation for additional information.
- Message an expert at mainframe.broadcom.com/contact.


BROADCOM®
MAINFRAME SOFTWARE