Chapter 9: Channels

Welcome to this chapter on channel implementation in BIZUIT. Throughout this series, we will dive into the tools that BIZUIT gives us to integrate systems, automate tasks, and ensure efficient connectivity between applications. Not only will this knowledge allow us to optimize processes, but it will also help us reduce the repetitive workload and improve the overall efficiency of our operations.

In the business world, the need for systems to communicate with each other is becoming increasingly crucial. BIZUIT allows us to connect different applications and systems through what we call channels. These function as endpoints that manage data transfer and command execution without manual intervention.

Think, for example, of a customer service system: with BIZUIT channels, we can receive, process and respond to requests automatically, improving the customer experience and optimizing our team's time.

Chapter Structure

This chapter is divided into four main units:

1. Introduction to channels in BIZUIT: In this first unit, we will explore the basics and importance of channels in the context of automation.
2. Channel types and use cases: In the second unit, we will delve into the different types of channels available in BIZUIT and analyze concrete examples of their application.
3. Practical configuration: In the third unit, we will focus on the configuration of each type of channel with practical examples that will help us implement what we have learned.
4. Best practices and optimization: Finally, in the fourth unit, we will review best practices to ensure that our channels are secure, reliable, and optimized.

Throughout this journey, we will not only acquire theoretical knowledge, but also develop the ability to apply it in our own organizations.

Ideal Audience

This class is designed for software developers looking to implement complex integrations into BIZUIT, automate processes, and connect external systems, as well as for technology architects interested in maximizing BIZUIT's interoperability and connectivity to other enterprise systems using APIs, web services, and other protocols.

Objectives

1. Understand the concept and purpose of channels in BIZUIT: Know how channels work as integration and automation tools within the platform.
2. Configure and use the different types of channels: Learn how to configure and manage each type of channel available in BIZUIT, such as Process Schedulers, File Monitoring, TCP Servers, and Web Services, among others.
3. Implement integrations with external systems: Know how to use channels to connect BIZUIT with other external systems and services, facilitating the interoperability of the platform.
4. Automate workflows and processes: Use channels to execute BIZUIT processes in an automated way, optimizing times and improving operational efficiency.
5. Apply good security and monitoring practices: Configure channels securely and learn how to monitor their activity to maintain the integrity and performance of the system.

Unit 1: Introduction to Channels in BIZUIT

Channels in BIZUIT are automated connections between different systems or applications that allow us to exchange information in real time or at defined intervals. Let's imagine that we have an inventory system on one platform and a sales system on another.
Without channels, communication between them would be complex and manual; With channels, we achieve an instant update of inventories every time a sale is made, avoiding errors and improving the accuracy of our data.

Benefits of using channels in BIZUIT

What do we gain by implementing these channels? The benefits can be summarised in three fundamental pillars:

- **Automation**: We reduce manual tasks and allow our team to focus on higher-value activities.
- **Connectivity**: By linking systems that would otherwise not be able to communicate, we create a continuous flow of information that improves decision-making.
- **Scalability**: The channels in BIZUIT can adapt to changes in workload and the expansion of our business. This means that no matter how much we grow or evolve, the channels will adjust to our needs.

Examples of scenarios where channels add value

Let's look at some specific examples in different sectors:

- **Healthcare**: In a hospital, requests for medical studies need to be handled quickly. Through a file monitoring channel, BIZUIT can automatically detect and upload these orders to the management system, ensuring that the medical team has immediate access to the information.
- **Transportation and logistics**: In a transportation company, delivery management can be optimized with a channel that monitors email accounts. In this way, service requests are automatically processed and assigned to the corresponding units without manual intervention.

These examples reflect how channels not only optimize our processes, but also allow us to scale efficiently, ensuring that our operation remains agile and adaptive.

Unit 2: Introduction to Channel Types

In this unit, we will explore in detail the different types of channels that BIZUIT offers us. Each of them has a specific purpose and allows us to interact with different systems and data sources. We will analyze how these channels facilitate automation and improve the integration of processes within our organizations.
As we move forward, we'll see that they're not only versatile, but also critical to building a connected and efficient business infrastructure.

API Rest

The Rest API channel allows us to expose BIZUIT functionalities as services accessible through HTTP requests. This is key to connecting BIZUIT with external applications that need to query or send data in real time. Let's think of an external inventory system that requires knowing the stock of products in BIZUIT. By setting up a Rest API channel, we can expose this information and ensure that every time a request is made, the inventory system gets the updated data.

This type of channel allows us to integrate BIZUIT in an agile way with modern applications, especially those that depend on microservices or *serverless* architectures, facilitating interoperability.

Web Services

Web services allow us to run processes in BIZUIT when a web service is invoked, using the SOAP protocol. This option is ideal for integrating systems remotely and ensuring seamless communication between platforms.

For example, if an external application needs to check the status of an order in BIZUIT, we can expose this functionality through a web service. In this way, each query is automatically processed in real time.
With web services, we achieve flexible integration with third-party applications, which is essential in business environments where multiple systems must interact.

Process Programmers

Process schedulers are channels designed to execute tasks automatically and at regular intervals. They are especially useful for repetitive activities that need to be done at specific times or predefined cycles.

Let's imagine that we must generate a sales report daily. With a process scheduler, we can set it up so that this report is automatically generated at midnight and sent to the management team without the need for manual intervention.

This ensures that critical tasks are performed on time and without relying on staff availability, reducing the risk of delays and errors.

File Monitoring

The file monitoring channel allows us to monitor specific folders in the operating system or on FTP servers and run processes automatically when a new file or a change to existing ones is detected.

Let's take a logistics company as an example, where suppliers upload inventory documents to an FTP folder. By implementing a file monitoring channel, BIZUIT can detect each new document, process it, and automatically update the inventory database.

This eliminates the need for manual intervention and ensures that information is kept up to date in real-time, which is crucial in operations where speed and accuracy are critical.

IMAP Account Monitoring

This channel allows BIZUIT to monitor IMAP email accounts and run processes automatically whenever a new message is received. It is ideal for companies that handle requests or complaints through email.

For example, in a customer service center, every time an email is received in the support account, BIZUIT can analyze its contents, assign it to an agent, and record the case in the management system. Automating email processing not only saves the support team time, but also improves response speed and customer satisfaction.

Kafka Message Monitoring

Kafka is a messaging platform for real-time data transmission. With BIZUIT, we can set up a Kafka channel to receive messages and run processes in response to them.

For example, in a company that handles sales in multiple stores, every time a sales message is received in the Kafka channel, BIZUIT can update the database and send alerts in case of low stock.

This type of integration is ideal for environments where the flow of data is constant and of high volume, allowing us to manage information efficiently and with high scalability.

Message Queue Managers

Message queue managers allow us to manage the queuing of messages before sending them to BIZUIT, ensuring that each message is processed, even if there are temporary system failures.

Let's imagine that in an online store we have to manage customer orders. With a queue manager, each order is stored before being processed in BIZUIT, preventing it from being lost in the event of a system crash.

This channel ensures secure and controlled management of messages, streamlining workflow and ensuring that no important information is misplaced.

TCP Servers

The TCP Servers channel in BIZUIT allows us to expose a TCP server so that other applications can send messages directly. This is especially useful in production environments where real-time communication is critical.

Consider a hospital where BIZUIT acts as a TCP server to receive data from medical devices, such as vital signs monitors in the ICU.
Every time one of these devices detects a value outside the normal range, it sends a message to BIZUIT, which immediately triggers an alert process.
This can include sending notifications to medical staff, recording the event in the medical record system, and triggering visual alarms in the control room.

Thanks to this channel, we ensure continuous monitoring and rapid medical intervention, improving patient safety and optimizing emergency response.

Unit 3: Channel Configuration in BIZUIT

In this unit, we will explore the configuration of the different types of channels in BIZUIT, fundamental tools for the integration and automation of processes within the system. Channels allow connections to be established with external applications, expose services such as REST APIs, schedule recurring tasks, monitor files or emails, and even interact with message queues and TCP servers.

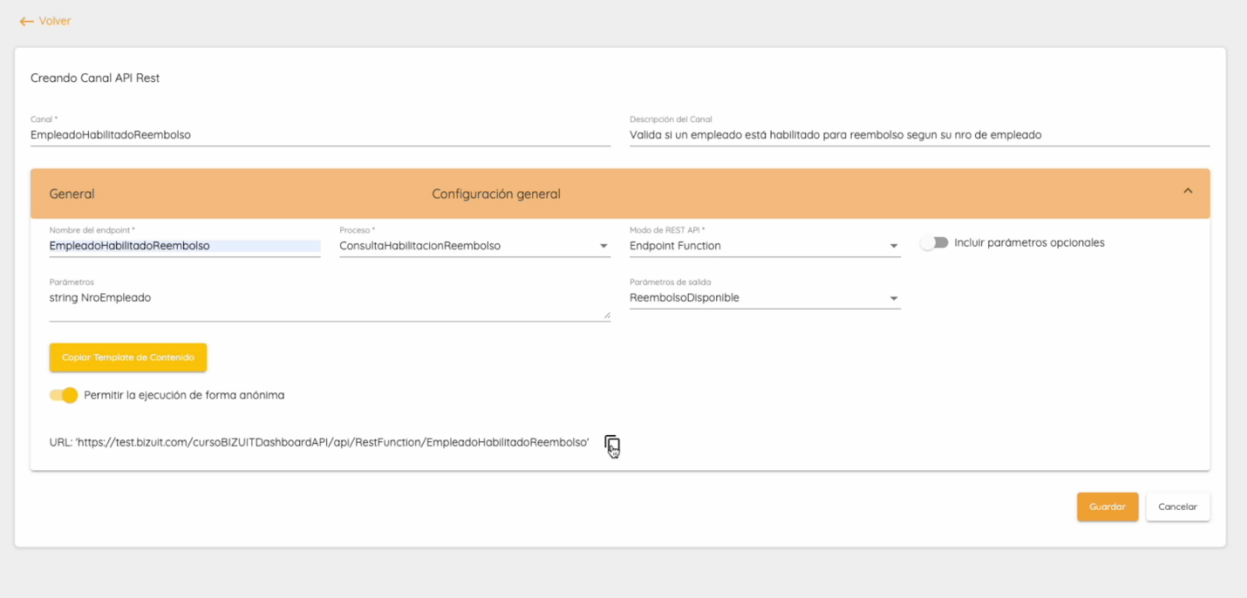
We'll detail step-by-step how to set up a REST API channel, one of the most flexible and efficient ways to expose BIZUIT functionalities to other systems. We'll look at how to define the endpoint, set input and output parameters, configure security options, and perform connection tests using tools like Postman. In addition, we will learn about other key channels, such as Web Service channels, ideal for integrations with SOAP-based systems, and process schedulers, which allow the automatic execution of tasks at predefined intervals.

Finally, we will address the configuration of more advanced channels, such as file monitoring and IMAP emails, which facilitate the automation of workflows based on the arrival of new data, and TCP and Kafka channels, which allow real-time message processing from external devices and applications.

With this unit, you will not only understand how to set up each channel type, but you will also learn how to optimize its use to ensure robust and secure integrations in any business environment.

Configuring a REST API Channel

Next, we'll set up a REST API channel together in BIZUIT, step by step. This channel will allow us to expose BIZUIT functionalities as REST services, allowing other applications to access the information in real time. In this case, we'll set up a channel called *Employee Enabled for Reimbursement*, which will validate whether an employee is eligible for a refund based on their employee number.



Defining the channel and its purpose

We start by giving the channel a name. In this case, we've decided to call it *Employee EnabledRefund*. This name will be key to identifying the channel in our system and remembering its purpose clearly.

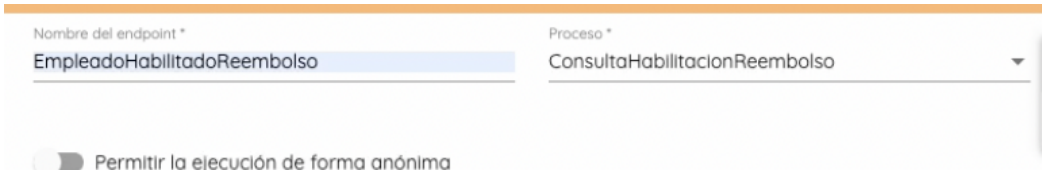
It is advisable to include a detailed description so that other users or administrators understand its use.

In our case, we can describe it as: *Validates if an employee is eligible for reimbursement based on their employee number*. Adding a detailed description will help us maintain clarity and organization in large-scale projects.

Endpoint Configuration

Next, we define the name of the endpoint, which will be the unique identifier of our API within BIZUIT. For this example, we use the same name: *Employee Eligible for Reimbursement*. It is important that this name is descriptive enough to clearly reflect the function it fulfills.

Then, we select the process that will run when this endpoint is invoked. In this case, we chose *Reimbursement Enablement Consultation*, which is the specific process in BIZUIT that will verify if the employee is eligible to receive a reimbursement.



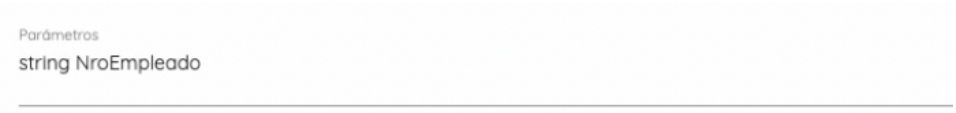
REST Mode API and Input Parameters

In this step, we need to choose between two REST API mode options:

- **Endpoint Function:** Exposes a specific function as a standalone endpoint.
- **BIZUIT SDK API:** Allows deeper integration with the BIZUIT SDK, making it easier to start, continue, or delete processes.

For our case, we will use the Endpoint Function option.

Now, we define the input parameters that our endpoint requires. In this case, the selected process contains only one input parameter: *Employee Number*. If there were more optional parameters, we could include them according to our need.



Defining Output Parameters

The output parameters are the data that the API will return after the process is executed. In this case, we have only one output parameter: *ReimbursementAvailable*, which indicates whether the employee is eligible for reimbursement. If the process had multiple output parameters, we could select them according to our needs.

It is essential that the output parameters are clear and relevant to those who consume the service, thus ensuring effective and understandable integration.

Channel activation and security options

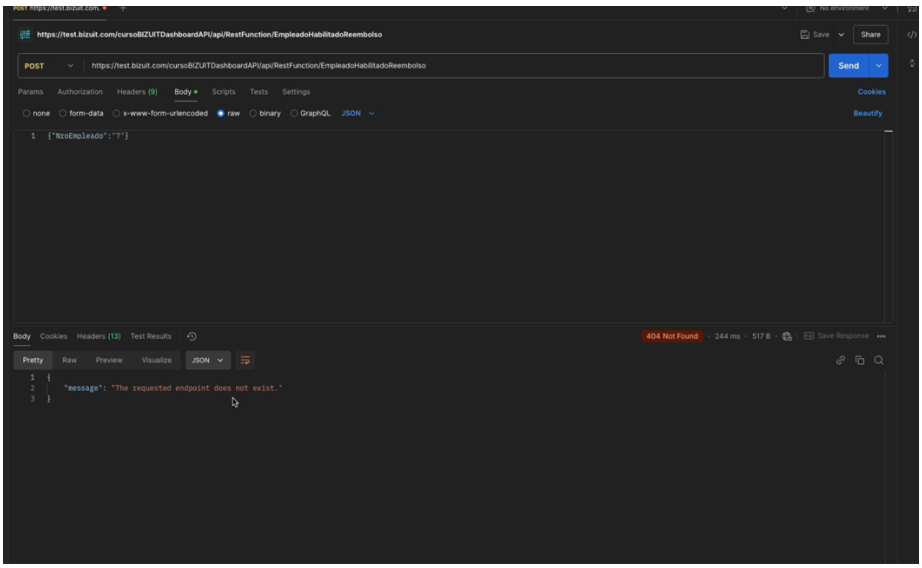
Here we find the option to *Allow Anonymous Execution*. If we enable this option, any user will be able to access the endpoint without authentication. This can be useful in certain cases, but

we must carefully assess the security risks. In this example, we'll turn on the option to make endpoint testing easier.

Once the channel name and endpoint are configured, an API access URL is automatically generated. This address will be the one that other systems will use to connect and query the API.

Endpoint testing with Postman

We copy the generated URL and test it in Postman. By default, *endpoint functions* work with the POST method. In the *Body* section, we select raw.json and copy the content structure.
If we enter an employee number, for example *7*, and the endpoint is not yet saved, it will return an error indicating that the endpoint does not exist. This happens because we haven't activated it yet.



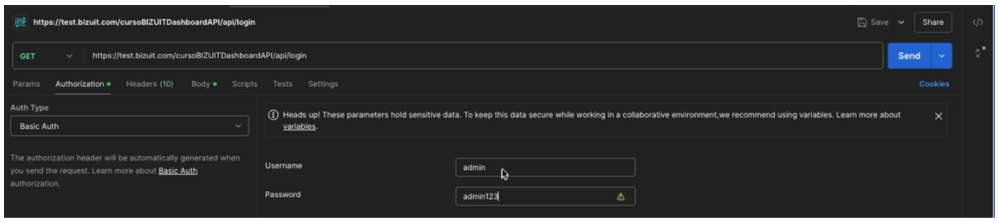
After saving and enabling the endpoint, if the request does not comply with the expected format, the API will return an error indicating that the Employee *Number parameter* is required.

If we do not enable anonymous execution, the system will return a 401 (Unauthorized) code, which means that we need to authenticate to BIZUIT before executing the process.

Authenticating and Obtaining a Token

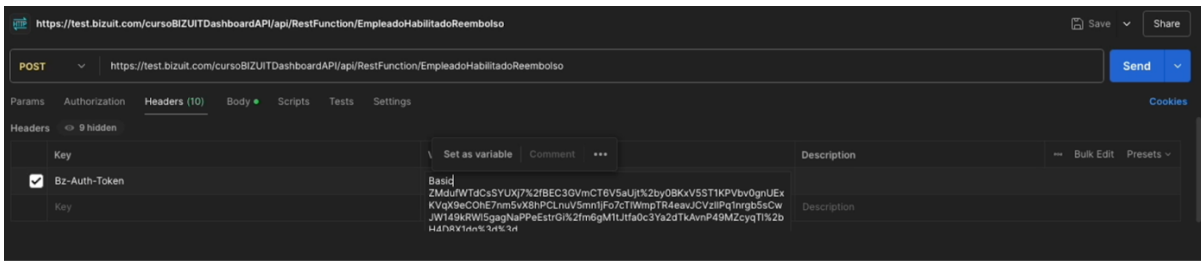
To access a REST API service that requires authentication, we need to get a token.

To do this, we perform a GET to the API URL of our BIZUIT installation followed by "/api/login" as indicated in the image



In the **Authorization** tab, select **Basic Auth** and type in our username and password.
The API will return a token, which we will use to authenticate our requests:

Then, in Postman, we add this token in the *headers* of our request to the *EmployeeEnabledReimbursement* endpoint, which will allow us to execute the query successfully.



Advanced options with BIZUIT SDK API

If we choose to use the REST API channel as the BIZUIT SDK API, we can define different methods to obtain and manipulate data from the process:

- **GET**: Obtains the data of an instance by sending the *Instance ID* in the *Query String* or in the *Body*.
- **POST:** Starts a process, sending the data in the *Query String* or in the *Body*.
- **PUT**: Allows you to continue a process, indicating the *Instance ID* and the employee number to be modified.
- **DELETE:** Deletes an instance, allowing you to choose between sending it in the *Query String* or in the *Body*.

Channel Completion and Testing

Once all the parameters and options of the REST API channel have been configured, we press Save to finish the configuration. We must not forget to activate the channel so that the endpoint can be accessed from the URL provided.
It is advisable to perform a test of the endpoint after saving it, to verify that it works correctly.

With these steps, we have completed the configuration of a REST API channel in BIZUIT. This process allows us to expose information in a controlled and secure way, facilitating integration with other systems.

In this case, our endpoint validates whether an employee is eligible to receive a refund. It's important to test and adjust settings based on the needs of each use case.

Setting up a Web Service Channel

Now, we will learn how to set up a Web Service channel in BIZUIT. This type of channel allows us to expose a process as a web service accessible using standard protocols such as SOAP. This is especially useful when we need other systems, especially older ones, to be able to invoke methods in BIZUIT and receive data or perform actions in a controlled manner.

Below, we will see step by step how to carry out this configuration.

Step 1: General Channel Settings

Channel Name

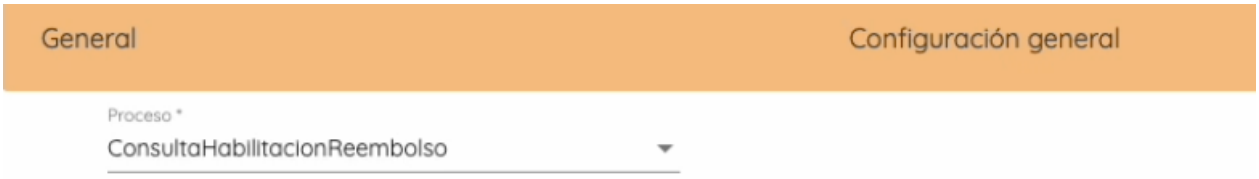
We start by naming our Web Service channel. This name will allow us to identify the channel within BIZUIT and differentiate it from other services. It is important that the name is descriptive and clearly reflects the objective of the service we are explaining.

Channel Description

Below, we added a detailed description to indicate the purpose of this channel. A good description makes it easier for other users to quickly understand what it's used for.

Process Selection

Then, we select the process in BIZUIT that we want to expose through this channel. In our case, we will choose the ConsultationEnableRefund process.



Web Service and Publication Type

In this section, we must select the existing Web Service to which we will add the method that will invoke the process, or we can create a new WebService. In our case, we will create a new one.

To do this, we click on "+", assign a name to the service and select the type of publication, choosing between WCF or ASMX technology. To simplify the setup, we'll use ASMX.

Step 2: Method Configuration

Method Name

Here we specify the name of the method that we will expose through the web service. This name should be clear and descriptive, as it will be the one that other systems will use to invoke the function in BIZUIT.

For example, an appropriate name might be QueryEnabledEmployee, as it accurately describes the functionality of the method.

Parameter Prefix

We can define whether or not the method parameters obtained from the mandatory parameters of the process will have a prefix, such as "p".

Optional Parameters

If our process had optional parameters, we could add them to the endpoint. However, in this case, our process does not possess them.

Default Authentication

If we activate this option, it will not be necessary to enter the username and password in the method parameters, since the service will use the credentials configured in the BIZUIT settings.

Step 3: Configuring Output Parameters

Unlike the REST API channel, where we can select the output parameters, in the Web Service channel, BIZUIT will automatically select the first output parameter found in the process.

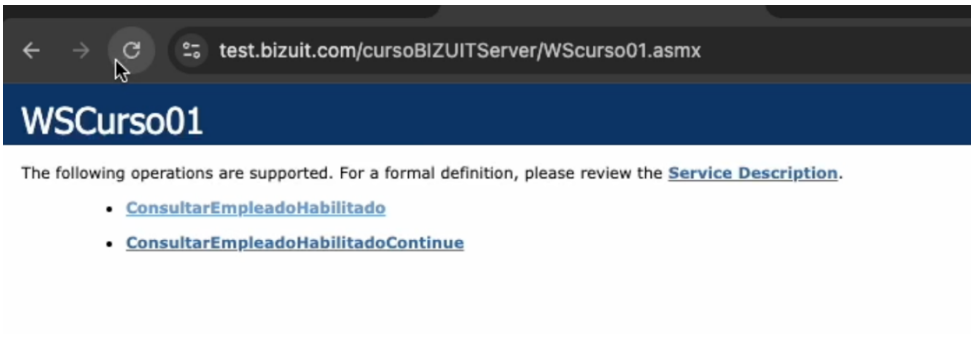
The system will also tell us the type of return of the selected output parameter, ensuring that integration with other systems is clear and consistent.

Step 4: Save Settings and Test

Once we have configured all aspects of the channel and method, we press "Save" to activate the web service.
It is critical to perform service invocation tests from an external application to verify that all input and output parameters are correctly configured and that the service responds as expected.

To do this, we will follow these steps:

1. We navigate to the URL of our BIZUIT server and add the name of the configured WebService.
    - In our case: https://[server]/WSCurso01.asmx.
2. We check that the method we have created is available.

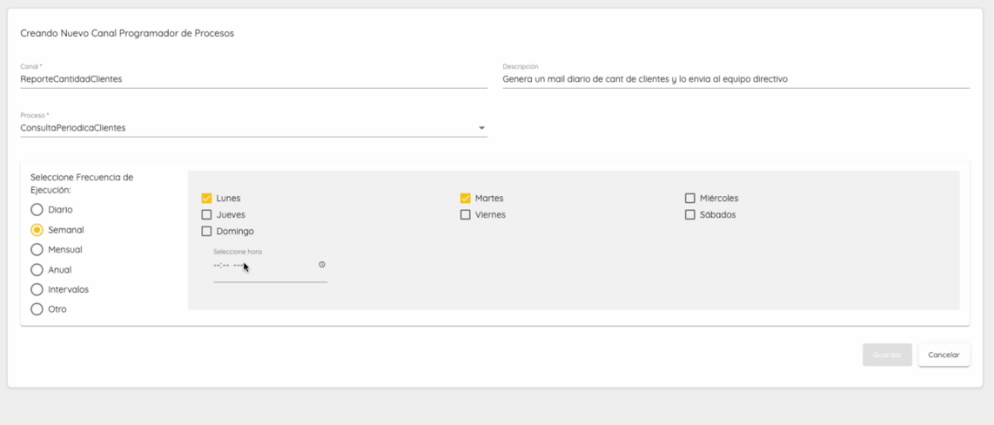


With these steps, we have set up a Web Service channel in BIZUIT. Thanks to this channel, other systems will be able to interact with processes in BIZUIT in a safe and controlled manner.
It is important to document every detail of the configured method, including its parameters and the type of return, to facilitate its use and maintenance in the future.
With this foundation, we can continue to make progress in configuring and integrating more services into our infrastructure.

Process Programmers Channel Configuration

In this part, we will learn how to set up a Process Scheduler channel in BIZUIT. This type of channel allows us to automate tasks that need to be executed at specific times or intervals, without manual intervention. It is ideal for managing repetitive tasks such as generating reports, sending automated emails, updating data or periodic backups.

Throughout this setup, we'll explore each step in detail.



Step 1: Basic Channel Setup

Channel Name

The first thing we do is give the channel a name. This name should be clear and descriptive so that any user can quickly identify the programmer's role.

For example, if the channel is used to generate daily emails with the number of customers, an appropriate name could be "ReportQuantityCustomers".

Channel Description

Then, we add a description detailing the purpose of the channel.

This field is key to providing context and organization, especially when working with multiple channels on a single project.

Process Selection

Then, we select the process in BIZUIT that we want to automate. It is important to remember that only processes that do not require mandatory input parameters will be available, as this channel cannot send values, it only runs the process at the defined intervals.

In our case, we will use the ClientPeriodic Consultation process, which does not require parameters and is responsible for:

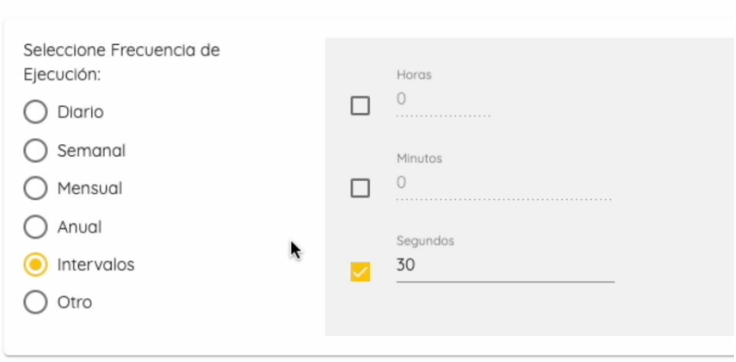
1. Check the number of customers in the system.
2. Send an email with this information to the management team.

Step 2: Selecting the Execution Frequency

One of the most important decisions is to define how often the process will be executed. BIZUIT offers us several options:

- **Daily**: Runs the process every day at a certain time.
- **Weekly**: Allows you to choose the days of the week and the time of execution.
- **Monthly**: Runs on a specific day of each month or in a certain cycle (e.g., the 10th day of every 2 months).
- **Annual**: It is scheduled to run once a year on a specific date and time.
- **Intervals**: Run the process at regular intervals of time, for example, every 30 seconds.
- **Custom Option**: Allows you to define a single run on a specific date and time.

In our case, we'll set the channel to run every 30 seconds.



Step 3: Review and Save Settings

Before we finalize, we review all of the settings to make sure that:

- The process selected is the correct one.
- The frequency and time of execution match our needs.

- The description is clear to other users or administrators.

This step is key to avoiding errors and ensuring that the channel fulfills its function correctly.

Step 4: Save and Activate the Channel

Finally, we press "Save" to activate the channel. From this point on, the scheduler will automatically run the process according to the frequency and schedule you set.
It is advisable to monitor the channel after its first execution to verify that it works correctly and generates the expected results.

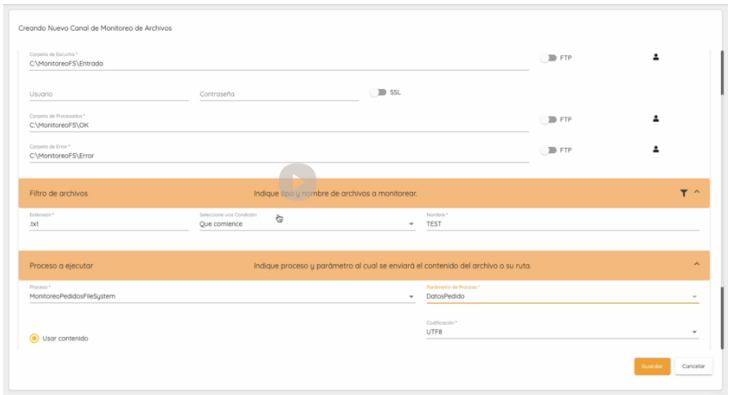
With these steps, we have set up a Process Scheduler channel in BIZUIT. This type of channel is essential for the automation of periodic tasks, allowing us to keep critical processes running without manual intervention.

Whether it's generating reports, making updates, or running backups, Process Schedulers at BIZUIT help us ensure that these tasks are executed in a timely and efficient manner.

File Monitoring Channel Configuration

Next, we'll learn how to set up a File Monitoring channel in BIZUIT.
This channel allows us to automate processes based on the detection of new files in specific folders, whether they are local, shared, or located on FTP servers.

Thanks to this functionality, BIZUIT automatically detects new files and executes a process in response, optimizing data integration and reducing manual intervention. Below, we'll look at each step in detail.



Step 1: Basic Channel Setup

Channel Name and Description

The first thing we do is assign a name to the channel, which must be descriptive enough so that any user can quickly identify its function.

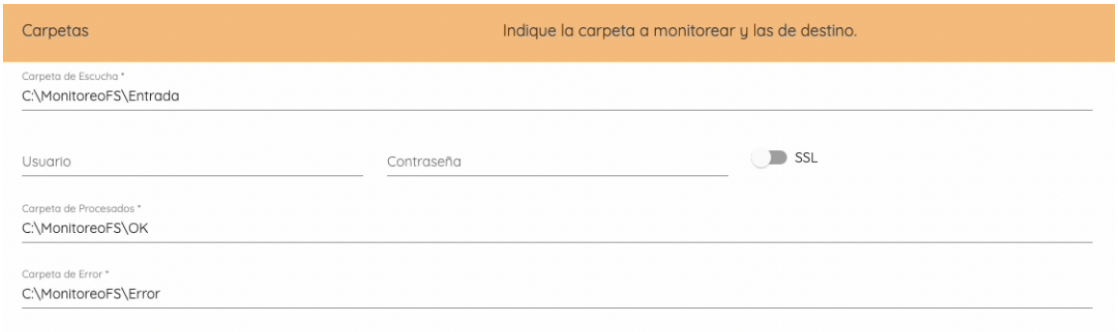
For example, if the channel monitors orders on a file system, we might name it "MonitorOrdersFileSystem".

Then, we add a detailed description, stating the purpose of the channel. A good example would be:

"Monitor the shared order folder and process each new file."

This will make it easier to manage the channel and help other users quickly understand its role.

Step 2: Folder Configuration



Listening Folder

In this section, we set up the folder where BIZUIT will monitor the arrival of new files.

If we work with FTP, we must enable the corresponding option and, if the server does not support anonymous access, we enter the username and password.

If the folder is on a shared network that requires credentials, we must also specify them. In case we use FTPS, we activate the SSL option.

We'll use the local folder C:\FS-Monitoring\Input, but we could set up a shared folder or a location on an FTP server.

Processed Folder

We define the folder where BIZUIT will move the files once they are successfully processed.

This keeps the Listening Folder tidy and allows you to keep track of files that have already been managed.

We'll use the folder C:\FS\OK monitoring.

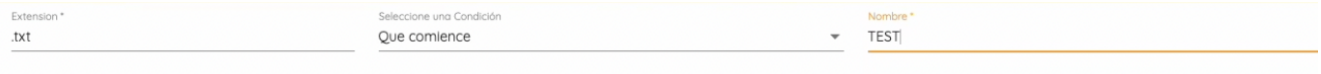
Error Folder
We also set up an Error Folder, where files that could not be processed correctly due to errors in the system or file format will be stored.

This allows us to manually review problematic files and take corrective action.

We'll use the folder C:\FS\Monitoring\Error.

Step 3: File Filter

File Extension
We define a filter for the file types we want to monitor.
For example, if we only want to process .txt or .csv files, we specify that in this section. This prevents unwanted files from being processed.



Additional Terms
We can add **advanced selection criteria**, such as specific file names or patterns in names, ensuring that **only relevant files are processed**.

Step 4: Selecting the Process to Run

Choice of Process
Here we select the process in BIZUIT that will run automatically when a file is detected in the Listen Folder.
This process must meet certain conditions:

- It must have a single required input parameter and be scalar.
- If you have optional parameters, at least one must be scalar.

In our case, the process we'll use has one mandatory parameter scale and two optional parameters. As the mandatory one is a priority, the two optional ones are disabled.

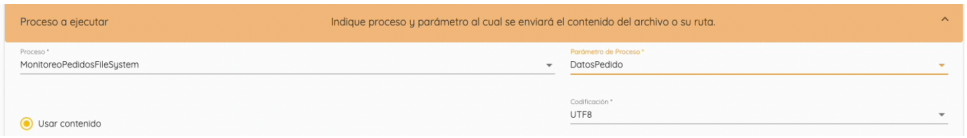
This process sends an email with the contents of the received file. Therefore, we select that mandatory parameter in our settings.

Submission Parameter: Content or File Path
Here we choose whether BIZUIT should send the full contents of the file or just the file path to the selected process.

- **Use content**: BIZUIT reads the entire file and sends its content to the process.
- **Use File Path**: Only the file location is sent, allowing the process to access it directly.

If the file is large, it is most efficient to send only the path to avoid data overload.

Codification
We select the file encoding (e.g. UTF-8), ensuring that special characters are processed correctly.



Step 5: Advanced Settings

Monitoring Interval
In this section, we define how often BIZUIT will check the Listening Folder for new files.
We set up monitoring every 5 seconds for quick detection, but this interval can be adjusted according to the needs of the system.

Step 6: Save Settings and Test

Once all the parameters are configured, we review the settings and press "Save" to activate the channel.
From this moment on, BIZUIT will start monitoring the Listening Folder and execute the configured process whenever it detects a file that meets the set filters.

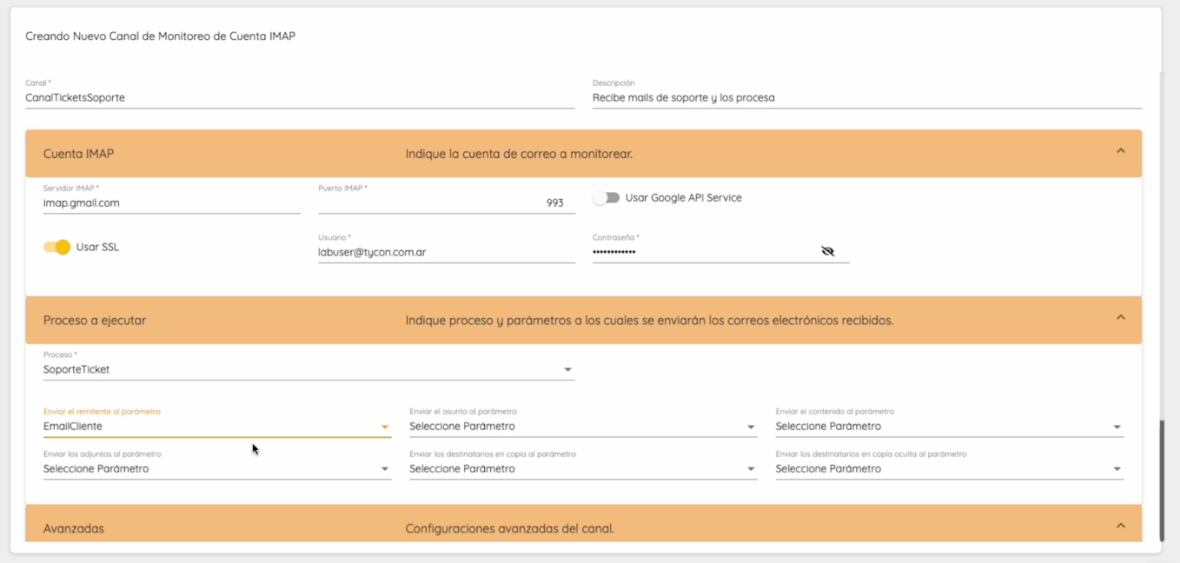
Testing and Validation

- We place a file in the Listening Folder with the extension .txt and the prefix TEST in the name.
- We verify that the file is moved to the Processed Folder. We confirm that the process is running correctly and sends an email with the contents of the file.
- We change the contents of the file and save it back to the Listen Folder.
- We observe that the new file is processed without overwriting the previous one, since BIZUIT assigns it a unique identifier.

IMAP Account Monitoring Channel Configuration

In this section, we will learn how to set up an IMAP Account Monitoring channel in BIZUIT. This type of channel allows BIZUIT to monitor an email account, detect incoming messages, and run a process in response to each new email received.

It is an ideal tool for automating email management, facilitating tasks such as receiving support requests, purchase orders, or any incoming communication that requires automated processing



Step 1: Basic Channel Setup

Channel Name and Description

The first thing we do is assign a name to the channel, which must be descriptive and easy to identify within our system.

We will call our channel "CanalTicketSupport", as its function will be to process support emails.
Below we add a detailed description to indicate the purpose of the channel.
This description will make it easier for other users or admins to understand the channel's role.

Step 2: IMAP Account Setup

IMAP Server and Port

Here we must enter the data of the IMAP server and the connection port of the email account we want to monitor.
If we use Gmail, the server will be imap.gmail.com and port 993 for secure connections.

Security Options

If the IMAP server requires an encrypted connection, we enable the "Use SSL" option.
This ensures the protection of data during transmission, preventing unauthorized access.

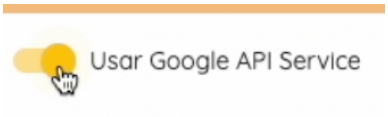
Account Credentials

Here we enter the username and password of the email account that BIZUIT will monitor.

It is important that the credentials are correct, since without them BIZUIT will not be able to access the emails.

Google API Service (Opcional)

If we are setting up a Google account, we can activate the "Use Google API Service" option.



This has an advantage that allows us to integrate directly with Google services, offering a more fluid and secure connection.

Step 3: Configuration of the Process to Run

Process Selection

Next, we choose the process in BIZUIT that will run when a new mail is received. This process can perform a variety of actions, such as:

- Create a support ticket.
- Store mail in a database.
- Send an automatic reply.

In order for a process to be selected, it must meet the following conditions:

- It must have between 1 and 6 mandatory input parameters of type scalar.
- If you don't have required parameters, you must have at least one optional scalar parameter.

Assigning Mail Parameters

Here we configure how the different elements of the mail will be sent to the process in BIZUIT:

- **Sender**: Assigned to a process-specific parameter
- **Subject**: Sent to a selected parameter.
- **Message content**: It is sent to the process for processing.
- **Attachments**: We can assign them to a parameter of the process.
- **Recipients in Copy and Blind Copy**: Can be captured in separate parameters.

This configuration allows us to customize the data flow according to the needs of the business.

Step 4: Advanced Settings

Process emails received from...
Here we indicate from which date we want BIZUIT to process emails.
If we want to include emails from the last 5 days, we set up "Process emails received from 5 days ago".

Write to the Database
If we activate this option, the data of each processed email will be automatically saved in a database.

Benefit: It is useful to have a historical record of the emails processed, which is ideal for document management or subsequent analysis.
Check: In the BIZUIT log viewer we can verify that the channel is working and that the data is being stored correctly.

Step 5: Save Settings and Activate the Channel

Once all the parameters are configured, we perform a final review and press "Save" to activate the channel.
From this moment, BIZUIT will start monitoring the email account and execute the configured process every time a new email arrives.

Step 6: Channel Testing

To make sure the settings are correct, we perform a test:

1. We send an email to the account set up.
2. BIZUIT detects the mail and executes the SupportTicket process.
3. The process receives 1 mandatory parameter and 3 optional parameters.
4. In the BIZUIT interface, we verify that a new task has been generated in User Interaction.

In this task we will see the following columns:

- **Mail:** Sender's address.
- **Subject**: Subject of the email received.
- **Description:** Contents of the email.

If everything works as expected, the channel is set up correctly.

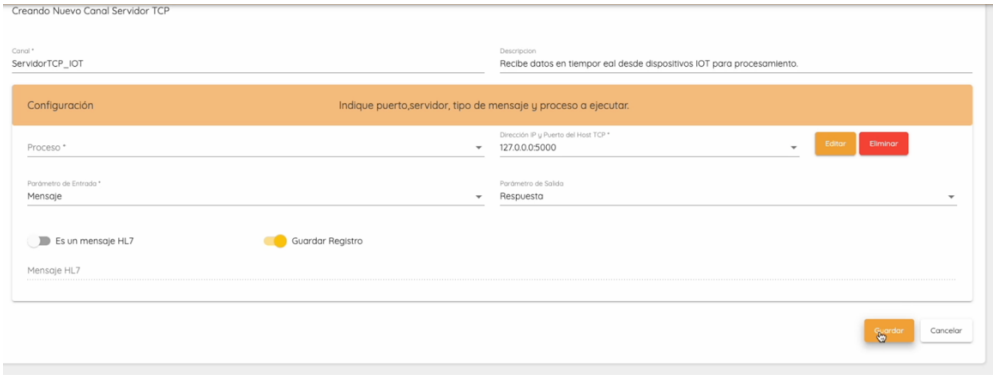
With these steps, we have successfully set up an IMAP Account Monitoring channel on BIZUIT. This allows us to automate email management, run processes in response to new emails, and facilitates integration with other systems and databases.

As a final recommendation we can test the configuration with test emails, verify that all parameters are correctly assigned and also monitor the log viewer to detect possible errors.

This type of channel is essential for the automation of mail management, allowing companies to process messages quickly and efficiently.

TCP Server Channel Configuration

We will now proceed to configure a TCP Server channel in BIZUIT. This type of channel allows us to receive messages directly through the TCP protocol, ideal for real-time integrations in which external devices or systems send data continuously.
Its usefulness is especially relevant in IoT environments, medical devices or industrial communications.
Throughout this section, we'll explore the process of setting up the channel and its server step-by-step.



Step 1: Channel Setup

Channel Name and Description

First, we give the channel a descriptive name, such as "ServidorTCP_DispositivosIoT." This name will help us identify it within BIZUIT. Then, in the description, we specify the purpose of the channel, for example: "Receives real-time data from IoT devices for processing."

Step 2: Process and Parameter Selection

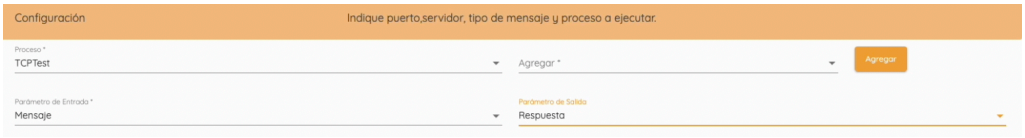
Process

We select the process that we want BIZUIT to execute every time the server receives a message. This process must be pre-configured and could include any action, such as storing the data in a database or generating an alert. We must bear in mind that only those processes will be available that:

- Have one and only one mandatory scalar input parameter, or
- Have at least one optional scalar input parameter and any number of output scalar parameters.

Input and Output Parameter

We set up the "Input Parameter" where the contents of the received message will be stored so that the process can use it. If necessary, we also set an "Output Parameter" so that the process can respond with a message to the sender.



Step 3: Advanced Message Options

If the message we expect to receive is in HL7 format, we activate the option "It is an HL7 message". This format is commonly used in health systems and allows BIZUIT to interpret the message appropriately.

HL7 Message

If we have enabled the HL7 option, we specify the expected message type here. This helps BIZUIT to correctly interpret the structure and content of the message.

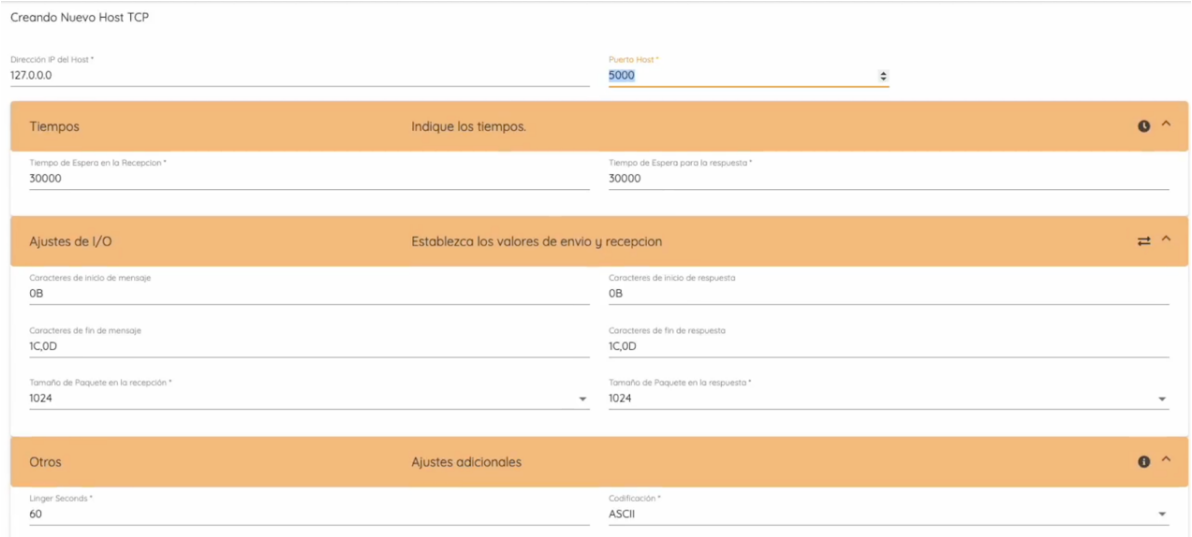
Save Log

We activate "Save Log" if we want to store each message received in a database for later auditing or analysis. This option is especially useful when we need to keep a history of the processed messages.

TCP Server Configuration

Once the channel has been configured, we move on to the configuration of the TCP server, which will be in charge of receiving the messages. Here we set technical details such as IP, port, timeouts, and input/output settings. We can select an existing server or create one from scratch.

Step 4: TCP Host Configuration



IP Address and Host Port

Enter the "Host IP Address" and "Host Port" on which the TCP server will be listening. For example, the IP could be the internal address of the network or a public address, and the port must be open to receive incoming connections (for example, port 5000).

Waiting Time at Reception and for Response

We define the "Wait Time at Reception", which indicates how long the server will wait to receive a message, and the "Wait Time for Response", which determines the time that will be waited before sending a response. These values are set in milliseconds; For example, 30000 milliseconds equals 30 seconds.

Message Start and End Characters

We define the "Message Start Characters" and "Message End Characters" that BIZUIT will use to delimit each message received. These characters are essential for correctly structuring data, especially in continuous communications. An example of a configuration could be the hexadecimal character "0B" for the start and "1C,0D" for the end.

Package Size at Reception and Response

We set the "Package Size at Reception" and the "Package Size at Response". These values determine the amount of data that the TCP server will process in each packet. A size of 1024 bytes is commonly used to handle data in efficient blocks. The choice of size will depend on the length of the message.

Linger Seconds

The "Linger Seconds" parameter defines the additional time for which the TCP server will keep the connection open after sending a response. For example, if we set it to 60 seconds, the connection will remain open for that time to facilitate recurring communications.

Maximum Number of Concurrent Messages

We specify the maximum number of messages that the TCP server can handle simultaneously, as well as the time interval for their processing. This allows us to manage server load and optimize efficiency in receiving messages.

Codification

We select the encryption of the data that the TCP server will use when receiving and sending messages. Although ASCII is a common choice, we can choose other encoding depending on the compatibility needs of specific systems.

Step 5: Save Settings

Once we have completed all the settings, we verify that each configuration is correct and press "Save". Then, we select the server from the list and enable the TCP Server Channel service. From this point on, BIZUIT will start listening on the specified IP and port, ready to receive and process messages in real-time.

With these steps, we have configured a TCP Server channel in BIZUIT, including the channel parameters and TCP host configuration. This type of channel is critical for real-time integrations

and allows BIZUIT to receive and process data from external systems efficiently. It is advisable to test with simulated data to ensure that everything is working correctly.

Kafka Message Monitoring Channel Settings

Now, we will set up a Kafka Message Monitoring channel in BIZUIT. This channel will allow us to connect BIZUIT to a Kafka system to consume messages on a specific topic.
It is ideal for integrating real-time data streams and responding to events generated in multiple applications or systems, such as IoT, data analytics, or event processing.
Next, we'll set up the basics and advanced aspects of this channel.

Channel Name and Description
First, we assign a descriptive name to the channel, such as "ChannelKafkaTopicCustomers". This name will help us identify it within BIZUIT.
Then, in the description, we specify the purpose of the channel, for example: "Receive real-time data from online customer systems."

Step 1: Basic Kafka Account Setup

Broker Server
We configure the "Broker Server", which is the address of the Kafka server we want to monitor. This broker will be the connection point of BIZUIT to the Kafka system.

Group ID

We specify the "Group ID", which will allow us to coordinate multiple consumers in reading messages, ensuring that each message is processed only once. It is important to define this ID correctly so that BIZUIT can identify messages appropriately.

AutoOffsetReset

We define the "AutoOffsetReset" option, which determines from which point we will start consuming messages if there is no previous offset (offset). We can choose between:

- **"Earliest"**: Starts from the oldest message.
- **"Latest"**: Starts from the most recent message.

This configuration will allow us to control the starting point of the consumption of messages in the system.

Topic

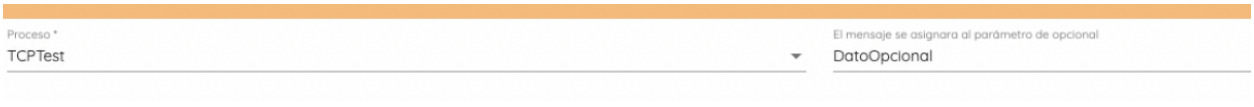
Enter the name of the "Topic", which is the communication channel in Kafka where the messages that BIZUIT will process are published.

Username and Password

If the Kafka server requires authentication, we enter the corresponding username and password. This will ensure that only authorized users can access the topic's data.

Step 2: Configuration of the Process to Execute

We select the process in BIZUIT that will be executed every time we receive a message from Kafka. This process must be pre-configured and may include actions such as storing data in the database, triggering alerts, or initiating other workflows.
We must also select the parameter of the process to which the message will be sent.



Write to the Database

If we want to record each message received in a database, we activate the "Write to the Database" option. This will allow us to keep a history of the messages and perform subsequent analysis.

Step 3: Advanced Settings

High importance

ApiVersionRequest, EnableAutoCommit, EnableAutoOffsetStore

- We activate "ApiVersionRequest" to request the version of the Kafka API, which will help us ensure compatibility with different versions of the server.
- We enable "EnableAutoCommit" so that BIZUIT automatically confirms receipt of each message.
- We activate "EnableAutoOffsetStore" to store the offsets automatically.

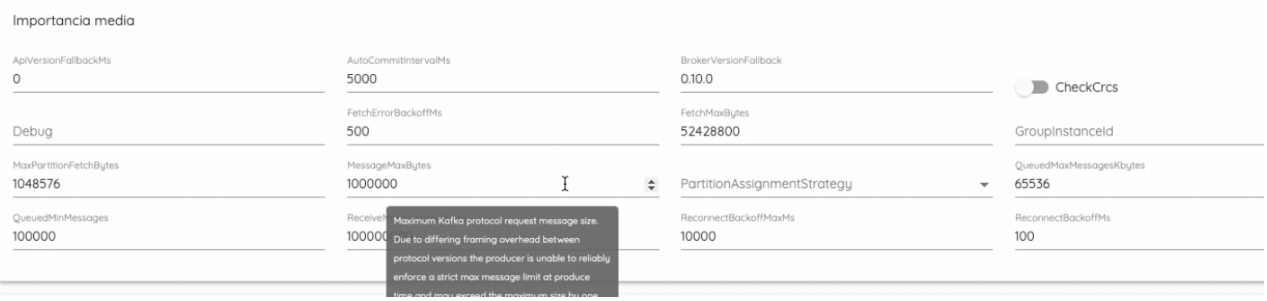
SecurityProtocol

We select the appropriate "SecurityProtocol" for the connection, choosing between options such as SSL or SASL_SSL, depending on Kafka's security settings. This will ensure encrypted and secure communication.

SessionTimeoutMs y MaxPollIntervalMs

We set "SessionTimeoutMs", which defines the session timeout in milliseconds.
We also specify "MaxPollIntervalMs", which determines the maximum interval between each message read to keep the consumer active.

In medium and low importance, we define each of the necessary fields. We can consult the online documentation to learn the purpose of each.



Step 4: Save Settings

Once we have completed all the settings, we review the settings to make sure they are correct and press "Save". Then, we enabled Kafka's Message Monitoring service.
From this moment on, BIZUIT will start consuming messages of the specified topic and executing the configured process in real time.

With these steps, we have set up a Kafka Message Monitoring channel in BIZUIT.

This channel is essential for real-time integrations with event systems and will allow us to consume and process large volumes of data efficiently. It is advisable to test the configuration to verify that messages are received correctly and that the process runs as expected.

Message Queue Managers Channel Settings

We will now set up a Message Queue Manager channel in BIZUIT. This channel will allow us to manage the queue of messages and define how they will be processed sequentially or in case of errors.

It is especially useful in applications where we need to ensure message delivery or run processes reliably, with retry options in case of failures.

We'll explore every aspect of setting up this channel.

Step 1: General Message Queue Settings

Name of the Message Queue

We assign a descriptive name to the message queue, such as "DemoQueue". This name will allow us to identify the queue within the system and will facilitate its management when we have multiple queues configured in BIZUIT.

Process

We select the process we want to run when a message is received in the queue. In this case, we use "AsynchronousProcess", a process that queries the customer base based on an ID received. This process must be previously configured in BIZUIT and prepared to receive and process the corresponding data.

We must bear in mind that only those processes will be available that:

- Have one and only one required scalar input parameter.
- Or at least an optional scalar input parameter.
- They can have as many output scalar parameters as we need.

Input Parameter

The "Input Parameter" defines the field where the message information will be stored. In this case, we select "QueueCustomerData", which will contain the information that the SQL query needs to execute successfully.

Output Parameter

The "Output Parameter" is where the process will send the response or processing result. In this example, we set "CustomerCountry" as the output field for our asynchronous process. This configuration ensures that the result of the process is available for future reference or action.

Step 2: Configure Retries

Number of Retries

In the "Retry" field, we specify the number of attempts that BIZUIT will make in case the process fails. In this example, we set the system to try three times if an error occurs. This allows us to have multiple opportunities to process the message in case of temporary problems.

Retry Interval

We define the time interval between each Re-Run attempt. In this case, we set up the following intervals:

- **First attempt**: 5 seconds.
- **Second attempt**: 15 seconds.
- **Third attempt**: 30 seconds.
-

This means that if the process fails, BIZUIT will wait 5 seconds before trying to process the message again. If the second attempt also fails, it will wait 15 seconds before trying again. If the third attempt also fails, it will wait 30 seconds. If after these attempts the message fails to process successfully, it will be sent to the dead-letter queue.

This approach helps reduce the load on the system and allows you to resolve transient issues that could be causing the process failures.

Step 3: Enable the Channel

Enabled

Finally, we turn on the **"Enabled" switch** to make sure the queue is operational and ready to receive and process messages. If this option is not enabled, the channel will not be active and will not process any messages.

Step 4: Save Settings

Once we have configured all the parameters, we review the settings to make sure everything is correct. Then, we press "Save" and enable the Message Queue Manager channel. From this point on, BIZUIT will handle the messages in the queue according to the configured parameters, including retries in case of error.

With these steps, we have set up a Message Queue Manager channel in BIZUIT.

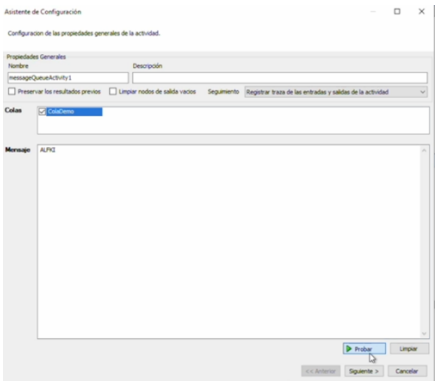
This channel is critical to ensure reliable message delivery and execute processes in a controlled manner, especially in critical applications that require efficient error handling and automatic retry.

Now, we'll perform configuration tests to verify that messages are processed correctly and that the system responds appropriately in the event of failures.
Testing the Message Queue Managers Channel
Now, we'll explore how to test the BIZUIT message queue manager works. We start by creating a process that receives, as parameters, the data we want to send to the process that will manage the queue.

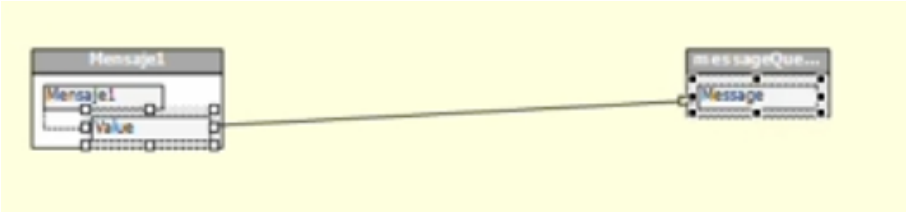
Step 5: Creating the Testing Process

Once we've created the process, we set up a Message Queue activity. In it, we select the queue to which we want to send the message. In our case, the queued process (called *AsynchronousProcess)* is designed to receive an Employee ID, and we'll use the ALFKI client ID as an example.

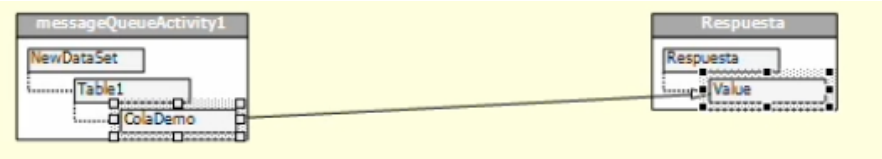
When entering the customer ID, we press the Test button. If everything works correctly, the system will return a message ID, confirming that the message has been successfully queued.



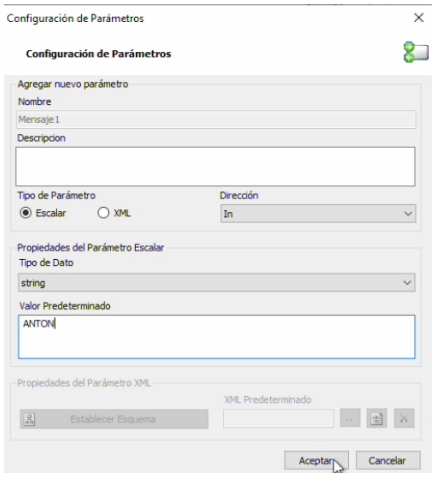
We proceed to map the input parameter of the process, called *Message1*, with the *Message* parameter expected by the activity.



Next, we map the *DemoQueue* node returned by the activity to the process's output parameter, called *Response*, so that we can return the Id of the queued message.

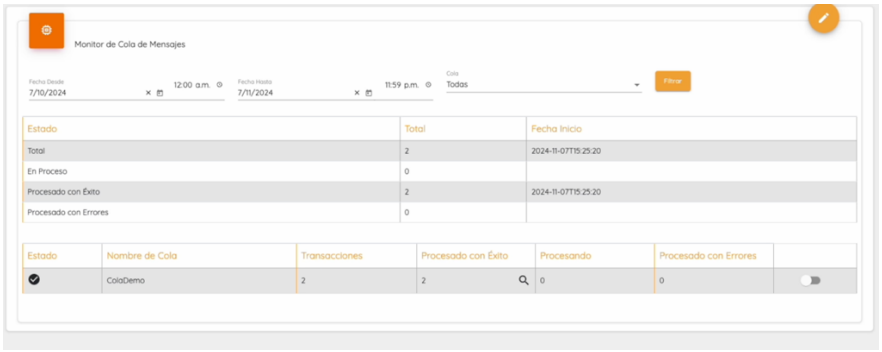


Then, we set the default value of the input parameter *Message1* to the value "ANTON" and run the process.



Step 6: Configuring the Queue Viewer in BIZUIT Dashboard

We enter the Message Queue Viewer module in the BIZUIT Dashboard, which allows us to view and monitor the status of the queued messages.



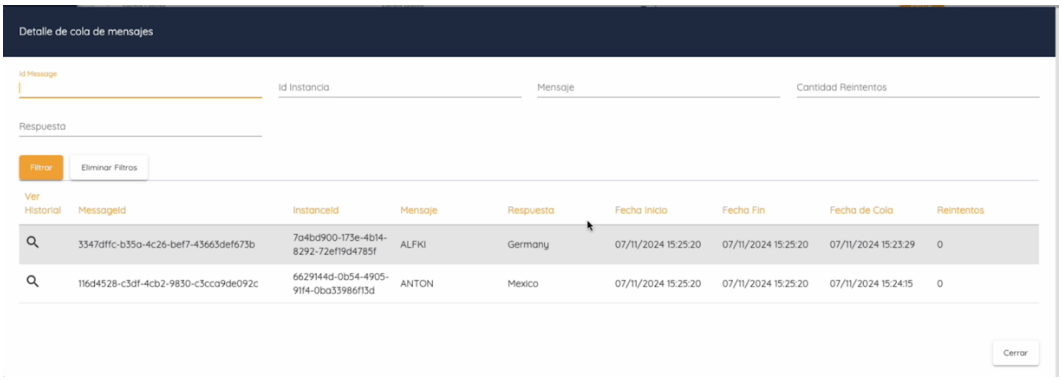
General Monitoring Screen

On the general monitoring screen we see a summary of the messages in the configured queues, which includes the following statistics:

- **Total: The** total number of messages processed.
- **In Process: Messages** that are currently being processed.
- **Successfully Processed:** Number of messages processed without problems.
- **Error Processing:** The number of messages that failed during processing.

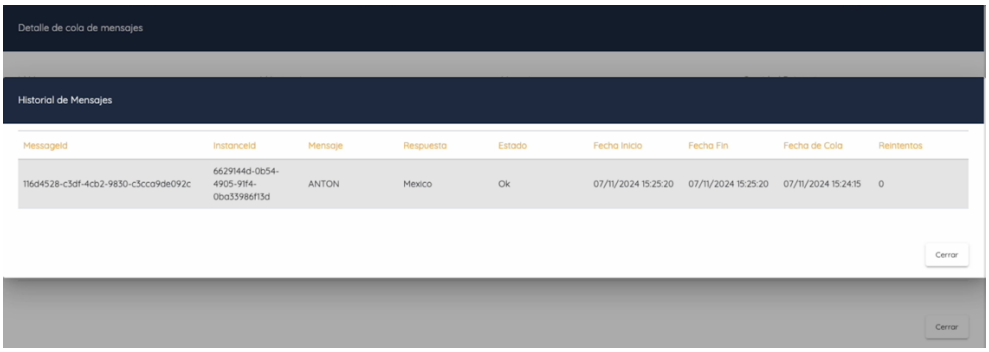
Message Queue Detail Screen

When refreshing the view, the detail of the *Successfully Processed* messages is displayed, allowing us to access more specific information. Here we can visualize, for each message, the following data:



- **MessageId: Unique** identifier for the message.
- **InstanceId:** Identifier of the instance that processed the message in BIZUIT.
- **Message:** Content of the message, in our example, the employee ID (e.g., *BOLID* or *ALFKI).*
- **Response: Result** or response generated by the queued process when processing the message.
- **Start Date and End Date:** indicate the start and end times of processing.
- **Queue Date:** The time at which the message was queued.
- **Retries –** The number of times the message has been attempted, which helps us identify potential issues.

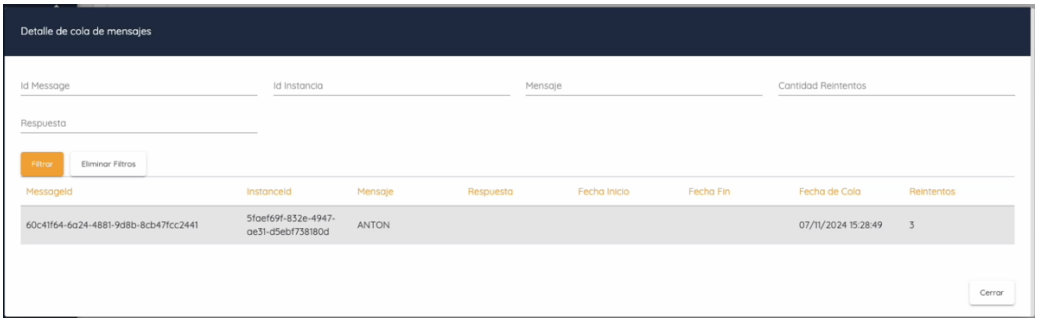
In addition, we can visualize the execution trace of that instance that was successfully processed.



Retry Verification

To check the retries are working, we intentionally cause the queuing process to fail. To do this, we insert an exception activity, republish the process, and run the queuing process again.

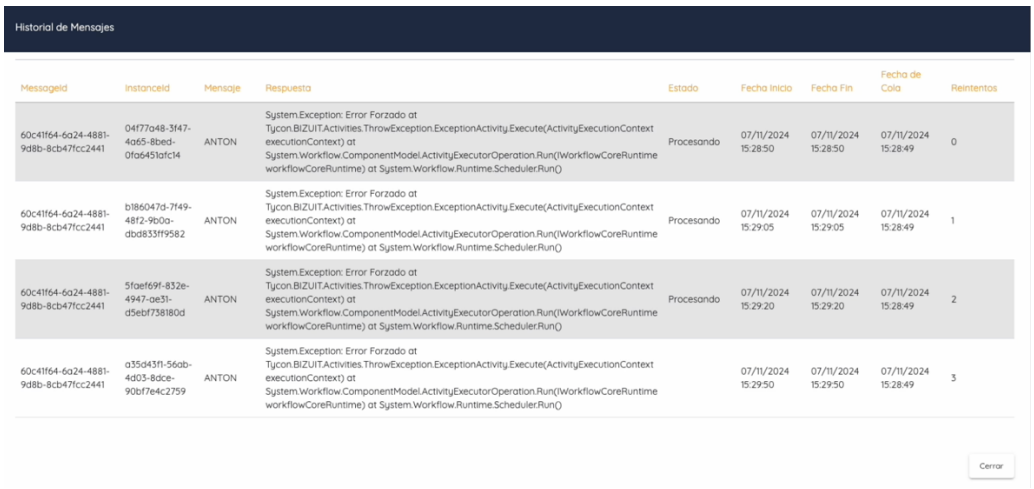
We will notice that, after starting the execution, a message in process appears. When we refresh our eyes, we notice that three retries have been recorded, until finally the message stops being retried.



We access the details of the messages processed with errors and examine the specific information of each one; in our case, only one has been registered.

Here we can identify the exception or error that occurred during processing, which is critical for troubleshooting as it allows us to know in detail the cause of the failure, the activity involved, and the reprocessing attempts.

If we select the detail option (by clicking on the magnifying glass of a specific message), we access a complete record of all processing attempts.



This section shows each attempt with its status, the exact date and time of each processing, and any errors that may have occurred.

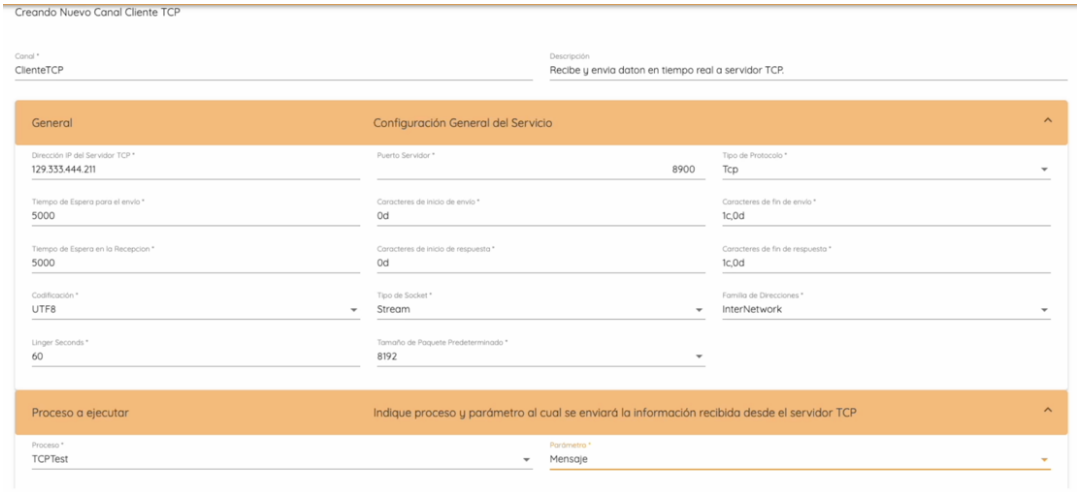
Finally, if we want to forward a message that failed, we have the forwarding option so that it can be processed again. This is especially useful once the problem that caused the error has been fixed, allowing us to process messages without needing to re-enter them.

We proceed to remove the error activity and republish the queued process. Then, we forward the message from the dead-letter console and verify that, this time, it is processed correctly.

With this demo we have learned how to configure and test the message queue manager in BIZUIT, interpret the detailed information of the queue viewer module and effectively use the tools to manage messages.
This process is critical to ensure that the flow of messages in our processes is carried out in a controlled manner, allowing us to identify and resolve any issues that may arise.

TCP Client Channel Configuration

In this section we will learn how to configure a TCP Client channel in BIZUIT. This type of channel allows us to establish a connection with an external TCP server to send and receive data in real time, which is essential for integrating BIZUIT with remote devices or applications.



Initial Channel Setup

Name and Description Assignment

We start by giving the channel a descriptive name, for example, ClientTCP.
In the description, we specify its purpose, such as: "Receives and sends real-time data to a TCP server."
This will help us easily identify the channel in the BIZUIT interface.

General Service Settings

In this section, we configure the basic parameters for the connection:

TCP Server IP Address

Enter the IP address of the server that BIZUIT will connect to. Let's verify that the IP is accessible from the BIZUIT network.

Server Port

We define the Server Port that will be used for communication.
Many services use port 5000 or 8080, but this value will depend on the server configuration.

Type of Protocol

We select the Protocol Type:

- TCP: For reliable connections.
- UDP**:** For faster transmissions, although with less security.

Waiting Times

We configure maximum wait times for:

- Sending: The time BIZUIT will wait to send a message.
- Receive: The time you will wait for the server's response.
- Example: 5000 ms (equivalent to 5 seconds) for each.

Message Start and End Characters

We define the characters that indicate:

- Start and end of the sent message: This clearly delimits the content of the message.
- Start and end of the response**:** So that BIZUIT knows when the server's response starts and ends.

Codification

We select the appropriate encoding for data exchange, e.g. UTF-8 or ASCII, which ensures the correct interpretation of special characters.

Socket Type and Address Family

We choose the Socket Type and Address Family (e.g., IPv4 or IPv6) based on the network and server we communicate with.

Default Package Size

We set the maximum size of data that will be sent or received in each package. This is essential to efficiently handle large volumes of information.

Configuration of the Process to Be Executed

We choose the BIZUIT process that will be executed when information is received from the TCP server. In this case, we use the TCPTest process.

Parameter Definition

We assign the parameter that will receive the information from the server. We use the Message parameter, which will contain the input data.

Save Settings

Before finishing, we check that all the details are correct. Then, we press the Save button to activate the TCP Client channel. From that moment on, BIZUIT will be ready to connect to the TCP server and manage the sending and receiving of data according to the parameters we have configured.
Using these steps, we have effectively configured a TCP Client channel in BIZUIT.

This channel is a fundamental tool to ensure real-time communication with external servers, allowing us to integrate and automate the exchange of data in a reliable way. It is advisable to test the configuration with the corresponding server to confirm that everything is working as expected and adjust any parameters if necessary.

With this guide, we want you to understand in a clear and practical way each of the components necessary to implement and make the most of a TCP Client channel in BIZUIT.

Unit Overview

In this unit, we address how to set up different types of channels in BIZUIT, starting with the REST API channel, which allows you to expose services and access real-time information. Through progressive steps, we learned how to define a channel, configure your endpoint, select input and output parameters, and activate security options.

We also explore how to test these services using tools like Postman, as well as detailing how to set up Web Service channels, Process Schedulers, and IMAP File and Account Monitoring. All of these channels automate processes, improving integration and operational efficiency.

We explore the configuration of various channels in BIZUIT, from creating a REST API channel to expose services in real time to configuring web services, process schedulers, and channels to monitor files, IMAP emails, TCP servers, and Kafka messages. Each configuration starts by assigning a descriptive name and a detailed explanation that clarifies the purpose of the channel, defining endpoints, input and output parameters, and establishing security and authentication measures to ensure secure and efficient communications.

In addition, practical aspects such as integration with specific BIZUIT processes, the automation of periodic tasks and the implementation of retries to handle errors in message transmission were addressed. In summary, this unit offers a comprehensive and didactic guide that allows users to configure, test and optimize different communication channels, facilitating the integration of BIZUIT with external systems and improving the management and supervision of processes in complex environments.

Unit 4: Best Practices in Channel Configuration

In this unit we will delve into the best practices for channel configuration in BIZUIT. In our educational approach, we will not only focus on getting the channels to work properly, but also on ensuring their safety, efficiency and reliability over time.

Below, we'll review how to protect our channels, monitor their activity, and optimize their performance to ensure continuous, seamless operation.

Secure Channel Configuration

At BIZUIT we know that security is the foundation of any integration system. When setting up a channel, whether it's REST API or web services, it's critical that we implement strong authentication methods. We recommend enabling token-based authentication or API keys, so that only authorized users or applications can access the information.

Let's periodically review permissions and apply the principle of "least privilege", ensuring that each user has access to only what is strictly necessary. In a REST API channel that exposes customer data, we assign a specific API key and update it regularly to strengthen security.

Update Rates and Connection Limits

Each channel must operate with an update frequency according to its function. For example, in file monitoring channels or IMAP accounts, you don't need to check for changes every second; A frequency of 10 to 15 minutes is usually sufficient.

Let's set connection limits to prevent multiple concurrent requests from overloading the system. In the case of a Kafka Message Monitoring channel, for example, it is prudent to limit the number of concurrent consumers to prevent conflicts in data processing. If we manage a File Monitoring channel that monitors an FTP folder shared by multiple vendors, an update every 5 minutes can balance efficiency and system load.

Encryption and Secure Connections

For channels such as TCP Servers and Clients, it is essential to enable secure connections (SSL/TLS) that protect the integrity of the data during transmission.
Always use up-to-date SSL certificates and avoid outdated protocols.

On any channel that supports secure connections, make sure the latest version of TLS is used. On a TCP channel that receives data from medical devices, encryption ensures that the information cannot be intercepted or tampered with.

Activity Monitoring and Auditing

Alert Settings
Configuring alerts allows us to detect problems in the operation of the channels at an early stage. BIZUIT allows us to trigger notifications that are triggered in case of connection errors, overloads or failed retries.
Let's use "failed retries" alerts to quickly identify critical issues, such as in File Monitoring channels or IMAP accounts, and act before incidents accumulate.

If an IMAP Account Monitoring channel detects three consecutive access failures, an alert will alert us to review the authentication and fix the issue.

Records and Auditing
Maintaining a detailed history of our channel activity is key to both issue analysis and compliance. BIZUIT allows us to enable audit logs that document all important events.
Let's activate the audit log on critical channels and store this information in a database, thus facilitating the review and tracking of possible incidents.

In a REST API channel that handles customer requests, auditing will allow us to know who accessed each resource, which is essential in case of queries or disputes.

Use of Real-Time Monitoring Dashboards
Integrating dashboards in real time helps us to immediately visualize the health and performance of our channels. With these tools, we can monitor activity, resource consumption, and detect potential bottlenecks.

Configure specific dashboards for high-load channels, such as Kafka, and adjust the settings if we notice spikes in message volume. If we notice a sudden increase in the flow of messages in a Kafka channel, we can temporarily allocate more resources or limit consumption until the flow stabilizes.

Optimization and Maintenance Tips

Adjustment of Performance Parameters

As the workload increases, it's vital to adjust the performance parameters of our channels. This includes the size of the data packets, the intervals of retries and the number of concurrent connections.

For TCP Server channels, let's experiment with different packet sizes to optimize data transmission and avoid bandwidth saturations. A TCP channel that transmits sensor data can benefit from larger packets, allowing for smoother and uninterrupted transmission.

Load Testing and Bottleneck Monitoring

Performing load tests on a regular basis allows us to verify that the channels support the volume of data smoothly and detect potential bottlenecks.
Let's establish a test environment where we simulate the maximum expected load and analyze the behavior of the channel, adjusting parameters such as waiting times or concurrent message limits.
In a Kafka channel that receives real-time sales data, load testing can reveal the need to tune the system to handle multiple messages per second without impacting performance.

Preventive Maintenance and Documentation

Preventive maintenance is essential to ensure the continuous operation of our channels. This involves regularly reviewing configurations, updating credentials, and cleaning up old records to avoid overcrowding.

We schedule quarterly channel reviews and document each change in a centralized document, ensuring a smooth transition in case of modifications or changes in the support team. If we modify the retry interval in an IMAP channel, document the change and the rationale so that any team member understands the modification.

With these practices, we ensure that the channels in BIZUIT not only function optimally, but are also proactively secured, monitored, and maintained.

By applying these tips and strategies, we ensure a robust and resilient integration environment, ready to meet any challenge and adapt to the growing needs of our systems.

Chapter Summary

In this chapter, we explored how channels in BIZUIT enable the integration and automation of business processes, facilitating communication between internal and external systems. Its ability to update data in real time, reduce manual intervention and improve operational efficiency in sectors such as health and logistics was highlighted.

The concept of channels was introduced, explaining their role as automated connectors that optimize the synchronization of information and speed up decision-making. The different types of channels available in BIZUIT were analyzed, including REST APIs, web services, process schedulers, IMAP account and file monitoring, TCP servers and clients, Kafka messages, and message queues, each with specific applications for workflow automation and optimization.
Use cases were illustrated where channels enabled everything from efficient medical records management to automating order updating and real-time data transmission in distributed architectures.

Subsequently, the practical configuration of the channels was addressed, detailing the process to define endpoints, establish input and output parameters, configure security options and perform integration tests with tools such as Postman.
It was explained how to expose internal processes such as REST or SOAP services, schedule automatic tasks with process schedulers, activate workflows with file and mail monitoring, and establish TCP and Kafka servers for real-time event processing. Message queue management was also covered, ensuring that processes run in a controlled and reliable manner.

Finally, good practices were presented to guarantee the safety, efficiency and sustainability of the configured channels.
Authentication and access control strategies, performance optimization through the correct configuration of frequencies and limits, monitoring with alerts and dashboards, and error management with retry and automatic recovery mechanisms were addressed.

Throughout the chapter, the knowledge about channels in BIZUIT was consolidated, demonstrating their importance in the integration of systems and the automation of processes. Tools and methodologies were provided to implement these channels efficiently, ensuring that their use is secure, scalable and aligned with business needs.