Predictive Routing Deployment and Operations Guide

Genesys Predictive Routing 9.0.0

2/9/2022

Table of Contents

Welcome to the Deployment and Operations Guide

This guide covers the following topics, enabling you to plan, set up, and maintain your Genesys Predictive Routing (GPR) environment. Predictive Routing enables you to match interactions with agents for optimal outcomes.

- **Planning:** system requirements, architecture and security, sizing, data preparation
- **Configuring and installing** single-server and HA environments:
  - AI Core Services Single-Host Deployment
  - Deploy in High Availability Environments: HA deployment prerequisites and procedures for all GPR components
  - Deploy Agent State Connector (ASC)
  - Configuration Options
  - Start and Stop All GPR Components
- **Operations**: logging, backing up and restoring, troubleshooting
- **Genesys Routing integration**: deploying and optimizing the GPR subroutines components
- **Genesys Reporting integration**: configuring GPR to work with the Genesys Reporting components
- **Quick Start**: to get new users up and running
- **New Features**: lists all new features by release and includes links to documentation describing them
- The Genesys Predictive Routing Overview video presents a high-level picture of what Predictive Routing can do for you:
  Link to video
- Conceptual topics go in-depth on certain aspects of GPR functionality:
  - How does GPR Score Agents?
  - Routing Scenarios Using GPR

Looking for something else?

Consult the following guides for other GPR topics:

- *Genesys Predictive Routing Help* shows how to use the GPR application and gives in-depth explanations of the following GPR functionality:
  - The Lift Estimation, Feature Analysis, and Agent Variance Reports
  - Agent Profile and Customer Profile schemas
  - Datasets

    - Predictors and Models
    - Accounts, User Management, Password Management, and Auditing
- *Predictive Routing API Reference* (Requires a password for access. Please contact your Genesys representative if you need to view this document.)

The home page provides links to the following resources:

- Release Notes for all GPR components
- GPR Read Me

Quick Start

To use Genesys Predictive Routing (GPR), you'll need to install and configure the following products and components:

Install

1. Install any Genesys components that aren't already part of your environment. You'll need:
   - Genesys Framework
   - Genesys Administrator Extension
   - Stat Server
   - Universal Routing Server and Interaction Routing Designer
     OR

     Orchestration Server, Universal Routing Server and Interaction Routing Designer, and Composer
   - Interaction Concentrator
   - Genesys Info Mart
2. Install Genesys Predictive Routing, which consists of the following components:
   - Agent State Connector (ASC)—Synchronizes agent state data from Stat Server (agent availability) and Configuration Server (agent profile) for use by AICS.
     In ASC release 9.0.015.04 and later, you can choose to have ASC read agent availability information from Universal Routing Server (URS) rather than from Stat Server, minimizing the number of components ASC connects to.
   - Integrate with Genesys Routing
   - AI Core Services (AICS)—Provides the Genesys Predictive Routing scoring engine, the user interface, and the API.

Configure

To complete your setup of Predictive Routing, configure the following components:

1. Set the desired values for the configuration options.
   You use configuration options to configure a wide range of application behavior, including:
   - The mode GPR is running in, which might be `off`
   - Login parameters and access URLs
   - KPI criteria to decide what makes for a better match
   - Scoring thresholds, agent hold-out, and dynamic interaction priority.
   - Many other important functions

2. To configure how the match between interactions and agents is determined, configure Predictors and Models, as explained in the *Predictive Routing Help*.

Import Data

Using the Predictive Routing interface, you import a dataset that is available in CSV format. A *dataset* is a collection of raw event data. The primary purpose of a Dataset is to be the source of Predictor data.

- GPR automatically analyzes the data and creates a schema, identifying the various types of data you are importing.
- You can adjust the schema during the import process.
- After the Dataset has been imported, you can append additional data as long as it is consistent with the schema that has already been established.

Create Predictors and Models

Predictors are based on the dataset information that you have imported and that has been analyzed into a schema.

- A Predictor defines a view on that underlying dataset. It can select from some or all of the data in the dataset; you can use a predictor with multiple datasets.
- A Model is based on a Predictor, and uses the same target metric or KVP as that Predictor. You can configure multiple Models for each Predictor. These Models can use different selections of the features available in the underlying dataset. Models are the objects actually used to perform agent scoring and interaction matchups.

As you configure a Predictor, you can choose which metric you want to work with, what kinds of situations you want to evaluate, and other parameters, constructing a way to determine the Next Best Action in the specified situation, based on the possible actions available at that time. As you gather more data, you can add that new data to your dataset, and have the Predictor test against the actual results coming in, enabling you to refine how successful your Predictor is.

Reporting and Analysis

You can report on various parameters, such as:

- The success of your predictors.
- The results of A/B testing.
- The factors affecting a KPI you are trying to influence.

Reports are available through the following reporting applications:

- The Predictive Routing interface. See Creating and Interpreting Reports in the Predictive Routing Help for specific information.
- Genesys Interactive Insights/GCXI, as part of the Genesys historical reporting offering. The following reports are available in the *Genesys Customer Experience Insights User's Guide*:

  - Predictive Routing - AHT & Queue Dashboard
  - Predictive Routing - Model Efficiency Dashboard
  - Predictive Routing Agent Occupancy Dashboard
  - Predictive Routing A/B Testing Report
  - Predictive Routing Detail Report
  - Predictive Routing Operational Report
  - Predictive Routing Queue Statistics Report
- Pulse, where the Agent Group KPIs by Predictive Model and Queue KPIs by Predictive Model templates for real-time reporting are available from the Genesys Dashboard Community Center.

New Features Log

This topic provides a single point to find new feature information included in Genesys Predictive Routing (GPR). It includes the software version number and release date, and includes links to the related documentation updates (if any were needed).

List of New Features and Modifications

The following significant changes in functionality were made in 9.0.0 releases.

Feature Description	Type of Change	Occurred in Release	Documentation Updates
The addition of nine new KVPs enables detailed reporting on scoring results. In addition, the gpmResult and gpmMode KVPs have new valid values, enabling more precision in reporting on routing outcomes.	New feature	URS Strategy Subroutines 9.0.015.00	*Deployment and Operations Guide* • GPR KVPs for Genesys Reporting
You can now log scoring details that you can then use to monitor and better understand the scoring process and outcomes. This release also includes scripts to clean up unneeded score logs from MongoDB.	New feature	URS Strategy Subroutines 9.0.015.00	*Deployment and Operations Guide* • View the Scoring Logs
The URS strategy subroutines include the new GPRIxnSetup and GPRIxnCleanup subroutines, as well as some modifications to existing subroutines. These enable you to pin-point when GPR started handling interactions.	New feature	URS Strategy Subroutines 9.0.015.00	*Deployment and Operations Guide* • Deploy the URS Strategy Subroutines
The gpmWaitTime value is now calculated using START_TS rather than gpm-ixn-timestamp.	Improvement	URS Strategy Subroutines 9.0.015.00	*Deployment and Operations Guide* • Deploy the URS Strategy Subroutines
The GetActionFilters subroutine was enhanced to identify the list of agents matching the target skill group along with the configured login status expression. This information is also reported in the action filters of the scoring request. This functionality is invoked only when the **use-action-filters** configuration option	New feature	URS Strategy Subroutines 9.0.015.00	*Deployment and Operations Guide* • **use-action-filters**

Feature Description	Type of Change	Occurred in Release	Documentation Updates
is set to `false`.			
Two new documentation pages, Routing Scenarios Using GPR and How Does GPR Score Agents? provide detailed discussions of those aspects of GPR functionality.	New feature	AICS 9.0.015.03/ ASc 9.0.015.04/ URS Strategy Subroutines 9.0.015.00	*Deployment and Operations Guide* • Routing Scenarios Using GPR • How Does GPR Score Agents?
The *Deployment and Operations Guide* now contains complete instructions for configuring HTTPS connections among all GPR components.	New feature	AICS 9.0.015.03/ ASc 9.0.015.04/ URS Strategy Subroutines 9.0.015.00	*Deployment and Operations Guide* • Configure GPR to Use HTTPS
New API endpoints have been added, enabling you to use the MinIO container to upload Agent Profile and Customer Profile data. Previously, only Dataset data used the MinIO container, which provides a performance improvement over the data_upload worker alone.	New feature	AICS 9.0.015.03	*Genesys Predictive Routing API Reference* • Predictive Routing API Reference (Requires a password for access. Please contact your Genesys representative if you need to view this document.)
The `numeric` datatype now replaces both `float` and `integer` datatypes in Agent and Customer Profiles. This change resolves confusion about when to use `float` and `integer` datatypes. Important This change has been made in both AICS 9.0.015.03 and ASC 9.0.015.04. Neither AICS 9.0.015.03 nor ASC 9.0.015.04 is compatible with earlier releases of these components. If you upgrade one, you must also upgrade the other.	New feature	AICS 9.0.015.03; ASC 9.0.015.04	*Deployment and Operations Guide* • GPR KVPs for Genesys Reporting
The connection between ASC and Stat Server is now optional. ASC can now read agent availability information from URS, reducing the number of connections necessary in your environment. Important This increases the load on URS. Contact your Genesys representative for sizing guidelines. If you use custom statistics for Predictive	New feature	ASC 9.0.015.04	*Deployment and Operations Guide* • Deploy Agent State Connector

Feature Description	Type of Change	Occurred in Release	Documentation Updates
Routing, such as Agent Occupancy, you must retain the connection to Stat Server.			
ASC now validates the values you enter in the **include-skills** and **include=groups** configuration options. If ASC detects a skill name or Agent Group specified in these options that does not exist in Configuration Server, ASC triggers a Standard-level log message (Message Server log event number 60401).	New feature	ASC 9.0.015.04	*Deployment and Operations Guide* • **include-skills** • **include-groups**
The GPR API now returns a `file_path` parameter in the response message when you request a presigned URL, which replaces the uploaded file name. You must now pass this `file_path` parameter in requests to create Datasets or the Agent Profile or Customer Profile instead of the uploaded file name, used in previous releases.	New feature	AICS 9.0.015.03	*Genesys Predictive Routing API Reference* • Predictive Routing API Reference (Requires a password for access. Please contact your Genesys representative if you need to view this document.)
The GPR web application has enhanced security by logging out inactive users.	New feature	AICS 9.0.015.03	*Predictive Routing Help* • Welcome
GPR now accepts only CSV and zipped CSV files for upload to Datasets, the Agent Profiles, and the Customer Profile. JSON file uploads are no longer supported.	Discontinued support	AICS 9.0.015.03	
The NGINX container has been removed from AICS. NGINX is an optional load balancer that had been provided only for use only in test environments.	Discontinued support	AICS 9.0.015.03	*Deployment and Operations Guide* • High-Level Architecture
The method for configuring HTTPS for Agent State Connector has been changed. Previously configuration was done by setting the desired value in the USE_HTTP environment variable, a procedure that was applicable only for Linux-based environments. Now, HTTPS is configured in the **jop-base-url** configuration option.	Improvement	URS Strategy Subroutines 9.0.015.00	*Deployment and Operations Guide* • Configure HTTP/HTTPS Connections
This release includes an updated and improved version of the *Predictive Routing API Reference*.	Improvement	AICS 9.0.015.00	*Genesys Predictive Routing API Reference*

Feature Description	Type of Change	Occurred in Release	Documentation Updates
In particular, there are now cURL request examples for each endpoint.			• Predictive Routing API Reference (Requires a password for access. Please contact your Genesys representative if you need to view this document.)
AI Core Services now requires Docker version 18.09.2, which addresses important security issues.	New supported platform	AICS 9.0.015.00	See the Release Notes for AI Core Services 9.0.015.00 for important information about the reason for this change and for Docker deployment information.
You can now convert a regular account into an LDAP account.	New feature	9.0.015.00	*Predictive Routing Help* • Configuring Accounts
This release includes the following improvements to the user interfaces in the GPR web application: • A new navigation panel provides a tree view of all Datasets, Predictors, and Models configured for the current Tenant. This tree-view pane is available from the Settings > Datasets and Settings > Predictors windows. Each item in the tree view links to the specified object, enabling easy access to the entire hierarchy of Datasets, Predictors, and Models. • For simplified navigation, breadcrumb links now appear at the top of windows in the GPR web application if you have drilled-down past a top-level window.	New features	9.0.015.00	*Predictive Routing Help* • Navigating the Predictive Routing Interface
You can now upload Dataset, Agent Profile, and Customer Profile data to Genesys Predictive Routing (GPR) from CSV files that use certain legacy encodings (listed below). By default, GPR always assumes the CSV files are encoded with UTF-8. This change applies to uploads using both the GPR web application and the GPR API. The following encodings are supported:	New feature	9.0.014.00	*Deployment and Operations Guide* • Supported Encodings

Feature Description	Type of Change	Occurred in Release	Documentation Updates
• UTF-8 • Shift JIS All data returned *from* GPR uses UTF-8 encoding.			
GPR has optimized how cardinalities are stored. Cardinalities are now written into a dedicated database collection, so that the cardinalities for each field are stored in their own document. Previously, the cardinalities were stored along with the schema data. With high-cardinality features, this could lead to performance degradation due to additional conversions needed to extract the cardinality data.	Improvement	9.0.014.00	
The schema management workflow for Agent and Customer Profiles has been simplified and streamlined. The **Discovered Fields** tab has been removed and cardinality counts have been added to the schema view. This change ensures GPR always presents up-to-date Profile information. The schema tab always presented updated information, if available, but the **Discovered Fields** tab display was generated only once and did not reflect changes to the Profile schema.	Improvement	9.0.014.00	*Predictive Routing Help* • Configuring Agent Profiles • Configuring Customer Profiles • Importing and Managing Datasets
The explanation for how to configure and interpret the Agent Variance Report has been clarified and expanded.	Improvement	9.0.014.00	*Predictive Routing Help* • Agent Variance Report
AICS now supports deployment in an environment running on a Kubernetes cluster.	New feature	9.0.014.00	*Deployment and Operations Guide* • (Optional) Installing AICS on a Kubernetes Cluster.
AI Core Services supports Security Enhanced Linux (SELinux) on CentOS 7.	New Feature	9.0.014.00	
If the ASC configuration contains non-empty values for the new **filter-by-skills** and/or **filter-by-groups** configuration options, ASC	New Feature	9.0.014.00	At present, these new options are documented only in the Agent State Connector Release Note.

Feature Description	Type of Change	Occurred in Release	Documentation Updates
subscribes to Stat Server for agent statistics only for the agents included in the specified Agent Groups or those satisfying the configured skill expression. If both options are configured, the agents are subscribed for statistics if they either satisfy the skill expression specified in the **filter-by-skills** option or are included in one of the Agent Groups specified in the **filter-by-groups** option. This functionality enables you to limit the number of agents monitored by GPR or to use GPR in environments where multiple Stat Servers are deployed to monitor different groups of agents.			
Agent State Connector (ASC) now supports email interactions, as well as voice.	New Feature	9.0.014.00	At present, the procedures for enabling email support are documented only in the Agent State Connector Release Note.
Dataset handling has been made significantly faster by means of the following improvements: • For the initial data upload, this release introduces the MinIO container. Set the new S3_ENDPOINT environment variable to take advantage of this faster processing. • The Dataset import to MongoDB now uses a multithreaded process.	New feature	9.0.013.01	*Deployment and Operations Guide* • High-Level Architecture • Set Values for Environment Variables
The sizing worksheets for Genesys Predictive Routing (GPR) have been entirely reworked and expanded.	Improvement	9.0.013.01	*Deployment and Operations Guide* • *Sizing Guide*
The explanation for how to use Composite Predictors has been revised and clarified.	Improvement	9.0.013.01	*Predictive Routing Help* • Composite Predictors
The GPR web application and GPR API now use the same process to create Agent and Customer Profile schemas. In addition, the instructions for creating the Agent Profile schema have been revised and expanded.	Improvement	9.0.013.01	*Predictive Routing Help* • Configuring Agent Profiles
The LOG_LEVEL environment	Improvement	9.0.013.01	*Deployment and Operations Guide*

Feature Description	Type of Change	Occurred in Release	Documentation Updates
variable has been added to the **tango.env** configuration file. By default, it is set to INFO, which is a minimal logging level, adequate for most circumstances.			• Set Values for Environment Variables
This release upgrades AICS to Python 3.6 from Python 2.7.	Improvement	9.0.013.01	*AI Core Services Release Note* - Upgrade Notes • Upgrade Notes
AICS now performs automatic clean up processes which should maintain an adequate amount of free disk space.	New feature	9.0.013.01	*Deployment and Operations Guide* • Clean Up Disk Space
Memory handling for MongoDB was improved In this release.	Improvement	9.0.013.01	*Deployment and Operations Guide* • Scaling AICS
The Lift Estimation report has been improved, adding Export functionality, the ability to toggle between graph and table displays, and showing the Aggregated view as the first tab listed.	Improvement	9.0.013.01	*Predictive Routing Help* • Lift Estimation report
This release includes a number of changes to the look and feel of the GPR web application interface, as well as multiple user experience improvements to provide more intuitive workflows and better presentation of information. For a complete list, refer to the *AICS Release Note*.	Improvement	9.0.013.01	*AI Core Services Release Note* • AI Core Services Release Note for release 9.0.013.01
This release provides a number of improvements and additions to the GPR API, including the ability to check job status and support for nesting dictionary fields. For a complete list, refer to the *AICS Release Note* and the *GPR API Reference*	Improvement	9.0.013.01	*AI Core Services Release Note* and *Genesys Predictive Routing API Reference* • AI Core Services Release Note for release 9.0.013.01 • *Predictive Routing API Reference* (Requires a password for access. Please contact your Genesys representative if you need to view this document.)
The **Quality** column in the Models list table on the Model configuration window now includes a new metric, *Local models.* The metric displays the number of local	Improvement	9.0.012.01	*Predictive Routing Help* • Configuring, Training, and Testing Models

Feature Description	Type of Change	Occurred in Release	Documentation Updates
models generated for agents in the dataset on which the predictor is built.			
AI Core Services (AICS) has improved handling of UTF-8 characters. Data ingestion, model training, and analysis reports are all correctly processed for data containing non-ASCII UTF-8 characters.	Improvement	9.0.012.01	
The GPR API now enables you to run Feature Analysis reports. The API returns a JSON response containing a list of features ordered by weight--that is, by the strength of the impact that feature has on the value of the target metric. The resulting report is also automatically available for view from the GPR web application.	New feature	9.0.012.01	*Genesys Predictive Routing API Reference* • *Predictive Routing API Reference* (access requires a password; contact your Genesys representative for assistance)
The GPR API now enables you to run Lift Estimation reports. The API returns a JSON response containing the Lift Estimation results. The resulting report is also automatically available for view from the GPR web application.	New feature	9.0.012.01	*Genesys Predictive Routing API Reference* • *Predictive Routing API Reference* (access requires a password; contact your Genesys representative for assistance)
The Lift Estimation report now uses the scoring expression configured for the predictor (if any) to decide whether the target metric should be minimized or maximized.	Improvement	9.0.012.01	*Predictive Routing Help* • Understanding Score Expressions
You can now configure Agent State Connector (ASC) to monitor the StatAgentOccupancy Stat Server statistic.	New feature	9.0.012.01	*Deployment and Operations Guide* • (Optional) Configure ASC to Monitor Statistics
You can now configure ASC to monitor a subset of the total list of agent groups present in agent profiles.	New feature	9.0.012.01	*Deployment and Operations Guide* • **include-groups**
You can now choose to have ASC ignore the following unsupported ASCII characters: [Space], -, <, >.	New feature	9.0.012.01	*Deployment and Operations Guide* • **ignore-ascii-characters**
You can now configure ASC to monitor a subset of the total list of skills present in agent profiles.	New feature	9.0.012.01	*Deployment and Operations Guide* • **include-skills**

Feature Description	Type of Change	Occurred in Release	Documentation Updates
ASC now supports a connection to Stat Server running in single-server mode, without a backup.	Improvement	9.0.012.01	
You can configure the Predictive Routing application to display custom messages on the login screen.	New feature	9.0.011.00	*Deployment and Operations Guide* • Set Values for Environment Variables
New supported platforms: • Mongo DB 3.6 (requires a special upgrade procedure; see the *AI Core Services Release Note* for details) • Oracle Linux 7.3	New feature	9.0.011.00	*AI Core Services Release Note, Upgrade Notes* • Upgrade Notes
You can now configure parameters to control password-related behavior such as how often users must change them, blocking users after a specified number of login attempts, and adding a custom message when users are blocked.	New feature	9.0.011.00	*Predictive Routing Help* • Password policy configuration
The audit trail functionality has been improved to record additional actions and provide the ability to specify how long audit trail records are kept.	Improvement	9.0.011.00	*Predictive Routing Help* • Audit Trails
This release includes a number of updates to the functionality offered through the Predictive Routing API: • You can now generate and purge predictor data. • You can now create a new predictor by copying an existing one. To do so, send a POST request to the new **copy_predictor** endpoint. • You can now use GET commands to retrieve dataset and predictor details. • The way Predictive Routing recomputes cardinalities when you append data to Agent or Customer Profiles using the API has been changed: • Cardinalities are no longer	New feature	9.0.011.00	*Genesys Predictive Routing API Reference* • *Predictive Routing API Reference* (access requires a password; contact your Genesys representative for assistance)

Feature Description	Type of Change	Occurred in Release	Documentation Updates
recomputed automatically across the whole collection each time you append data. Full automatic computation happens only once, when an Agent or Customer Profile is uploaded the first time for schema discovery. When you append data to an Agent or Customer Profile via the API, cardinalities are computed only for the appended data portion and only when the number of agents or customers set in the **ADD_CARDINALITIES_EVERY_N_RECORDS** parameter is reached. The results of computation are added to the already-stored cardinality values. This new behavior significantly improves speed when loading new data by avoiding simultaneous recomputations on the full data collection when there are multiple frequent appends done in small batches. • The **ADD_CARDINALITIES_EVERY_N_RECORDS** parameter has been added to the **tango.env** file with the default value of `1000`. Each time the counter for appended agents/ customers reaches this number, computation for the last appended 1000 records takes place. The default value can be changed in the **tango.env** file, which is located in the **IP_<*version*>/conf** directory. When you change the value, restart the application to have the new value take effect. • You can force recomputation of cardinalities on the full Agent or Customer Profiles			

Feature Description	Type of Change	Occurred in Release	Documentation Updates
collection using the new POST **compute_cardinalities** API endpoint. • You can now retrieve information on the currently deployed platform using the new **version** endpoint.			
Predictive Routing now correctly recognizes columns with any combination of the following Boolean values: `y/n`, `Y/N`, `Yes/No`. Previously, only columns with `true/false` and `0/1` values were discovered as Booleans. The identification is case insensitive.	Improvement	9.0.011.00	
You can now upload data (agent, customer, and dataset) using zip-archived .csv files. Only one .csv file per archive is supported.	Improvement	9.0.011.00	
GPR complies with GDPR requirements for handling sensitive customer information.	New feature	9.0.010.01	*Genesys Security Deployment Guide* • *Genesys Security Deployment Guide* *Deployment and Operations Guide* • Handling Personally Identifiable Information in Compliance with General Data Protection Regulation (EU) *Genesys Predictive Routing API Reference* • *Predictive Routing API Reference* (access requires a password; contact your Genesys representative for assistance)
The Lift Estimation report now offers **Advanced Group By** functionality, which provides more flexibility in customizing the report.	Improvement	9.0.010.01	*Predictive Routing Help* • Lift Estimation Report Overview
GPR now supports LDAP authentication when users log in.	New feature	9.0.009.00	*Predictive Routing Help* • Configuring LDAP-Enabled Accounts

Feature Description	Type of Change	Occurred in Release	Documentation Updates
The following improvements have been made to the Lift Estimation report: • When generating the Lift Estimation report, GPR now provides the option to produce a report for each unique value for a selected column (feature). Previously, any feature with a cardinality of more than 20 was excluded, which meant that you could not produce reports with a granularity higher than 20 unique features. • The agent pool for lift estimation is now constructed on a per-day basis for the interactions in the dataset. Previously, you might have observed a negative lift for higher agent availability or an unexpectedly high lift for low agent availability due to overcorrection caused by a mismatch between the input sample size and the actual sizes encountered through daily simulation.	New feature/ Enhancement	9.0.009.00	*Predictive Routing Help* • Lift Estimation report
Journey Optimization Platform (JOP) was renamed to AI Core Services (AICS).	Update	9.0.009.00	
You can now enable GPR to look up updated values for certain agent attributes, based on customer or interaction attributes during a scoring request. For instance, you can look up agent performance by virtual queue, enabling you to evaluate the agent's previous performance when handling interactions from that queue. This avoids comparing agent performance for a specific queue against other agents who handle interactions from a different mixture of virtual queues.	New feature/ Enhancement	9.0.008.00	*Predictive Routing Help* • Gather Updated Scoring Data Using Profile Look Ups
You can now view an entire Agent Profile or Customer Profile record from the **Agents Details** and **Customers Details** tabs or an entire record on the **Datasets**	New feature/ Enhancement	9.0.008.00	*Predictive Routing Help* • Analyzing Agents

Feature Description	Type of Change	Occurred in Release	Documentation Updates
Details tab. Click a single record to open a new window containing a table with all the related key-value pairs.			• Analyzing Customers • Analyzing Dataset Trends and Details
Agent State Connector now supports connection to a secured Configuration Server port and TLS 1.2 connections to Stat Server.	New feature/ Enhancement	9.0.008.00	*Deployment and Operations Guide* • Secure Connections
You can now configure Agent State Connector to automatically create an Agent Profile schema, if none exists, or to verify the existing schema.	New feature/ Enhancement	9.0.008.00	*Deployment and Operations Guide* • Create the Agent Profile Schema
You can now have Agent State Connector collect call **connId** data from Stat Server and write it to the Agent Profile schema for use in Predictive Routing.	New feature/ Enhancement	9.0.008.00	*Deployment and Operations Guide* • Enable Collection and Storage of Call ConnIds
Agent State Connector is now supported on Windows and Linux 64-bit systems.	New feature/ Enhancement	9.0.008.00	For the exact Windows and Linux versions supported, see the *Genesys Supported Operating Environment Reference Guide*. • For Windows and Linux 64-bit deployment instructions, see Installing on Windows and Installing on RedHat Linux 7 64-Bit in the *Deployment and Operations Guide*.
Predictive Routing now supports datasets of up to 250 columns for predictor data generation, model training, and analysis.	New feature/ Enhancement	9.0.008.00	*Deployment and Operations Guide* • Data Size Guidelines - Data Import, Model Training, and Feature Analysis
Model training speed has been considerably improved.	New feature/ Enhancement	9.0.008.00	
Predictive Routing now provides progress indicators when loading predictor data and generating predictors. The progress indicators show the percent complete and the number of data rows already loaded.	New feature/ Enhancement	9.0.008.00	
The maximum supported cardinality for the **Group By** functionality in the **Lift Estimation** report has been increased to 20. All features with	New feature/ Enhancement	9.0.008.00	*Predictive Routing Help* • Lift Estimation Report Overview

Feature Description	Type of Change	Occurred in Release	Documentation Updates
cardinalities between 1 and 20 are now available in the **Group By** selection menu.			
You can now enter a maximum value of 500 simulations in the **Lift Estimation** analysis report settings. This prevents you from entering numbers too large to efficiently analyze and which can lead to an out-of-memory situation. The **Number of Simulations** field accepts any value larger than 0 and less than or equal to 500.	New feature/ Enhancement	9.0.008.00	*Predictive Routing Help* • Lift Estimation Report Overview
Predictive Routing now provides a text search field for use when selecting attributes for analysis.	New feature/ Enhancement	9.0.008.00	*Predictive Routing Help* • Creating and Interpreting Analysis Reports
The documentation and the user interface have been updated to reflect the new product name (changed from Genesys Predictive Matching to Genesys Predictive Routing).	Update	9.0.007.01	
The product name has changed from Genesys Predictive Matching to Genesys Predictive Routing.	New Feature	9.0.007.00	The documentation and the user interface are scheduled to be updated as a post-release activity.
Support for both single-site and multi-site high availability architectures.	New Feature	AICS and ASC 9.0.007.00	*Deployment and Operations Guide:* • Deploying: High Availability
Support for historical reporting, provided by the Genesys Reporting solution. The following reports are available in Genesys Interactive Insights: Predictive Routing AB Testing Report, Predictive Routing Agent Occupancy Report, Predictive Routing Detail Report, Predictive Routing Operational Report, and Predictive Routing Queue Statistics Report. Important This functionality requires Genesys Info Mart 8.5.009.12 or higher, Reporting and Analytics Aggregates 8.5.002 or higher, and Genesys Interactive Insights 8.5.001 or higher.	New Feature	AICS and Strategy Subroutines 9.0.007.00	*Deployment and Operations Guide:* • Deploying: Integrating with Genesys Reporting • **send-user-event** (new) • **vq-for-reporting** (new)
Real-time reporting templates are available for use in Pulse dashboards: Agent Group KPIs by	New Feature	AICS and Strategy Subroutines	*Deployment and Operations Guide:*

Feature Description	Type of Change	Occurred in Release	Documentation Updates
Predictive Model and Queue KPIs by Predictive Model.		9.0.007.00	• Deploying: Integrating with Genesys Reporting
Two new analysis reports have been added to the Genesys Predictive Routing application: Agent Variance and Lift Estimation.	New Feature	AICS 9.0.007.00	*Help File:* • Agent Variance Report • Lift Estimation Report
The Model creation interface now includes additional model quality and agent coverage reporting.	New Feature	AICS 9.0.007.00	*Help File:* • Receiver Operating Characteristic (ROC) Curve
The Feature Analysis report, the model creation and training functionality, and the dataset import functionality have been improved to handle large datasets.	Improvement	AICS 9.0.007.00	*Deployment and Operations Guide:* • Data Size Guidelines - Data Import, Model Training, and Feature Analysis
You can now combine simple predictors to create composite predictors.	New Feature	AICS 9.0.007.00	*Help File:* • About Composite Predictors
Agent State Connector now enables you to set alarms if there are persistent connection issues with Configuration Server or Stat Server.	New Feature	ASC 9.0.007.00	*Deployment and Operations Guide:* • Monitoring Agent State Connector
Improved logging for the AICS Tango container when you train a model. The relevant log message now includes the model ID and feature size.	New Feature	AICS 9.0.007.00	*Deployment and Operations Guide:* • Model Training Logs
The agent occupancy control feature was improved.	Improvement	Strategy Subroutines 9.0.007.00	*Deployment and Operations Guide:* • **agent-occupancy-factor** (new) • **use-agent-occupancy** (updated) • **agent-occupancy-threshold** (updated)
The behavior of the time-sliced A/B testing mode (the **prr-mode** option is set to `ab-test-time-sliced`) has been improved.	Improvement	Strategy Subroutines 9.0.007.00	*Deployment and Operations Guide:* • **ab-test-time-slice** (new default value)
You can now set a timeout value that enables Genesys Predictive	New Feature	Strategy Subroutines	*Deployment and Operations Guide:*

Feature Description	Type of Change	Occurred in Release	Documentation Updates
Routing to tell whether URS is overloaded, at which point Predictive Routing turns itself off.		9.0.007.00	• **overload-control-timeout** (new)
When you are training a model, new status indicators immediately inform you of the progress of model training, from "IN QUEUE", to a blinking "IN TRAINING" when the training job starts being processed, to "TRAINED" after job has been completed.	Improvement	AICS 9.0.007.00	
Predictive Routing now enforces conversion of non-string ID values into strings in the Agent and Customer Profile schemas.	Improvement	AICS 9.0.007.00	
The Predictive Routing strategy integration with URS now automatically deletes interaction scoring data stored in the URS global map once the interaction is routed or abandoned. As a result, the PrrIxnCleanup subroutine is no longer needed. This change requires URS version 8.1.400.37 or higher.	Improvement	Strategy Subroutines 9.0.007.00	
Apache Kafka is no longer used for triggering the execution of model training or analysis jobs. This functionality has been taken over by MongoDB. As a result, the kafka container is no longer part of the AICS installation package.	Discontinued container	AICS 9.0.007.00	*Deployment and Operations Guide:* • Deploying AICS on a Single Host
In the Predictive Routing interface, the Action Features label is now *Agent Features*; Context Features is now *Customer Features*; Action Type is now *Agent Identifier*; Context Type is now *Customer Identifier*.	Improvement	AICS 9.0.007.00	*Help File:* • Settings: Creating and Updating Predictors
Added support for updating, deleting, and reading the indexes on Agent and Customer Profile collections through the Predictive Routing API.	Improvement	AICS 9.0.007.00	*API Reference:* • *Predictive Routing API Reference*
Models now show how many versions have been created. In addition, you can now copy activated models.	Improvement	AICS 9.0.007.00	*Help File:* • Settings: Configuring, Training, and Testing Models
The Composer	Improvement	Strategy	*Deployment and Operations Guide:*

Feature Description	Type of Change	Occurred in Release	Documentation Updates
ActivatePredictiveMatching subroutine now supports two types of responses to score requests, either containing both **list** and **list_ranks** fields or just the **list** field. These are controlled by the request parameter **format_as_map**.		Subroutines 9.0.007.00	• **format-as-map** (new)
The following two configuration options are no longer supported as of this release. The functionality they enabled has been adjusted to make them unnecessary.	Discontinued Support	ASC 9.0.006.07	*Deployment and Operations Guide:* • **reset-jop-on-startup** - This option is no longer required, and is now permanently set to `false`. To delete agents, delete your current agent profile schema in the Predictive Routing application, and then upload an updated schema. • **stat-srv-ws-conn** - This option is no longer required, and is now permanently set to `true`. ASC now supports warm standby connections by default.
You can now deploy the AI Core Services (AICS) in Docker containers.	New Feature	AICS 9.0.006.05	*Deployment and Operations Guide:* • Deploying: AI Core Services
This release includes updates to the Predictive Routing web interface for improved usability.	Improvement	AICS 9.0.006.05	*Help File:* • Genesys Predictive Routing Help
This release includes context-sensitive Help. To open the Help from the interface, click the **?** icon on the top menu bar.	Improvement	AICS 9.0.006.05	*Help File:* • Genesys Predictive Routing Help
In ASC, you can now specify a timeout between each subscription to avoid overloading Stat Server.	Improvement	ASC 9.0.006.06	*Deployment and Operations Guide:* • A new option, **ss-subscription-timeout**, enables this functionality.
Agent State Connector and Strategy Subroutines components can be deployed in a high availability configuration.	New Feature	ASC 9.0.006.06/ URS Strategy Subroutines 9.0.006.00/ Composer Strategy	*Deployment and Operations Guide:* • Deploying: High Availability

Feature Description	Type of Change	Occurred in Release	Documentation Updates
		Subroutines 9.0.006.01	
Routing using Predictive Routing can now take agent occupancy into account when selecting the best target.	Improvement	URS Strategy Subroutines 9.0.006.00/ Composer Strategy Subroutines 9.0.006.01	*Deployment and Operations Guide:* • The new **use-agent-occupancy** and **agent-occupancy-threshold** options control this functionality.
You can customize how Predictive Routing handles the authentication tokens that enable the URS Strategy Subroutines to request agent scores from AICS.	New Feature	URS Strategy Subroutines 9.0.006.00/ Composer Strategy Subroutines 9.0.006.01	*Deployment and Operations Guide:* • Two new configuration options, **scoring-token-expiration** and **emergency-scoring-token**, enable this functionality.
Predictive Routing IRD strategy subroutines now utilize the URS TimeBehind[] function to detect when URS is overloaded and adjust their behavior accordingly.	New Feature	URS Strategy Subroutines 9.0.006.00/ Composer Strategy Subroutines 9.0.006.01	*Deployment and Operations Guide:* • Troubleshooting for a URS-based Predictive Routing Environment
The workflow for creating Predictors has been made more logical and straightforward.	Improvement	AICS 9.0.006.05	*Help File:* • Settings: Creating Predictors
Users can now reset their passwords from the Predictive Routing web interface.	Improvement	AICS 9.0.006.05	*Help File:* • User Profile: Managing Passwords

Code Samples

The code samples included in this topic are for use with the Predictive Routing API.

cURL Examples

The *Predictive Routing API Reference* includes cURL examples. The *Reference* requires a password for access. Please contact your Genesys representative if you need to view this document.

> **Important**
>
> If you copy and past the cURL example code into a Windows-based code editor, you must remove the \ characters at the ends of the lines.

Python Examples

The following examples provide a guideline for how to implement GPR API calls using Python. The examples are presented in the same groupings as in the *Predictive Routing API Reference*.

Authenticate Example

```
#POST / authenticate
    response = requests.post(
        '%s/api/v2.0/authenticate' % HOST,
        data={
            'username': USERNAME,
            'password': PASSWORD,
            'api_key': API_KEY
        }
    )
    token =  response.json()['token']
    print token
```

Datasets Examples

```
# GET /datasets (get presigned URL) Get Dataset presigned URL for uploading file to S3
def get_presigned_url(token):
    response = requests.get(
        '%s/api/v2.0/datasets' % HOST,
        json={
            'token': token,
            'command': 'presigned_url',
            'filename': 'dataset.csv.zip'
        }
```

```
    )
    return response.json()['signed_url']

# PUT /presigned URL (upload to S3) Upload dataset to S3
def upload_dataset_s3(presigned_url):
    response = requests.put(presigned_url,
        data = open(DATASET_PATH, 'rb').read()
    )
    return response.json()

# POST /datasets (create using S3) Create a dataset using S3
def create_dataset_s3(token):
    response = requests.post(
        '%s/api/v2.0/datasets' % HOST,
        json={
            'token': token,
            'name': 'DatasetName',
            's3_separator': 'TAB',
            's3_filename': 'dataset.csv.zip',
            'encoding': 'shift-jis'
        }
    )
    return response.json()

# PUT /datasets/{id}/ (using s3) Append a dataset using a file uploaded to S3.
def append_dataset_s3(token):
    response = requests.put(
        '%s/api/v2.0/datasets/%s/append' % (HOST, DATASET_ID),
        json={
            'token': token,
            's3_separator': 'TAB',
            's3_filename': 'dataset.csv.zip',
            'encoding': 'shift-jis'
        }
    )
    return response.json()

# POST /datasets (create using file system)
def create_from_csv(token):
    response = requests.post(
        '%s/api/v2.0/datasets' % HOST,
        data={
            'token': token,
            'sep': 'TAB',
            'name': 'TestDataset ' + str(datetime.datetime.now())
        },
        files={
            'csv_file': open(SMALL_FILE, 'rb')
        }
    )
    return response.json()

# GET /datasets Get all datasets details
def get_all_ds_details(token):
    response = requests.get(
        '%s/api/v2.0/datasets' % HOST,
        json={
            'token': token,
            'fields2show': ['name', 'id', 'sync_status']
        }
    )
    return response.json()
```

```
# GET /datasets/{id} Get Dataset details.
def get_dataset_details(token):
    response = requests.get(
        '%s/api/v2.0/datasets/%s' % (HOST, DATASET_ID),
        json={
            'token': token,
            'include_cardinalities': 'false'
        }
    )
    return response.json()

# PUT /datasets/{id} Edit dataset.
def edit_dataset(token):
    response = requests.put(
        '%s/api/v2.0/datasets/%s' % (HOST, DATASET_ID),
        data={
            'token': token,
            'name': 'NR TestDataset 2019-07-02 16:33:12.116667',
        }
    )
    return response.json()

# DELETE /datasets/{id} Delete dataset.
def delete_dataset(token):
    response = requests.delete(
        '%s/api/v2.0/datasets/%s' % (HOST, DATASET_ID),
        data={
            'token': token,
        }
    )
    return response.json()

# GET /datasets/{id}/data Get Dataset details using a filter or search.
def filter_ds(token):
    response = requests.get(
        '%s/api/v2.0/datasets/%s/data' % (HOST, DATASET_ID),
        json={
            'token': token,
            'skip': 0,
            'limit': 100,
            "filter": "Agent_Age>30&Agent_Skill2=Billing",
            "search": "Buyer"
        }
    )
    return response.json()

# PUT /datasets/{id}/append (using file system) Append a dataset using a CSV file or a zipped
CSV file.

def append_data_csv(token):
    response = requests.put(
        '%s/api/v2.0/datasets/%s/append' % (HOST, DATASET_ID),
        data={
            'token': token,
            'sep': 'TAB',
        },
        files={
            'csv_file': open(SMALL_FILE, 'rb')
        }
    )
    return response.json()

# PUT /datasets/{id}/append (using JSON) Append a dataset using JSON batch data.
```

```
def append_data_batch(token, dataset_id):
    response = requests.put(
        '%s/api/v2.0/datasets/%s/append' % (HOST, dataset_id),
        json={
            'token': token,
            'batch_data': [
                {
                    "a": "1",
                    "b": "abc",
                    "ts": "12345"
                }]
        }
    )
    return response.json()

# PUT /datasets/{id}/apply_sync Sync dataset.
def sync_dataset(token):
    response = requests.put(
        '%s/api/v2.0/datasets/%s' % (HOST, DATASET_ID),
        json={
            'token': token,
            'command': 'apply_sync'
        }
    )
    return response.json()

# PUT /datasets/{id}/accept_sync Accept dataset.
def accept_sync(token):
    response = requests.put(
        '%s/api/v2.0/datasets/%s' % (HOST, DATASET_ID),
        json={
            'token': token,
            'command': 'accept_sync'
        }
    )
    return response.json()

# PUT /datasets/{id}/cancel_sync Cancel dataset synchronization.
def cancel_sync(token):
    response = requests.put(
        '%s/api/v2.0/datasets/%s' % (HOST, DATASET_ID),
        json={
            'token': token,
            'command': 'cancel_sync'
        }
    )
    return response.json()
```

Schemas Examples

```
# GET /schemas         Get presigned URL
def get_presigned_url(token):
    response = requests.get(
        '%s/api/v2.0/schemas' % HOST
        json={
            'token': token,
            'command': 'presigned_url',
            'schema_type': 'customers',                # or 'agents'
            'filename': 'customers.csv'
        },
    )
    tuplex = (response.json()['signed_url'], response.json()['file_path'])
```

```
    return tuplex

# PUT /presigned_url        Upload source file to s3(minio)

def upload_profile_s3(presigned_url):
    response = requests.put(
        presigned_url,
        data=open(PROFILE_PATH, 'rb').read()
    )
    return response

# POST /schemas         Upload from s3 to the GPR platform
def upload_from_s3(token, file_path):
    response = requests.post(
      '%s/api/v2.0/schemas' % HOST,
      data = {
              'token': token,
              'schema_type': 'customers',
              'command': 'finish_s3_upload',
              's3_separator': 'TAB',
              's3_filename': file_path
             }
    )
    return response.json()

def upload_from_s3(token, file_path):
    response = requests.post(
      '%s/api/v2.0/schemas' % HOST,
      data = {
              'token': token,
              'schema_type': 'customers',
              'command': 'finish_s3_upload',
              's3_separator': 'TAB',
              's3_filename': file_path
             }
    )
    return response.json()

# PUT /schemas (create schema from a sample
def create_schema_from_template(token):
    response = requests.put(
        '%s/api/v2.0/schemas' % (HOST),
        json={
            'token': token,
            'schema_type': 'agents',
            'sample_data': {
                'string_field': 'string',
                'integer_field': 1,
                'boolean_field': False
            }

        }
    )
    return response.json()

# PUT /schemas (create final schema)
def create_final_schema(token):
    response = requests.put(
        '%s/api/v2.0/schemas' % (HOST),
        json={
            'token': token,
            'schema_type': 'agents',
            'schema_info': [
```

```
                {
                    'name': 'boolean_field',
                    'type': 'boolean'
                },
                {
                    'name': 'string_field',
                    'type': 'string',
                    'is_id': True
                },
                {
                    'name': 'integer_field',
                    'type': 'integer'
                }
            ]
        }
    )
    return response.json()

# POST /schemas Sync, Cancel sync, Accept schema
def manage_schemas(token):
    response = requests.post(
        '%s/api/v2.0/schemas' % HOST,
        data={
            'token': token,
            'command': 'accept_sync',
            'schema_type': 'agents'
        }
    )
    return response.json()

# DELETE /schemas Delete schema
def delete_schema(token):
    response = requests.delete(
        '%s/api/v2.0/schemas' % (HOST),
        data={
            'token': token,
            'schema_type': 'agents'
        }
    )
    return response.json()

# GET /schemas Get schema details
def get_schema_details(token):
    response = requests.get(
        '%s/api/v2.0/schemas' % HOST,
        json={
            'token': token,
            'schema_type': 'agents'
        }
    )
    return response.json()

# GET /schemas Get schema details
def get_schema_details(token):
    response = requests.get(
        '%s/api/v2.0/schemas' % HOST,
        json={
            'token': token,
            'schema_type': 'agents'
        }
    )
    return response.json()
```

```
# POST /schemas (change fields visibility)
def change_schema_visibility(token):
    response = requests.post(
        '%s/api/v2.0/schemas' % HOST,
        json={
            'token': token,
            'command': 'fields_visibility',
            'schema_type': 'agents',
            'visible_fields': ['Agent_First']

        }
    )
    return response.json()
```

Agents Examples

```
# POST /agents (create using file)
def upload_agents_csv(token):
    response = requests.post(
        '%s/api/v2.0/agents' % HOST,
        data={
            'token': token,
            'sep': 'COMMA',
            'encoding': 'shift-jis'
        },
        files={
            'csv_file': open(AGENTS_FILE, 'rb'),
        }
    )
    return response.json()

# POST /agents (create using json)
def upload_agents_batch(token):
    response = requests.put(
        '%s/api/v2.0/agents' % (HOST),
        json={
            'token': token,
            'batch_data': [
                {
                    'Agent_AgentID': '1000',
                    'Agent_First': 'Jane',
                    'Agent_Last': 'Doe'
                }],
            'return_objects': True,
            'include_complex_attributes': True
        }
    )
    return response.json()

# GET /agents Get Agent details
def get_agent_details_id(token):
    response = requests.get(
        '%s/api/v2.0/agents' % HOST,
        json={
            'token': token,
            'Agent_AgentID': '1000',
            'include_complex_attributes': True
        }
    )
    return response.json()

def get_agent_details_search(token):
```

```
    response = requests.get(
        '%s/api/v2.0/agents' % HOST,
        json={
            'token': token,
            'filter': 'Agent_Experience>10&Agent_Location_Country=US',
            'search': 'John',
            'include_complex_attributes': True
        }
    )
    return response.json()

#PUT /agents Edit agent record
def edit_agent(token):
    response = requests.put(
        '%s/api/v2.0/agents' % (HOST),
        json={
            'token': token,
            'batch_data': [
                {
                    'Agent_AgentID': '1000',
                    'Agent_First': 'Joe',
                    'Agent_Last': 'Doe'
                }]
        }
    )
    return response.json()

#DELETE /agents Delete agent record.
def delete_agent(token):
    response = requests.delete(
        '%s/api/v2.0/agents' % (HOST),
        data={
            'token': token,
            'Agent_AgentID': '1000'
        }
    )
    return response.json()

# POST /agents/compute_cardinalities
def agent_cardinalities(token):
    response = requests.post(
        '%s/api/v2.0/agents/compute_cardinalities' % HOST,
        data={
            'token': token
        }
    )
    return response.json()
```

Customers Examples

```
#POST /customers (create using file)
def upload_customers_csv(token):
    response = requests.post(
        '%s/api/v2.0/customers' % HOST,
        data={
            'token': token,
            'sep': 'TAB',
            'encoding': 'shift-jis'
        },
        files={
            'csv_file': open(SMALL_FILE, 'rb'),
        }
```

```
    )
    return response.json()

#POST /customers (create using JSON)
def upload_customers_batch(token):
    response = requests.put(
        '%s/api/v2.0/customers' % (HOST),
        json={
            'token': token,
            'batch_data': [
                {
                    'Customer_CustomerID': 'eafa71e9-85b9-4fa1-9395-2f6f3d7f1d65',
                    'Customer_AccountValue': 759696320,
                    'Customer_Age': 37,
                    'Customer_AgeBucket': 1,
                    'Customer_Location_Country': 'Canada',
                    'Customer_Location_State': 'BC',
                    'Customer_Intent': 'Opening Account',
                    'Customer_Income': 798884,
                    'Customer_Escalations': 3
                }],
            'return_objects': True,
            'include_complex_attributes': True
        }
    )
    return response.json()

# GET /customers Get Customer details
def get_customer_details_id(token):
    response = requests.get(
        '%s/api/v2.0/customers' % HOST,
        json={
            'token': token,
            'Customer_CustomerID': 'eafa71e9-85b9-4fa1-9395-2f6f3d7f1d65',
            'include_complex_attributes': True
        }
    )
    return response.json()

def get_customer_details_search(token):
    response = requests.get(
        '%s/api/v2.0/customers' % HOST,
        json={
            'token': token,
            'filter': 'Customer_Age>20&Customer_Location_Country=US',
            'search': 'male',
            'include_complex_attributes': True
        }
    )
    return response.json()

#PUT /customers Edit Customer record
def edit_customer(token):
    response = requests.put(
        '%s/api/v2.0/customers' % (HOST),
        json={
            'token': token,
            'batch_data': [
                {
                    'Customer_CustomerID': 'eafa71e9-85b9-4fa1-9395-2f6f3d7f1d65',
                    'Customer_Age': '35'
                }]
        }
```

```
    )
    return response.json()

#DELETE /customers Delete customer record.
def delete_customer(token):
    response = requests.delete(
        '%s/api/v2.0/customers' % (HOST),
        data={
            'token': token,
            'Customer_CustomerID': 'eafa71e9-85b9-4fa1-9395-2f6f3d7f1d65'
        }
    )
    return response.json()

# POST /customers/compute_cardinalities
def customer_cardinalities(token):
    response = requests.post(
        '%s/api/v2.0/customers/compute_cardinalities' % HOST,
        data={
            'token': token
        }
    )
    return response.json()
```

Predictors Examples

```
# POST /predictors (simple)
def create_predictor(token):
    response = requests.post(
        '%s/api/v2.0/predictors' % HOST,
        json={
            "token": token,
            "name": "TestPredictor2",
            "from_dt": int(time.mktime((2018, 1, 31, 0, 0, 0, 0, 0, 0))),  # Dummy range to
fit my sample set
            "to_dt": int(time.mktime((2020, 1, 31, 0, 0, 0, 0, 0, 0))),
            "action_type": "agents",
            "context_type": "customers",
            "dataset": DATASET_ID,
            "metric": "NPS",
            "action_id_expression": "Agent_AgentID",
            "kpi_type": "Sales",
            "action_features_schema": [
                {
                    "label": "Agent_Skill4",
                    "type": "string",
                    "field_expr": "Agent_Skill4"
                },
                {
                    "label": "Agent_SalesConversionRate",
                    "type": "integer",
                    "field_expr": "Agent_SalesConversionRate"
                },
                {
                    "label": "Agent_Skill4_Level",
                    "type": "integer",
                    "field_expr": "Agent_Skill4_Level"
                }
            ],
            "context_features_schema": [
                {
                    "label": "Customer_Age",
```

```
                    "type": "integer",
                    "field_expr": "Customer_Age"
                },
                {
                    "label": "Customer_Location_Country",
                    "type": "string",
                    "field_expr": "Customer_Location_Country"
                },
                {
                    "label": "Customer_Intent",
                    "type": "string",
                    "field_expr": "Customer_Intent"
                }
            ],
        }
    )
    return response.json()

# POST /predictors (composite)
def create_composite_predictor(token):
    response = requests.post(
        '%s/api/v2.0/predictors' % HOST,
        json={
            "token": token,
            "name": "composite_predictor",
            "predictor_type": "Composite Predictor",
            "predictors_list": [PREDICTOR_ID, PREDICTOR_ID2],
            "raw_expression": "($predictor1Name1 + $predictor2Name)/2"
        }
    )
    return response.json()

# GET /predictors Get all predictors details
def get_predictors_info(token):
    response = requests.get(
        '%s/api/v2.0/predictors' % (HOST),
        json={
            'token': token,
            'fields2show': ['name', 'id']
        }
    )
    return response.json()

# GET /predictors/{id} Get predictor details
def get_predictor_info(token):
    response = requests.get(
        '%s/api/v2.0/predictors/%s' % (HOST, PREDICTOR_ID),
        json={
            'token': token,
        }
    )
    return response.json()

# PUT /predictors/{id} Edit predictor record
def edit_predictor(token):
    response = requests.put(
        '%s/api/v2.0/predictors/%s' % (HOST, PREDICTOR_ID),
        json={
            'token': token,
            'name': 'RenamedTestPredictor'
        }
    )
    return response.json()
```

```
#DELETE /predictors/{id} Delete predictor record.
def delete_predictor(token):
    response = requests.delete(
        '%s/api/v2.0/predictors/%s' % (HOST, PREDICTOR_ID2),
        data={
            'token': token
        }
    )
    return response.json()

# PUT /predictors/{id}/generate_data Generate predictor data.
def generate_training_data(token, from_timestamp, to_timestamp):
    response = requests.put(
        '%s/api/v2.0/predictors/%s' % (HOST, PREDICTOR_ID),
        json={
            'command': 'generate_data',
            'from_dt': from_timestamp,
            'to_dt': to_timestamp,
            'token': token
        }
    )
    return response.json()

# PUT /predictors/{id}/purge_data Purge predictor data
def purge_training_data(token):
    response = requests.put(
        '%s/api/v2.0/predictors/%s' % (HOST, PREDICTOR_ID),
        json={
            'command': 'purge_data',
            'token': token
        }
    )
    return response.json()

# GET /predictors/{id}/check_status Check status of predictor generate and purge jobs
def check_predictor_status(token):
    response = requests.get(
        '%s/api/v2.0/predictors/%s/check_status' % (HOST, PREDICTOR_ID),
        json={
            'token': token
        }
    )
    return response.json()

# GET /predictors/{id}/data Get predictor data
def get_predictor_data(token):
    response = requests.get(
        '%s/api/v2.0/predictors/%s/data' % (HOST, PREDICTOR_ID),
        json={
            'token': token,
            'filter': 'act.Agent_Skill4_Level=1&ctx.Customer_Age>30'
        }
    )
    return response.json()

# POST /predictors/{id}/copy_predictor Copy predictor
def copy_predictor(token):
    response = requests.post(
        '%s/api/v2.0/predictors/%s/copy_predictor' % (HOST, PREDICTOR_ID),
        json={
            'token': token
        }
    )
```

```
    return response.json()

# POST /predictors/{id}/score Score actions for predictor context
def predictor_score(token):
    response = requests.post(
        '%s/api/v2.0/predictors/%s/copy_predictor' % (HOST, PREDICTOR_ID),
        json={
            'token': token,
            'context': {
                'Customer_Location_Country': 'US'
            },
            'action_filters': "Agent_Skill4_Level in ['4','5']",
            'format_as_map': True,
            'warnings': True
        }
    )
    return response.json()

# POST /predictors/{id}/reset Reset predictor
def reset_predictor(token):
    response = requests.post(
        '%s/api/v2.0/predictors/%s/reset' % (HOST, PREDICTOR_ID),
        json={
            'token': token
        }
    )
    return response.json()

# POST /predictors/{id}/feedback Predictor feedback
def predictor_feedback(token):
    response = requests.post(
        '%s/api/v2.0/predictors/%s/feedback' % (HOST, PREDICTOR_ID),
        json={
            'token': token,
            'action': {
                'action_id': 'UUID_2',
                'skill': 'testing2',
                'age': '28',
                'fluency': 'good2',
                'seniority': 'veteran2'
            },
            'context': {
                'age': '36',
                'gender': 'M',
                'location': 'San Francisco2',
                'n_subs': '16',
                'intention': 'closing an account',
                'seniority': 'ancient2'
            },
            'reward': 0
        }
    )
    return response.json()

# POST /ab_testing/join A/B Test predictor
def ab_test(token):
    response = requests.post(
        '%s/api/v2.0/ab_testing/join' % (HOST),
        json={
            'token': token,
            'prrPredictor': 'RenamedTestPredictor',
            'from_date': '01/31/2018',
            'to_date': '01/31/2020'
```

```
        }
    )
    return response.json()
```

Score Log Examples

```
# POST /score_log Log your predictor score log data
def score_log(token):
    response = requests.post(
        '%s/api/v2.0/score_log' % (HOST),
        json={
            'token': token,
            'prrPredictor': 'RenamedTestPredictor',
            'context': {
                'Customer_Location_Country': 'US'
            },
            'action_filters': "Agent_Skill4_Level in ['4','5']",
            'format_as_map': True,
            'warnings': True
        }
    )
    return response.json()

# GET /score_log View your previously logged score data
def get_score_log(token):
    response = requests.get(
        '%s/api/v2.0/score_log' % (HOST),
        json={
            'token': token,
            'prrPredictor': 'RenamedTestPredictor',
            'context': {
                'Customer_Location_Country': 'US'
            }
        }
    )
    return response.json()
```

Models (PredictorsModels) Examples

```
# POST /predictor_models Create predictor model.
def create_predictive_model(token):
    response = requests.post(
        '%s/api/v2.0/predictor_models' % HOST,
        json={
            "token": token,
            "display_name": "TestModel_DISJOINT",
            "predictor_id": PREDICTOR_ID,
            "model_type": "DISJOINT",
            "context_features": [
                {
                    "label": "Customer_Age",
                    "type": "integer",
                    "field_expr": "Customer_Age"
                },
                {
                    "label": "Customer_Location_Country",
                    "type": "string",
                    "field_expr": "Customer_Location_Country"
                },
                {
```

```
                    "label": "Customer_Intent",
                    "type": "string",
                    "field_expr": "Customer_Intent"
                }
            ],
            "action_features": [
                {
                    "label": "Agent_Skill4",
                    "type": "string",
                    "field_expr": "Agent_Skill4"
                },
                {
                    "label": "Agent_SalesConversionRate",
                    "type": "integer",
                    "field_expr": "Agent_SalesConversionRate"
                },
                {
                    "label": "Agent_Skill4_Level",
                    "type": "integer",
                    "field_expr": "Agent_Skill4_Level"
                }
            ],
            "train_data_percentage": 80,
        }
    )
    return response.json()

# GET /predictor_models Get all predictor Models details
def get_models_info(token):
    response = requests.get(
        '%s/api/v2.0/predictor_models' % (HOST),
        json={
            'token': token,
            'predictor_id': PREDICTOR_ID
        }
    )
    return response.json()

# GET /predictor_models/{id} Get predictor Model details
def get_model_info(token):
    response = requests.get(
        '%s/api/v2.0/predictor_models/%s' % (HOST, MODEL_ID),
        json={
            'token': token,
            'predictor_id': PREDICTOR_ID
        }
    )
    return response.json()

# PUT /predictor_models/{id} Edit predictor model record
def edit_model(token):
    response = requests.put(
        '%s/api/v2.0/predictor_models/%s' % (HOST, MODEL_ID),
        json={
            "token": token,
            "display_name": "NewTestModel",
            "description": "Test model",
            "predictor_id": PREDICTOR_ID,
            "model_type": "HYBRID",
            "context_features": [
                {
                    "label": "Customer_Age",
                    "type": "integer",
```

```
                    "field_expr": "Customer_Age"
                },
                {
                    "label": "Customer_Location_Country",
                    "type": "string",
                    "field_expr": "Customer_Location_Country"
                },
                {
                    "label": "Customer_Intent",
                    "type": "string",
                    "field_expr": "Customer_Intent"
                }
            ],
            "action_features": [
                {
                    "label": "Agent_Skill4",
                    "type": "string",
                    "field_expr": "Agent_Skill4"
                },
                {
                    "label": "Agent_SalesConversionRate",
                    "type": "integer",
                    "field_expr": "Agent_SalesConversionRate"
                },
                {
                    "label": "Agent_Skill4_Level",
                    "type": "integer",
                    "field_expr": "Agent_Skill4_Level"
                }
            ],
            "train_data_percentage": 80,
        }
    )
    return response.json()

# DELETE /predictor_models/{id} Delete predictor model
def delete_model(token):
    response = requests.delete(
        '%s/api/v2.0/predictor_models/%s' % (HOST, MODEL_ID3),
        json={
            'token': token,
            'predictor_id': PREDICTOR_ID,
            'hard': True
        }
    )
    return response.json()

# DELETE /predictor_models (several models) Delete predictor models in batch
def delete_models_batch(token):
    response = requests.delete(
        '%s/api/v2.0/predictor_models' % (HOST),
        json={
            'token': token,
            'predictor_id': PREDICTOR_ID,
            'hard': True,
            'model_id': [
                MODEL_ID1,
                MODEL_ID2
            ]
        }
    )
    return response.json()
```

```
# PUT /predictor_models/{id}/train Train predictor model
def train_model(token):
    response = requests.put(
        '%s/api/v2.0/predictor_models/%s/train' % (HOST, MODEL_ID),
        json={
            'token': token,
            'predictor_id': PREDICTOR_ID
        }
    )
    return response.json()

# PUT /predictor_models/{id}/retrain Retrain predictor model
def retrain_model(token):
    response = requests.put(
        '%s/api/v2.0/predictor_models/%s/retrain' % (HOST, MODEL_ID),
        json={
            'token': token,
            'predictor_id': PREDICTOR_ID
        }
    )
    return response.json()

# PUT /predictor_models/{id}/activate Activate predictor model
def activate_model(token):
    response = requests.put(
        '%s/api/v2.0/predictor_models/%s/activate' % (HOST, MODEL_ID),
        json={
            'token': token,
            'predictor_id': PREDICTOR_ID
        }
    )
    return response.json()

# PUT /predictor_models/{id}/deactivate Deactivate predictor model
def deactivate_model(token):
    response = requests.put(
        '%s/api/v2.0/predictor_models/%s/deactivate' % (HOST, MODEL_ID),
        json={
            'token': token,
            'predictor_id': PREDICTOR_ID
        }
    )
    return response.json()

# PUT /predictor_models/{id}/copy Copy predictor model
def copy_model(token):
    response = requests.put(
        '%s/api/v2.0/predictor_models/%s/copy' % (HOST, MODEL_ID),
        json={
            'token': token,
            'predictor_id': PREDICTOR_ID
        }
    )
    return response.json()
```

Analysis Examples

```
# GET /analysis Get analysis reports data
def get_analysis(token):
    response = requests.get(
        '%s/api/v2.0/analysis' % HOST,
        json={
```

```
            'token': token
        }
    )
    return response.json()

# GET /analysis/{id} Get analysis report data by ID
def get_analysis_by_id(token):
    response = requests.get(
        '%s/api/v2.0/analysis/%s' % (HOST, REPORT_ID),
        json={
            'token': token
        }
    )
    return response.json()

# DELETE /analysis/{id} Delete analysis report
def delete_analysis_report(token):
    response = requests.delete(
        '%s/api/v2.0/analysis/%s' % (HOST, REPORT_ID),
        json={
            'token': token
        }
    )
    return response.json()

# POST /analysis/datasets/feature_analysis Generate Feature Analysis report for dataset
def generate_fa_report_dataset(token):
    response = requests.post(
        '%s/api/v2.0/analysis/datasets/feature_analysis' % HOST,
        json={
            'token': token,
            'dataset_id': DATASET_ID,
            'target_metric': 'HandleTime'
        }
    )
    return response.json()

# POST /analysis/predictors/feature_analysis Generate Feature Analysis report for predictor
def generate_fa_report_predictor(token):
    response = requests.post(
        '%s/api/v2.0/analysis/predictors/feature_analysis' % HOST,
        json={
            'token': token,
            'predictor_id': PREDICTOR_ID
        }
    )
    return response.json()

# POST /analysis/predictors/lift_estimation Generate Lift Estimation report for predictor
def generate_le_report(token):
    response = requests.post(
        '%s/api/v2.0/analysis/predictors/lift_estimation' % HOST,
        json={
            'token': token,
            'predictor_id': PREDICTOR_ID,
            'model_id': MODEL_ID
        }
    )
    return response.json()
```

Index Management Examples

```
# GET /index_management Get existing indexes on a collection
def get_indexes(token):
    response = requests.get(
        '%s/api/v2.0/index_management' % HOST,
        json={
            'token': token,
            'coll_type': 'agents'
        }
    )
    return response.json()

# PUT /index_management Edit indexes on a collection
def edit_indexes(token):
    response = requests.put(
        '%s/api/v2.0/index_management' % HOST,
        json={
            "token": token,
            "coll_type": "agents",
            "fields": [
                "Agent_Status",
                "Agent_Location_Country"
            ]
        }
    )
    return response.json()

# PUT /index_management Delete indexes on a collection
def delete_indexes(token):
    response = requests.delete(
        '%s/api/v2.0/index_management' % HOST,
        json={
            "token": token,
            "coll_type": "agents",
            "fields": [
                "Agent_Status",
                "Agent_Location_Country"
            ]
        }
    )
    return response.json()
```

PURGE APIs

```
# GET /purge/{purge_job_id} Get purge job details
def get_purge_status(token, purge_job_id):
    response = requests.get(
        '%s/api/v2.0/purge/%s' % (HOST, purge_job_id),
        json={
            'token': token
        }
    )
    return response.json()

# POST /purge/agents Add a purge job for agents
def purge_agents(token):
    response = requests.post(
        '%s/api/v2.0/purge/agents' % HOST,
        json={
            'token': token,
```

```
            'data_filter': '(Agent_Age>30)'
        }
    )
    return response.json()

# POST /purge/customers Add a purge job for customers
def purge_customers(token):
    response = requests.post(
        '%s/api/v2.0/purge/customers' % HOST,
        json={
            'token': token,
            'data_filter': '(Customer_Income<200000)'
        }
    )
    return response.json()

# POST /purge/datasets/{id} Add a purge job for dataset with given id
def purge_dataset(token):
    response = requests.post(
        '%s/api/v2.0/purge/datasets/%s' % (HOST, DATASET_ID),
        json={
            'token': token,
            'data_filter': '(CSAT=5)'
        }
    )
    return response.json()

# POST /purge/predictors/{id} Add a purge job for predictor with given id
def purge_predictor(token):
    response = requests.post(
        '%s/api/v2.0/purge/predictors/%s' % (HOST, PREDICTOR_ID),
        json={
            'token': token,
            'data_filter': '(ctx.Customer_Age=56)'
        }
    )
    return response.json()

def train_model(token):
    response = requests.put(
        '%s/api/v2.0/predictor_models/%s/train' % (HOST, MODEL_ID),
        json={
            'token': token,
            'predictor_id': PREDICTOR_ID
        }
    )
    return response.json()
```

Predictor Scoring Example

```
# POST /predictors/{id}/score Score actions for predictor context
def get_scores(token):
    response = requests.post(
        '%s/api/v2.0/predictors/%s/score' % (HOST, PREDICTOR_ID),
        json={
            'token': token,
            'context': {
                'Customer_Location_Country': 'US'
            },
            'action_filters': "Agent_Skill4_Level in ['3','4','5']",
            'format_as_map': True,
            'warnings': True
```

```
        }
    )
    return response.json()
```

System Requirements, Pre-Requisites, and Planning

This section contains information to get you started with your planning for an implementation of Predictive Routing. It contains the following topics:

System Requirements and Interoperability

Genesys Predictive Routing (GPR) includes several components. This topic provides an overview of the prerequisite hardware and software required to run each component.

It also includes an Interoperability table, showing which versions of the Genesys components required to run an end-to-end GPR solution are compatible.

> **Important**
>
> In addition to the prerequisites noted here, see the *Genesys Supported Operating Environment Reference Guide*, which provides operating system, database, and browser requirements information for most Genesys products.

The GPR Components: Hardware and Software Requirements

AI Core Services (AICS)

- Provides the Predictive Routing user interface, the API, and the scoring engine. This component consists of a number of Docker containers deployed from a single IP.

Important Considerations

- You might need an active internet connection to download additional libraries when installing Docker.
- The GPR uses CentOS 7 as the base Docker image.
- If you are deploying AICS in an HA architecture, the system clocks on all target servers must be synchronized. You can use Network Time Protocol (NTP) for this.
- For a list of ports required by the various Docker containers and components, see Required Ports for AICS Servers.

Considerations Related to Third-Party Software

- If you are deploying AICS in an HA architecture and running VMWare VXLAN, you might encounter a port conflict between VMWare VXLAN and Docker, both of which require port 4789. If you encounter this issue, Genesys recommends that you use a networking application such as Weave Net to manage networking among Docker containers. For additional information, consult the documentation for the respective products:
  - For the Docker Swarm port requirements: Use swarm mode routing mesh
  - For VMWare VXLAN port requirements: Ports and Protocols Required by NSX

- For Weave Net: Introducing Weave Net

Agent State Connector (ASC)

- Connects to Configuration Server (or Configuration Server Proxy) and (optionally, in release 9.0.015.04 and higher) Stat Server to read real-time updates on agents, agent groups, and agent availability. ASC passes these updates to AI Core Services, which connects to your Genesys Routing solution. With ASC release 9.0.015.04 and higher, and URS Strategy Subroutines 9.0.015.00 and higher you can choose the have GPR monitor agent availability through Universal Routing Server rather than Stat Server, reducing the connections required.
  The option to monitor agent availability from URS rather than Stat Server increases the load on URS. See the Sizing Guide for guidelines.

URS Strategy Subroutines/Composer Subroutines

- Editable out-of-the-box strategy subroutines to use with your Genesys routing components. Genesys Predictive Routing includes a set of subroutines created for use with Universal Routing Server (URS) and Interaction Routing Designer (IRD), and an analogous set that supports the addition of Orchestration Server (ORS) and Composer Universal Routing Server (URS) and Interaction Routing Designer (IRD).

Genesys Reporting Integration

- Configure GPR and your historical reporting components to ensure that the required data, in the form of key-value pairs, is made available to support Genesys historical reporting on GPR performance and outcomes.

General Recommendations

Genesys recommends that you set up at least two instances of Predictive Routing, a *test* or *development* instance and a *production* instance.

- The production instance runs Predictive Routing applications for both pre-production and production environments.
- The development instance runs a separate Predictive Routing application used for development and testing of the data collection pipeline.

System Requirements and Required Components/Versions

The following table lists the hardware and software requirements that should be in place before starting your deployment.

AI Core Services		
Hardware/Software Type	**Requirement**	**Comments**
OS	• CentOS Linux7 (64-bit) • RHEL 7 (64-bit)	

AI Core Services		
	• Oracle Linux 7.3	
RAM	32 GB	
CPUs	8 cores minimum	
Disk space	50 GB free space minimum required, 100 GB recommended	Ideally, the root directory should have at least 50 GB of free space to be used for operating system needs, Docker images, containers, and so on. The directory, `/datadir`, where MongoDB stores data should be on a separate volume with at least 50 GB, with the option to grow as needed.
Docker	docker-ce version 18.09.2 or higher; OR docker-ee 18.09.2 or higher	The recommended version was updated in March, 2019. For security reasons, *all* deployments should upgrade to the recommended Docker version.
MongoDB	version 3.6	The recommended version is required for GPR release 9.0.011.00 and higher. Earlier versions of AICS are compatible with earlier versions of Mongo DB.
Load Balancer	Depends on your environment	For production HA environments, Genesys recommends that you use a specialized load balancer, such as F5.
Agent State Connector (ASC)		
Hardware/Software Type	**Requirement**	**Comments**
OS	• CentOS Linux 7 (64-bit) • RHEL 7 (64-bit) • Oracle Linux 7.3 • Windows Server	
RAM	1 GB	
Java	Java JDK 1.8	
Configuration Server	8.1.300.26	
Stat Server	8.5.108.17	
Message Server—for logging	8.1.300.11	
URS Strategy Subroutines/Composer Strategy Subroutines		
Hardware/Software Type	**Requirement**	**Comments**
See the Genesys Predictive		

AI Core Services		
Routing Sizing Worksheet to calculate the memory and CPU requirements for URS/ORS when using Predictive Routing.		

Interoperability

Among GPR components:

GPR Component	Requirement	Comments
AI Core Services	AI Core Services 9.0.015.03 and higher requires Agent State Connector 9.0.015.04 and higher.	
Agent State Connector	Agent State Connector 9.0.015.04 and higher requires AI Core Services 9.0.015.03 and higher.	

For Routing using the URS Strategy Subroutines:

Hardware/Software Type	Requirement	Comments
Universal Routing Server	8.1.400.57	
Interaction Routing Designer	8.1.400.26 or higher	

For Routing using the Composer Subroutines. The addition of Orchestration Server and Composer enables you to use Composer to manage the routing workflow, but the Composer subroutine acts as a wrapper for the functionality implemented in the URS Strategy Subroutines component.

Hardware/Software Type	Requirement	Comments
Universal Routing Server	8.1.400.57	
Interaction Routing Designer	8.1.400.26 or higher	
Orchestration Server	8.1.400.40 or higher	
Composer	8.1.400.36 or higher	

For integration with Genesys Reporting:

Hardware/Software Type	Requirement	Comments
Genesys Predictive Routing	9.0.007 or higher	
Interaction Concentrator	8.1.5 or higher	
Genesys Info Mart	8.5.009.12 or higher is the base version. URS Strategy Subroutines 9.0.015.00 and	

Hardware/Software Type	Requirement	Comments
	higher requires Genesys Info Mart 8.5.014.09 and higher.	
Reporting and Analytics Aggregates	8.5.002 or higher	
Genesys Interactive Insights/ GCXI	8.5.001 or higher	

Architecture and Security

This topic presents Genesys Predictive Routing (GPR) architecture, first at a high-level overview, followed by more detailed views of the connections used by AI Core Services (AICS), Agent State Connector (ASC), URS Strategy Subroutines, and Composer Subroutines components.

This topic covers Genesys Predictive Routing architecture, with some additional Genesys components included in the diagrams for completeness. For a full list of required components and versions, refer to System Requirements and Interoperability.

In addition, you need to have adequate data source(s) and construct a well thought-out data pipeline.

- AI Core Services Architecture
- Agent State Connector Architecture
- Subroutines Architecture
- Security and Secure Connections

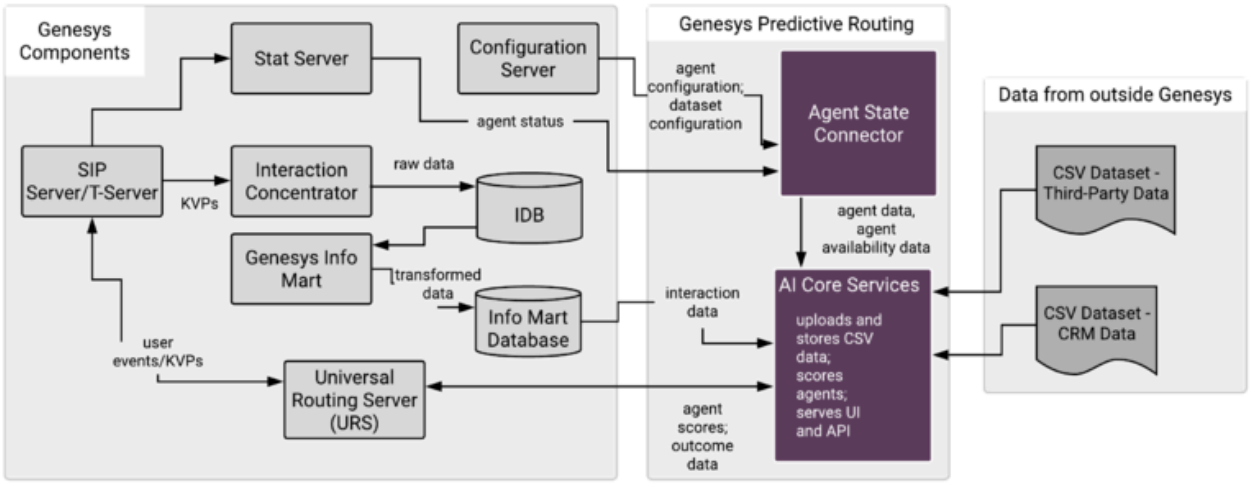
GPR Architecture

The following diagram shows a high-level view GPR, how it connects with other Genesys components, and how non-Genesys data enters GPR.

Important

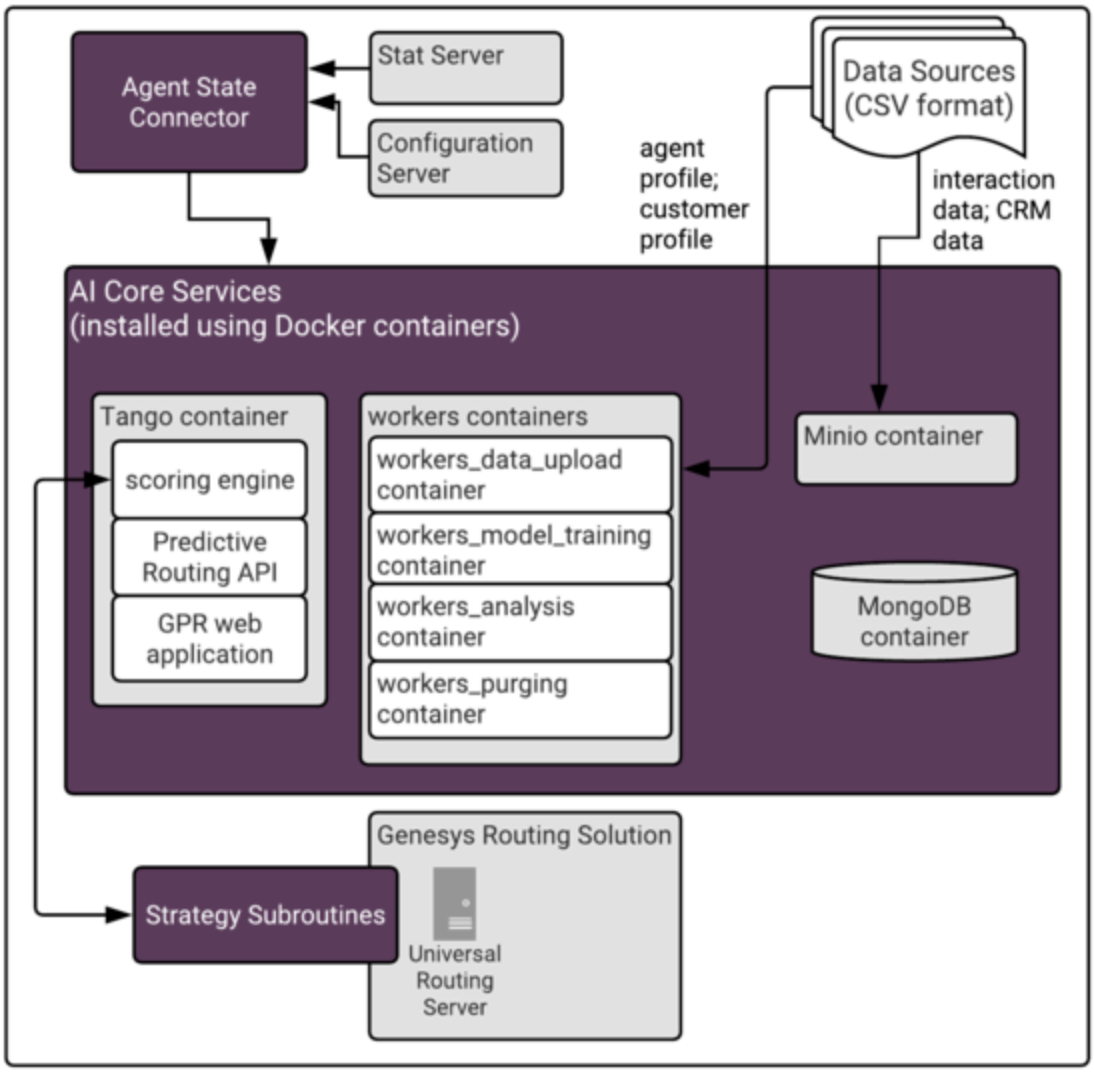
GPR architecture in a high availability (HA) environment is similar to that presented in the diagram except for the following details:

- Your load balancer or Docker container management application distributes traffic to AICS across the configured instances in whatever way your system architects choose. Deploy GPR in an HA Environment explains how to configure AICS if you require HA.
- ASC is a Java application that can be monitored, started, and stopped in Solution Control Interface. It supports a warm-standby high availability architecture.



AI Core Services Architecture

The following diagram shows the AI Core Services (AICS) internal structure and high-level connections to the other GPR components in a single-server deployment.



- See Scaling AI Core Services for information on how to scale each type of container included in the AICS deployment.
- See System Monitoring and Logging for information on how to access the logs for each container.

The Tango Container

Contains the Genesys platform that provides the GPR scoring engine, the Predictive Routing REST API, and the web-based user interface.

Workers Containers

Contain function-specific processes, identifiable by the descriptive container names.

> **Important**
> In earlier releases, the Tango container included the workers listed below. The functionality they provide has not changed. They have been relocated to separate containers to facilitate scaling of specific functional aspects.

- workers_dataset_upload - Uploads agent profile and customer profile data to MongoDB in releases prior to 9.0.015.03. Also uploads Dataset data in releases prior to 9.0.013.01, when the MinIO container was introduced. In release 9.0.015.03 and higher, the MinIO container performs the initial upload for all data, after which the workers_dataset_upload container moves the data from the server-based storage provided by MinIO to MongoDB.
- workers_model_training - Performs Model training jobs.
- workers_analysis - Runs the Feature Analysis, Lift Estimation, and Agent Variance reports.
- workers_purging - Purges your Dataset data.

The MinIO Container

Contains MinIO, which is a high performance, distributed object storage server, designed for large-scale data infrastructure. This container is available in releases 9.0.013.01 and higher, where it improves processing times for the initial Dataset upload. In release 9.0.015.03 and higher, this functionality has been expanded to include Agent Profile and Customer Profile uploads as well.

AICS handles data uploads without any need for you to handle configuration of MinIO. However, if you are interested in more detailed information about this component, see the MinIO web site and documentation.

The MongoDB Container

A highly scalable, highly available no-SQL database which is especially efficient at handling large batches of JSON format data. It also supports fast, efficient queries of that data. Starting in MongoDB 3.2, WiredTiger is the default storage engine for MongoDB.

In high availability (HA) deployments, MongoDB uses *replica sets* split across two data centers. A primary and secondary replica set are located in data center 1 (DC1) and a secondary replica set is located in DC2. All writes go to the primary replica set, from which they are distributed to the secondary replicas. Reads can be directed to either of the secondary sets. You can configure MongoDB to prefer reads from local secondary sets. Cross-site data traffic is required, however, because all writes are directed to the primary MongoDB in DC1 and the data then replicates across sites. Note that sufficient bandwidth will be required for the data replication traffic between data centers.

If the primary server fails, read operations can still continue. Another server can be elected as the primary server to continue write operations. Ideally, an Arbiter node should be set up in a third data center or availability zone. This facilitates the detection of a failed primary node when a data center

becomes inaccessible and the proper election of a primary in the other data center. If there are only two data centers, manual intervention is required to force one of the secondary replicants to become the primary replicant.

- Links to additional information about Mongo DB:
  - WiredTiger Storage Engine
  - https://eladnava.com/deploy-a-highly-available-mongodb-replica-set-on-aws/
  - https://docs.mongodb.com/manual/core/replica-set-architecture-geographically-distributed/
  - http://s3.amazonaws.com/info-mongodb-com/MongoDB_Multi_Data_Center.pdf
  - https://stackoverflow.com/questions/43083246/requires-simple-explanation-on-arbiters-role-in-a-givenmongodb-replica-set
  - https://docs.mongodb.com/manual/reference/method/Mongo.setReadPref/

The NGINX Container

> **Important**
>
> - Genesys recommends that you use NGINX only in test (non-production) environments.
> - The NGINX container was removed in release 9.0.015.03.

NGINX is open source software for web serving, reverse proxying, caching, load balancing, media streaming, and more. In addition to its HTTP server capabilities, NGINX is also used as a reverse proxy and load balancer for HTTP, TCP, and UDP traffic.

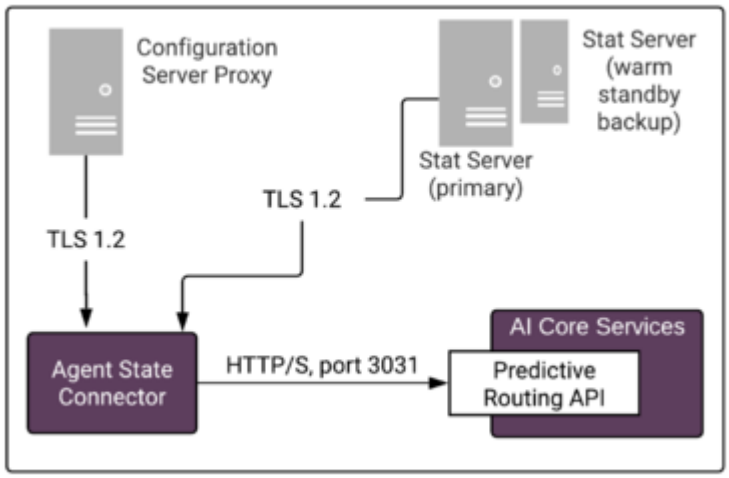
Agent State Connector Architecture

Agent State Connector (ASC) connects to Configuration Server Proxy for agent-related data to be stored in the Agent Profile (agent configuration details, such as a location, languages, skills and skill levels, and so on). You can use this connection to populate the entire Agent Profile or you can upload the initial agent data from a CSV file. In either case, agent data is updated via the connection to Configuration Server. See Configuring Agent Profiles in the *Predictive Routing Help* for additional information and procedures.

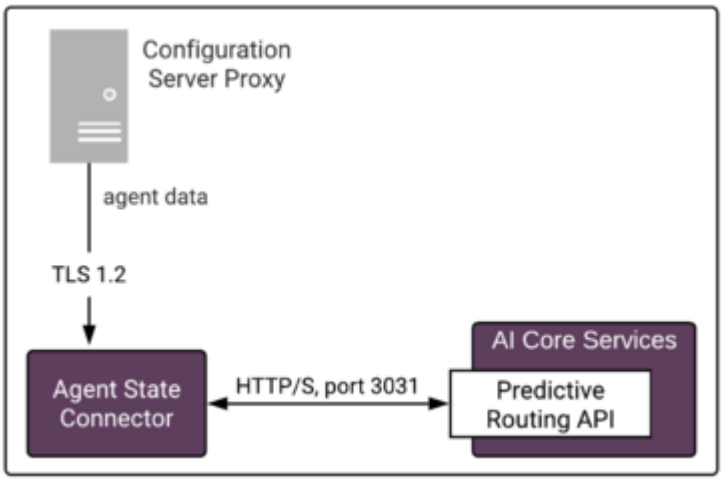
> **Important**
>
> Genesys recommends that you connect to Configuration Server Proxy, to reduce traffic on Configuration Server.

ASC is a Java application that can be monitored, started, and stopped in Solution Control Interface. It supports a warm-standby high availability architecture. Certain architectural details about ASC depend on the release you have deployed:

- In ASC 9.0.015.01 and lower, ASC connects to Stat Server to read agent availability data used in determining the preferred target agent and to configure and read the output for custom statistics.



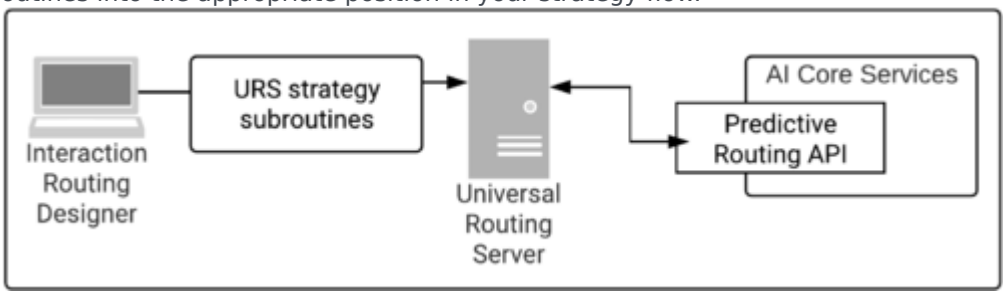
- In ASC 9.0.015.04 and higher, the connection to Stat Server is optional. If you do not add a Stat Server to the Connections tab of the ASC Application object, agent availability data is taken from Universal Routing Server (URS), reducing the number of connections required.



Subroutines Architecture

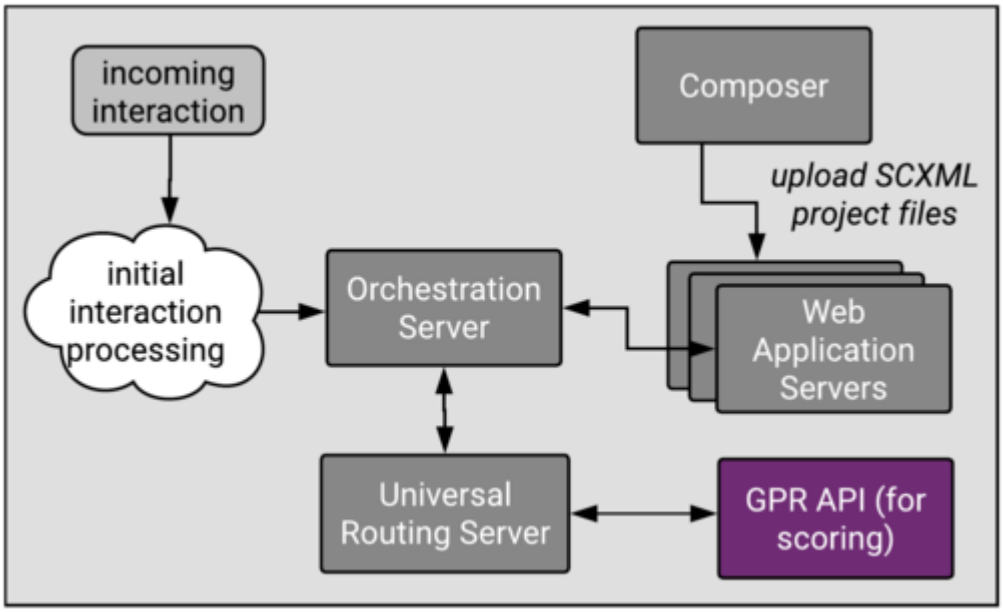
Predictive Routing supplies out-of-the-box subroutines for environments running either Interaction Routing Designer (IRD) + Universal Routing Server (URS) or Composer + Orchestration Server (ORS) + URS.

- IRD requires you to use the Predictive Routing URS Strategy Subroutines component. Insert the strategy subroutines into the appropriate position in your strategy flow.



- Composer requires the use of the Predictive Routing Composer Subroutines. Insert the subroutines into the appropriate position in your workflow. If you are using Composer, you need Orchestration Server (ORS) as well as URS in your environment.

> **Important**
>
> Predictive Routing is not supported for environments that use schedule-based routing.



The Subroutines invoke Predictive Routing in real time. They send a request to AICS, which performs the scoring based on the information you configured in your Predictor and the Model or Models based

on it. AICS returns the projected scores for each agent in the target group, indicating how well they would be expected to handle the specific interaction in question given the particular interaction type, customer intent, agent skill level, and whatever other factors you anticipate to be relevant. URS then chooses the optimal routing target.

Security

AI Core Services and Agent State Connector 9.0.015.05 and higher are delivered as a set of Docker images. This ensures consistent environments from development to production as Docker containers maintain all configurations and dependencies internally, without depending on software installed on host server. With Docker, upgrades are easier and more predictable. Scaling across multiple hosts requires starting the same Docker containers on multiple host servers. In addition, Docker provides isolation; every part of GPR can be scaled separately and has guaranteed access to hardware resources.

Genesys uses the following best practices when it comes to security:

- GPR supports TLS 1.2. To configure HTTPS connections, see Configuring GPR to Use HTTPS.
- GPR uses a CentOS 7 Docker image as the base image.
  - Genesys supports Security Enhanced Linux (SELinux) on CentOS 7. For a discussion of this functionality and how to configure it, see How to disable SELinux on the Linux web site.
- GPR Docker images containing Genesys software are continuously scanned for vulnerabilities as part of the build and test pipelines.
- All GPR Docker containers run in unprivileged mode.
- Inside Docker containers, GPR software is executed as a non-root user.
- All ports and volumes that should be exposed by each container are specified in Required Ports for Firewall Configuration.

The measures listed above, combined with properly secured host servers, ensures that GPR deployed using Docker containers is as secure as a deployment using more traditional methods.

- GPR delivered as set of Docker containers does not require any additional ports to be open.
- GPR uses MongoDB as its database, which is also delivered as Docker image. GPR uses the official MongoDB Docker image at **https://hub.docker.com/_/mongo/**.
- MongoDB inside the Docker container requires access to the same ports and same hardware resources as MongoDB running outside of a Docker container.

To understand how Docker containers comply with various security regulations and best practices, see the following pages on the Docker site:

- Docker standards and compliance.
- Docker Security

To understand how MongoDB databases comply with various security regulations and best practices, see the following page on the MongoDB site:

- MongoDB Security

Secure Connections

Predictive Routing supports the following security and connection protocols:

- ADDP
- HTTPS
- Transport Layer Security (TLS) 1.2

The following protocols are supported for the specified connections:

- ASC to Config Server: TLS 1.2; you can specify an upgrade-mode Configuration Server port by updating the **-port** command line parameter in the ASC `Application` object **Start Info** tab.
- ASC to Stat Server: TLS 1.2
- ASC to AICS: HTTPS
- URS or ORS to AICS: HTTPS

Configure GPR to Use HTTPS

GPR supports HTTPS by default. The procedures linked below provide the required configuration to use HTTPS with GPR.

HTTPS configuration for other components in your Genesys environment is covered in the *Genesys Security Deployment Guide* and in the product-specific documentation.

- Configure AICS to Use HTTPS
- Configure ASC to Use HTTPS
- Configure URS Strategy Subroutines/Composer Subroutines to Use HTTPS

Secure Logins

Predictive Routing supports LDAP authentication for user logins. See Settings: Configuring Accounts and Account: User Management for procedures to configure LDAP authenticated accounts.

Sizing Guide

Use the Excel worksheet linked below to calculate hardware sizing guidelines for all GPR components in your environment. This sizing guide provides guidelines for simple single-server deployments, single-server HA deployments, and multi-server HA deployments.

How to fill out the Sizing Guide Worksheet

The worksheet includes the following two tabs:

- **Cover**, which has version number, copyright, and legal information.
- **GPR Sizing**, which has three main sections:
  - Input values/customer sizing data - Enter the appropriate numbers for your environment. ***This is the only section you should edit.***
  - Calculations/outputs - This provides a few key values generated from your input in the first section. Check these first to ensure that you are getting output in line with your expectations for sizing.
  - Hardware requirements - This section contains the detailed sizing recommendations for each component.

    The worksheet provides descriptions for each field to guide you in understanding what should be entered or how to interpret the results. It also includes links that take you directly to relevant pages in this *Deployment and Operations Guide*.

Click here for the worksheet: Genesys Predictive Routing Sizing Guide

Prepare Your Data

This topic addresses issues such as data sources, what kind of data Genesys Predictive Routing (GPR) needs in order to function, how much data you need, and how best to structure it. See Supported Encodings and Unsupported Characters for information about constraints on data formats. It also includes data size guidelines and a list of Genesys Info Mart tables that provide data for GPR.

Because environments differ and the exact metrics you want to target for improvement are also dependent on your environment and business needs, this topic offers only general guidelines.

GPR can use any relevant data available in your environment. Some data is automatically imported into GPR, but the remainder must be combined into consistent schemas and saved as CSV files. You can use either the GPR web application or the API to import this CSV data. You can upload your CSV file itself or as a ZIP archive.

> **Important**
>
> - This *Guide* assumes you are using the GPR web application. For details about the API, see the *Predictive Routing API Reference*.
> - Genesys supports only the GPR application and the GPR API for uploading Dataset and Agent and Customer Profile data to Genesys Predictive Routing.
> - Once a Dataset schema is created and accepted in AICS, you cannot change the schema. If you need to modify the schema, you must delete the Dataset and create a new one with the desired schema.

In general, you need the following types of data:

Interaction data

- You must create a CSV file containing the required data and upload it, using either the GPR web application or the GPR API.

Agent Profile data

- ASC automatically extracts Agent Profile data from Configuration Server. See Configuring Agent Profiles in the *Predictive Routing Help* for how to configure an Agent Profile schema.

Customer Profile data

- You must create a CSV file and upload it, using either the GPR web application or the GPR API. See Configuring Customer Profiles in the *Predictive Routing Help* for how to configure a Customer Profile schema.

> **Important**
>
> If you are running AICS 9.0.015.00 or lower and your Customer Profile CSV file is very large, split your CSV file. Upload one section, then perform as many separate append operations as necessary to upload the entire Customer Profile. For example, in a medium-sized single-server deployment, (64 GB RAM; 16 CPU), each CSV upload should be no larger than 340 MB (approximately 100 features and 100000 rows). AICS 9.0.015.03 and higher uses MinIO for uploads, which resolves this issue.

Outcome Data

- You must create a CSV file and upload it, using either the GPR web application or the GPR API.

Correctly Specifying Data Types in Your Dataset

GPR automatically determines the data types of the columns in your dataset during dataset initialization by analyzing the first 1000 rows of each column. To ensure that GPR can make a correct determination, Genesys recommends that you insert a "dummy" row at the beginning of your dataset that contains values that can be unambiguously interpreted as the expected data types for each column. This prevents cases in which the first 1000 rows may contain all NULL or 0 values, which might lead to an incorrect data type assignment (since 0 can be a valid integer, float, or Boolean value). If a column does contain meaningful values, the dummy row is analyzed along with the other values and contributes to the data type determination. Genesys recommends you use the following data type specifications in your dummy row:

- `'a_string'` - Is recognized as a string.
- `2.1`, or any integer or float value > 1 - Is recognized as a float.
- `False` or `True` - Is recognized as Boolean.
- Unix Timestamp (such as 1535538976) - Is recognized as a timestamp.

Supported Encodings

By default, GPR handles data using UTF-8 encoding. However, starting with release 9.0.014.00, GPR supports importing of data that uses certain legacy encodings. Appendix: Supported Encodings lists those encodings currently supported. This list is updated as new encodings are verified. If you use an encoding type that is not listed, contact your Genesys representative for assistance.

> **Important**
>
> All responses and returned data is provided in UTF-8 encoding.

Unsupported Characters in Agent and Customer Profiles and Datasets

The following characters are not supported for column names in Datasets or Agent and Customer Profile schemas. If GPR encounters these characters in a CSV file, it reads them as column delimiters and parses the data accordingly.

- | (the pipe character)
- \t (the TAB character)
- , (the comma)

Workaround: To use these characters in column names, add double quotation marks (" ") around the entire affected column name, except in the following situations:

- If you have a comma-delimited CSV file, add double quotations marks around commas within column names; you do *not* need quotations for the \t (TAB) character.
- If you have a TAB-delimited CSV file, add double quotations marks around TAB characters within column names; you do *not* need quotations for the , (comma) character.
- You must *always* use double quotations for the | (pipe) character.

Unsupported characters in releases prior to 9.0.014.00

In releases prior to 9.0.014.00, certain characters in column names are ignored, are unsupported, or cause an upload to fail, as explained in the following points:

- Columns with the following symbols in their column names are not added to Agent Profiles or Customer Profiles:
  `*, !, %, ^, (, ), ', &, /, â, è, ü, ó, â, ï`
- The following symbols in column names are ignored, and the column is added with the symbol dropped out as though it had not been entered:
  `[Space], -, <`
- Non-ASCII characters are not supported. How they are handled differs depending on what data you are uploading:
  - In Agent Profiles and Customer Profiles, columns with non-ASCII characters in the column name are not added.
  - In Datasets, when a column name contains a mix of ASCII and non-ASCII characters, GPR removes the non-ASCII characters from the column name as though they had not been entered and correctly uploads all column values.
  - In Datasets, when a column name contains only non-ASCII characters, the column name is entirely omitted. All the column values are preserved, but you cannot modify or save the schema. In this scenario, GPR generates the following error message: `An unhandled exception has occurred: KeyError('name')`.

Logs for Unsupported Characters

The following Agent State Connector log messages record issues with unsupported characters:

- `<datetime> [47] ERROR <BOTTLE> schema_based.py:63 Invalid expression while parsing: <fieldname> = None`
- `<datetime> [47] ERROR <BOTTLE> agents.py:172 Fields set([u'<fieldname>']) were ignored because names were invalid.`

Considerations

> **Important**
> Genesys testing has identified certain guidelines about data size to keep in mind while creating datasets and planning modeling and feature analysis.

- What metrics of Contact Center operation do you want to optimize?
- What databases are available?
- Is it possible to join the data from those databases?
- Is the data clean and consistent in format? This is especially an issue if you are joining data from various sources.
- Do you have enough data to draw conclusions about agents performance?
- What data is available from the IVR and the CRM database at runtime?

Note that when you create your CSV file, you will need to include a value, a `context_id` or customer ID, that can be entered into your scoring request from your strategy and that enables you to retrieve the relevant record.

Data Size Guidelines - Data Import, Model Training, and Feature Analysis

Description	Column Count	Row Count/File Size
Size of data that can be uploaded to create a dataset in a single batch. You can append data; you can load bigger datasets in multiple batches. Data uploads successfully for a file with 250 columns but with a smaller number of total records so that the total file size is less than the 250 MB limit.	250 columns maximum	250 MB file size
Minimum number of records needed to train a DISJOINT model	Not Applicable	10

Description	Column Count	Row Count/File Size
for an agent.		
Total Cardinality limit for model training. **Total Cardinality** = the number of numeric columns plus the sum of the number of unique values across all string columns within a given dataset.	No specific column count; has been tested up to 250 columns.	Total Cardinality should be less than 2 to the power of 29.
Record count limit for GLOBAL model training.	Not Applicable; from a model-training perspective there is virtually no limit on the number of columns. The constraining issue is the possibility of compromising the Global model quality by ending up with a reduced number of samples for training.	The total number of records should be less than 2 to power of 29 (that is, 536870912) divided by Total Cardinality as defined above. • **Example 1:** You are required to use ALL of the data for training the GLOBAL model (note that the GLOBAL model is trained even if you select DISJOINT, so that the scoring engine can rank agents who do not yet have data). The dataset contains 1 million records. Therefore the maximum total cardinality is 536 (536870912 divided by 1 million) . • **Example 2:** You can *undersample* the data for training the GLOBAL model—that is, use fewer than the ideal number of records for training. You might take 10,000 as the total cardinality, but only 53,687 of your total of 1 million records will be used for training. The calculation to determine this is 10,000 * 53,687 = 536870912 (the maximum cardinality).
Column count limitation on the Feature Analysis report, Agent Variance report, Lift Estimation report, and Model Quality report.	250	Not Applicable

Genesys Info Mart Data

The following table represents the data required from Genesys Info Mart (GIM) and the Genesys Configuration Database.

Domain	Fields	Table
Agent Profile	Agent username	RESOURCE_
Agent Profile	Employee ID	RESOURCE_
Customer Profile	CustomerID	Attached Data to map with interaction metadata record
Customer Profile	SERVICE_TYPE	Attached Data to map with interaction metadata record
Interaction Metadata	INTERACTION ID	Interaction_Fact (IF)
Interaction Metadata	INTERACTION TYPE	IRF
Interaction Metadata	MEDIA TYPE	Media_Type
Interaction Metadata	RESOURCE ROLE	TECHNICAL_DESCRIPTOR
Interaction Metadata	ROLE REASON	TECHNICAL_DESCRIPTOR
Interaction Metadata	TECHNICAL RESULT	TECHNICAL_DESCRIPTOR
Interaction Metadata	RESULT REASON	TECHNICAL_DESCRIPTOR
Interaction Metadata	MEDIA RESOURCE (VIRTUAL QUEUE)	RESOURCE_
Interaction Metadata	RESOURCE GROUP COMBINATION	RESOURCE_GROUP_COMBINATION
Interaction Metadata	ROUTING TARGET	ROUTING_TARGET
Interaction Metadata	TARGET_OBJECT_SELECTED	TARGET_OBJECT_SELECTED
Interaction Metadata	SKILL EXPRESSION	REQUESTED_SKILL
Interaction Metadata	START_TS	INTERACTION_FACT
Interaction Metadata	END_TS	INTERACTION_FACT
Interaction Metadata	LAST ROUTING POINT	RESOURCE_
Interaction Metadata	LAST VQ	
Interaction Metadata	IS_LAST_RESOURCE	IRF
Interaction Metadata	SOURCE_ADRESS (ANI,..)	SOURCE_ADDRESS
Interaction Metadata	TARGET_ADRESS (DNIS,..)	TARGET_ADDRESS
Interaction Metadata	WAITING TIME	(QUEUE DURATION, ROUTING POINT DURATION, MEDIATION DURATION)
Interaction Metadata	RINGING TIME	RING_DURATION
Interaction Metadata	HANDLE TIME	(Talk_Duration, ACW Duration,..)
Interaction Metadata	HOLD TIME	HOLD_DURATION
Interaction Metadata	HOLD COUNT	HOLD_COUNT
Interaction Metadata	ACW COUNT	AFTER_CALL_WORK_COUNT
Interaction Metadata	FOCUS TIME	FOCUS_TIME

Domain	Fields	Table
Interaction Metadata	CONSULTATION TIME (CONS_RCV_TALK_DURATION, POST_CONS_XFER_TALK_COUNT,...)	

Appendix: Supported Encodings

By default, GPR handles data using UTF-8 encoding. However, starting with release 9.0.014.00, GPR supports importing of data that uses the legacy encodings listed below. This list is updated as new encodings are verified.

> **Important**
>
> If you use an encoding type that is not listed, contact your Genesys representative for assistance.

- UTF-8
- Shift-JIS

Install and Configure Predictive Routing

This section provides deployment and configuration instructions for AI Core Services (AICS) and Agent State Connector (ASC) in both single-server and high availability environments.

- For how to integrate the Predictive Routing Subroutines with your Genesys routing environment, see Integrate with Genesys Routing

This section contains the following topics:

AI Core Services Single-Host Deployment

AI Core Services (AICS) is deployed using Docker containers on hosts running Docker Service. It is required that you deploy Docker Service in your environment, and Genesys provides the AICS installation file, which includes the multiple Docker images packaged as .tgz files. These images are loaded to the Docker Engine during the installation phase and are started in the desired pattern to ensure services are configured properly.

> **Warning**
>
> The instructions on the majority of this topic are intended only for **new deployments**. In particular, do not run the **start.sh -I** script if you have an existing version of Predictive Routing running in your environment. It will clear all existing data from your database and would result in data loss. *To upgrade an existing deployment of Predictive Routing*, following the instructions in Install into an Existing AICS Deployment.

Target Server Requirements and Recommendations

You must have the prerequisite hardware and software available and have performed the necessary preliminary steps to install and start Docker before installing AICS. Note that some of the steps require **sudo** access to the target servers.

- For hardware and software prerequisites, see AICS Prerequisites.
- You must have at least 50 GB free disk space on your root partition. For complete sizing requirements, see the sizing worksheet linked from Sizing for Premise Deployments.
- The target server cannot be a Docker image. It can be a virtual machine.
- By default, AICS deployed on a single host uses port 443 to be open for access to the API and web application. If you need to change the default port assignment, see Change the Default Port for AICS.
- For faster Dataset uploads (in release 9.0.013.01 and higher) and Agent and Customer Profiles (in release 9.0.015.03 and higher), AICS uses a separate service, which requires that port 9000 be open and accessible to external world on the public IP address of the target server. If port 9000 is not open and available, you can still upload Datasets, but at a reduced upload speed.
- SELinux (Security Enhanced Linux) should be disabled or running in permissive mode. See How to disable SELinux for instructions.

Preliminary Steps: Install and Start Docker

Important

- Genesys does not ship Docker as a part of AICS. You must install Docker in your environment before you can load the AICS containers. See AI Core Services system requirements for supported Docker versions.
- You might need an active internet connection to download additional libraries when installing Docker.

1. Install Docker-ee or Docker-ce. Click the desired version to access the relevant deployment instructions on the Docker site.
   A Docker deployment provides a complete self-contained environment, so that you do not need to manually configure ports or address compatibility issues for communication among the Docker containers that comprise AICS. All of that is taken care of ahead of time, and the completed Docker containers work together seamlessly upon deployment.
2. Create a new user to be used for installing and starting AICS. This user must be a member of the **docker** Linux group that you created during the Docker installation process. In most cases, the following set of Linux commands is enough to create the user needed for installing and starting AICS:

   ```
   $ sudo useradd PR_USER
   $ sudo usermod -aG docker PR_USER
   $ sudo usermod -aG systemd-journal PR_USER
   $ sudo passwd PR_USER
   ```

   - You must have **sudo** rights to execute these commands.
   - In the example commands, the user is given as *PR_USER*. You can replace this user name with any valid Unix name. This topic refers throughout to PR_USER; if you choose a different name, substitute that actual name in its place.
3. Grant SSH access to PR_USER so that you can copy the AICS installation package to the target server. The AICS installation package should always be copied to the target server by PR_USER into the PR_USER home directory.
4. To enable the Docker service, execute the following command:

   ```
   $ sudo systemctl enable docker
   ```
5. To start the Docker service, execute the following command:

   ```
   $ sudo service docker start
   ```

Preliminary Step: Create a Separate Disk for the MongoDB Database

Always use a separate disk partition for storing MongoDB data. This partition should be mounted as `/datadir`. The size of partition depends on your expected data usage, but it must be at least 50 GB. For disk partitioning, use standard Linux tools.

> **Important**
> The `/datadir` partition **MUST** exist before you install GPR and the user who is executing the GPR installation should have write access to the partition.

Use the following command to check how much space MongoDB is currently using:

```
$ du -h /datadir
```

In HA scenarios, checking one node is enough because the same data is replicated on all nodes.

Installing and Configuring AICS

The following instructions are intended only for new installations into a fresh environment. If you have previously installed AICS using Docker containers, see Install into an Existing AICS Deployment.

A fresh AICS installation consists of the following steps:

1. Unzip and Unpack the Repository File
2. Install AICS
3. Initialize the Application
4. Restart the Containers
5. Verify the Installation
6. Set Values for the AICS-Related Configuration Options - some configuration options are mandatory
7. Set Values for Environment Variables - some environment variables are mandatory
8. Configure AICS to Use HTTPS - mandatory
9. Scale the AICS Deployment (jumps to the Scaling AICS topic in this *Guide*)
10. Access the Logs for AICS (jumps to the Logging topic in this *Guide*)
11. Clean Up Disk Space
12. (Optional) Configuring AICS for Large Datasets
13. (Optional) Change the Default Port for AICS
14. (Optional) Turn on SSL for NGINX
15. (Optional) View Container Disc Usage
16. (Optional) Back Up Your Data
17. (Optional) Map a Local Volume to a Container
18. (Optional) Uninstall AICS

Important

- Genesys strongly recommends that you do NOT use the **root** user to install and start AICS. On Linux, the root user should be used only for administrative tasks.
- You do not need to have **sudo** rights to install and start AICS. **Sudo** rights are required only when executing the preparatory steps documented above.
- All steps required to install, start, restart, or upgrade AICS should be executed as PR_USER.
- MongoDB connections among the Workers, MinIO, and Tango containers uses SSL (TLS 2.0).

Unzip and Unpack the Repository File

1. Copy the **IP_JOptPlatform_<*version_number*>_ENU_dockerlinux.zip** file to the desired installation directory. Genesys recommends that you use the PR_USER home directory as destination for the AICS installation package.
2. Unzip the **IP_JOptPlatform_<*version_number*>_ENU_dockerlinux.zip** file, using the Linux `unzip` command, to access the **IP_JOP_PRR_<*version_number*>_ENU_linux.tar.gz** repository file.
3. To unpack the **IP_JOP_PRR_<*version_number*>_ENU_linux.tar.gz** repository file, execute the following command:

   ```
   $ tar -xvzf IP_JOP_PRR_<version_number>_ENU_linux.tar.gz
   ```

   This creates the **IP_JOP_PRR_<*version_number*>_ENU_linux** directory.

   All bash scripts required to install and operate AICS can be found in the **IP_JOP_PRR_<*version_number*>_ENU_linux/scripts/** directory.

The Predictive Routing scoring engine, API, and web interface are all deployed in a single container, which has the internally-used informal name of Tango.

In addition, Gunicorn `workers_*` containers provide specific functionality, indicated by the container names. The number and functions of the Workers containers differ in some releases. In earlier versions of AICS, many of the functions later allocated to separate Worker containers were performed within the Tango container.

The AICS package also includes various third-party components:

- MongoDB: Stores data needed for scoring agents and making matching predictions.
- NGINX: (The NGINX container was removed in release 9.0.015.03.) A front-end proxy serving content to the Predictive Routing server. Can also be used *in non-production environments only* for load-balancing in multi-server high-availability architectures.
- Gunicorn: A Python WSGI HTTP Server for UNIX, which is a pre-fork worker model. The Gunicorn server is broadly compatible with various web frameworks, simply implemented, light on server resources, and fairly speedy.

- MinIO: Starting in release 9.0.013.01, the MinIO container provides fast dataset uploads and for temporary dataset storage after upload to GPR. In release 9.0.015.03 and higher, Agent Profile and Customer Profile data is also uploaded using MinIO. (Data is stored in MongoDB in the long-term.)

Install AICS

Perform the installation using PR_USER (or the name you assigned in the Preliminary Steps , above). Installation does not require **sudo** rights and should NOT be done by the root user.

To install AICS, execute the **install.sh** script:

```
$ cd IP_JOP_PRR_<version_number>_ENU_linux/scripts/
bash install.sh
```

If you are managing your MongoDB deployment externally, run the **install.sh** script with the **-externalMongo** flag, as follows:

```
$ bash install.sh -externalMongo
```

The installation script verifies that the target server has sufficient hardware resources. If not, the installation process terminates without performing the install.

The installation script also checks the following:

- Installed Docker package is docker-ce or docker-ee.
- Docker version is supported.
- Number of cores in the server is sufficient.
- Root partition free space is sufficient.
- PR_USER (or your username) belongs to the Docker group..
- PR_USER (or your username) belongs to the systemd-journal group (for container log checking).
- Docker service is running.

Configure AICS

After installation but *before starting AICS*, perform the following steps:

1. Configure the S3_ENDPOINT environment variable, as explained in Set Values for Environment Variables (below).
2. Ensure that port 9000 is open and available (not blocked by a firewall).

These steps ensure that you can take full advantage of fast dataset uploading provided by MinIO.

Initialize the Application

Use the following command to initialize the application database and start the application. This command also sets the password for your default user, super_user@genesys.com. Replace the

variable <'my_password'> in the command below with a strong password, and record it securely for future reference.

```
$ cd scripts; bash start.sh -l -p <'my_password'>
```

> **Warning**
> Loading initial data erases any existing data stored in database!

Use the following information to access the Predictive Routing application from a browser:

- **URL:** https://*<server_ip_address>*
- **username:** super_user@genesys.com
- **password:** the password you specified

By default, the AICS installation procedure creates two instances of the model_training worker container and one instance of each of the other worker containers. For a detailed discussion of how and when to scale all the AICS containers, see Scaling AICS.

You can change the number of workers using the following commands:

```
$ cd IP_JOP_PRR_gpr_rc_ENU_linux
$ ./docker-compose -f scripts/docker-compose.yml -p workers scale model_training=4
$ ./docker-compose -f scripts/docker-compose.yml -p workers scale analysis=2
$ ./docker-compose -f scripts/docker-compose.yml -p workers scale purging=2
```

To stop the application run:

```
$ cd scripts; bash stop.sh
```

Troubleshooting the Initialization Process

If you need to troubleshoot execution of the **start.sh** script:

- Run the following script:

```
$ cd scripts; bash -x start.sh
```

It shows every command executed and the resulting output.

To turn on the DEBUG level of logging:

1. Add the line `LOGLEVEL=DEBUG` to the **conf/tango.env** file
2. Restart the application.

Restart the Containers

To restart the Docker containers run the **restart.sh** script:

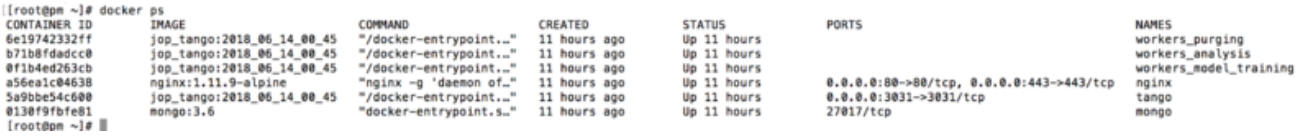
```
$ cd scripts; bash restart.sh
```

Verify the Installation

To check the status of the containers, run the following command on the target server:

```
$ docker ps
```

You should see output similar to the following:



Check the AICS logs for the various containers, as described in Operations: System Monitoring and Logging .

Set Values for the AICS-related Configuration Options

AICS-related options are configured on the `Predictive_Route_DataCfg Transaction List` object.

- For descriptions of the options, including default and valid values, see Deploying: Configuration Options .
- For instructions on setting values for configuration options, which is done on the **Configuration** tab of Genesys Administrator Extension, see Configuration Manager in the *Genesys Administrator Extension Help*.

Set Values for Environment Variables

This section lists environment variables that must or should be configured for optimal GPR performance, and the recommended values. Adjust these values as necessary, based on your specific environment.

> ### Warning
> The **tango.env** file, which contains the environment variables, is overwritten when you perform a software upgrade. Before upgrading, save a copy of the **tango.env** file and refer to it to reset your variables. Note that if you simply overwrite the new **tango.env** file with your existing one, any environment variables added in the new release are removed.

Environment variables are defined in the **IP_JOP_PRR_<*version_number*>_ENU_linux/conf/tango.env** file. The same file is used for both single node and HA deployments.

To add a new variable:

1. Create a new line in the **tango.env** file.
2. Add the variable and its value, using the following format:
   `<NEW_ENV_VAR>=value`

> **Important**
>
> - Do not use quotes for string parameters.
> - Remove trailing spaces.

Changes take effect on restart of the tango container (run the `bash scripts/restart.sh` command command). In an HA environment, with multiple instances of the containers running, restart is performed sequentially (a rolling restart), so that there is no downtime of the GPR application.

Configurable Environment Variables

- **ADD_CARDINALITIES_EVERY_N_RECORDS** - When you append data to an Agent or Customer Profile via the API, cardinalities are computed only for the appended data portion and only when the number of agents or customers set in the ADD_CARDINALITIES_EVERY_N_RECORDS parameter is reached. The results of computation are added to the already-stored cardinality values. This significantly improves speed when loading new data by avoiding simultaneous recomputations on the full data collection when there are multiple frequent appends done in small batches.enables you to specify how many appended records are added to an Agent or Customer Profile before GPR recalculates cardinalities. The default value is `1000`.
  - Notes:
    - This functionality is available only when you use the Predictive Routing API. If you append using the Predictive Routing application interface, all cardinalities are recalculated.
    - Full automatic computation happens only once, when an Agent or Customer Profile is uploaded the first time for schema discovery.
    - You can force recomputation of cardinalities on the full Agent or Customer Profiles collection using the POST **compute_cardinalities** API endpoint. For details, see the *Predictive Routing API Reference*. (This file requires a password to open it. Contact your Genesys representative if you need access.)
- **AUTOGENERATE_INDEXES** - Instructs GPR to create indexes on all Datasets, Agent Profile schemas, and Customer Profile schemas. By default, set to `True`.

  > **Important**
  >
  > Genesys strongly recommends you to leave the default value for this variable.

- **HOST_DOMAIN** - Use this variable to specify the public IP address or host name used for your deployment. The value should be one of the following, depending on your environment type:
  - For single-server deployments, specify the public IP address or the host name of the host where GPR is deployed.

  - For high availability (HA) deployments, specify the IP address of your load balancer.
- **LOG_LEVEL**
  - **INFO** - Informational messages that highlight the progress of the application: **LOG_LEVEL**=INFO. This setting is recommended for production deployments.
  - **DEBUG** - Fine-grained informational events that are most useful to debug the application: **LOG_LEVEL**=DEBUG. This setting should be used only for short periods of time because it can fill the disk.
- **LOGIN_MESSAGES** enables you have the Predictive Routing application display a custom message on the login screen.
  - When you enter this message, make sure that all special characters are properly escaped. *Special characters* are ones not part of the standard English alphabet, such as symbols, letters with umlauts, cedillas, and other such marks, and letters from other alphabets, such as the Greek or Cyrillic alphabets.
  - To simplify the task of converting characters, Genesys recommends an online conversion tool, such as **https://www.freeformatter.com/html-escape.html**.
  - For example, make the following substitutions:
    - & becomes &
    - < becomes <
    - > becomes >
    - " becomes "
    - ' becomes '
- **OMP_NUM_THREADS** (required for releases prior to 9.0.011.00; in releases 9.0.011.00 and higher, this parameter is set automatically)
  - Genesys recommends that you set the value to OMP_NUM_THREADS=1 for the best performance.
  - If you do not specify a value, GPR spawns one thread for each core it detects in your environment. The system assumes it can use all available cores for tasks such as analysis and model training, leaving no CPU resources for other processes running on the same machine, such as reading/writing to the database. The result is an overall slowdown of the application. Set this variable to allow the operating system to properly distribute CPU threads among the various running processes.
- **S3_ENDPOINT** - (Mandatory) The endpoint for the Minio container, introduced in AICS release 9.0.013.01 for Dataset uploads and expanded to Agent and Customer Profile uploads in AICS 9.0.015.03. Specifies the public IP address or domain name of the server where AI Core Services is installed, followed, optionally, by the port number.
  - The port number must always be 9000, which is the mandatory port value allocated for the Minio container.
  - In HA environments, locate the server on which the Minio container is running and use the public IP address or the domain name of that server. For example:
    For an IP address - `S3_ENDPOINT=https://<public_ip_address>:9000`

    For a domain name - `S3_ENDPOINT=https://<your_domain_name>:9000`
  - The S3_ENDPOINT value must always use the HTTPS protocol. If you do not configure this variable, the **start.sh** script generates a warning message and stops deploying AICS.
- (Optional) **GUNICORN_TIMEOUT**

- Adjust the timeout if you need to accommodate a large Dataset. The current default value is `600` seconds.

Configure AICS to Use HTTPS

The procedures here are those required to use HTTPS for connections among GPR components. HTTPS configuration for other components in your Genesys environment is covered in the *Genesys Security Deployment Guide* and in the product-specific documentation.

AICS supports HTTPS by default. Before you start using AICS, your organization should provide the certificates appropriate for your environment to enable the HTTPS connection protocol to work correctly. Genesys does not specify which certificates you should use.

After you have obtained the certificates, your procedure depends on your architecture:

- In a single-server environment, follow the procedure below.
- In a high availability (HA) environment, follow the instructions provided in the documentation for your load balancer. In an HA environment, you do not need to deploy the certificates on the individual nodes.

Single-Server Environment

1. Copy the certificates to the ***<GPR_IP_version>*/conf** folder.
2. Rename the default certificates originally located in that folder using the following commands:
```
$ mv tango.crt tango.crt_orig

$ mv public.crt public.crt_orig

$ mv tango.key tango.key_orig

$ mv private.key private.key_orig
```
3. Rename the new certificates using the following commands:
```
$ cp cert.pem tango.crt

$ cp cert.pem public.crt

$ cp priv_key.pem tango.key

$ cp priv_key.pem private.key
```
4. Open the **tango.env** file and change the value for the **S3_ENDPOINT** variable from the IP address to the DNS name.
   For example, replace **S3_ENDPOINT**=`https://18.217.189.106:9000` to **S3_ENDPOINT**=`https://fce-u0009.us.int.genesyslab.com:9000`
5. Restart AI Core Services.

> **Important**

- When you use the GPR web application, check that the URL starts with `https://`.

Next Steps

- Configure ASC to Use HTTPS - When you configure HTTPS for Agent State Connector, use the same certificate as for AICS
- Configure URS Strategy Subroutines/Composer Subroutines to Use HTTPS

Unusual HTTP/S Deployment Scenarios

- Default local certificate - Genesys ships AICS with a default local certificate. You can use that local certificate to access the GPR web application for internal testing purposes. Note the following points when using the local certificate:
  - The browser displays a Not Secure connection warning.
  - You **cannot** use the default certificate to configure connections from Agent State Connector or the Subroutines components to AICS.
- Self-signed certificate - *For lab environments only* - You can use OpenSSL to generate a self-signed certificate. You can use this self-signed certificate to configure secure connection between AICS and the other GPR components, as explained in the instructions for configuring HTTPS for ASC and the URS Strategy Subroutines.
  1. Generate a self-signed certificate by executing a command following the format in the following example:

     ```
     $ openssl req -new -newkey rsa:4096 -days 365 -nodes -x509 -subj "/C=US/ST=US/
     L=US/O=IT/OU=IT Department/CN=<ip address of the server where GPR is deployed>"
     -keyout tango.key -out tango.crt
     ```

  2. Accept the invalid certificate warning that appears when you open the GPR web application.
  3. If Dataset uploads from the GPR web application fail, navigate to the GPR Minio container at **https://<path_to-minio>:9000** and accept the security warning about the invalid certificate. You can then perform your Dataset uploads.
- HTTP connections in test environments - In AICS release 9.0.015.04 and higher, you can optionally configure HTTP connections.

  > **Warning**
  > HTTP connections are supported only in test environments. Genesys strongly recommends using the default HTTPS configuration in production environments and in lab environments that contain sensitive data. Genesys is not responsible for any potential damage and/or data loss if the solution is implemented without the recommended security practices and protocols.

  To use HTTP connections, perform the following steps:

  1. Comment out the following lines in the **/scripts/docker-compose.yml** file:

```
#    - ../conf/tango.key:/data/ssl/tango.key
#    - ../conf/tango.crt:/data/ssl/tango.crt
```

2. Edit the port configuration in the **/scripts/docker-compose.yml** file, as follows:
   Change

   ```
   443:3031
   ```

   to

   ```
   80:3031
   ```

3. Save your changes.
4. Restart GPR by running the following command:

   ```
   $ bash scripts/restart.sh
   ```

Clean Up Disk Space

Starting in release 9.0.013.01, GPR performs automatic cleanup processes which should maintain an adequate amount of free disk space. However, if you are running an earlier version of AICS, or are running 9.0.013.01 or higher and continue to encounter disk space problems, refer to the instructions in this section.

You might encounter performance issues if you do not clean up Docker data that is no longer required. The Docker `prune` command enables you to clean up your Docker environment. The Docker user documentation provides a detailed discussion of the `prune` command and how to use it to clean up images, containers, and volumes; see Prune unused Docker objects.

> **Important**
> The clean-up process does not affect normal GPR operation. It does not require downtime, there is no need to restart any component, and performance is unaffected.

Clean-Up Procedure

Genesys recommends that you use the following commands to remove unnecessary Docker data:

```
docker container prune -f
docker volume prune -f
docker network prune -f
```

To schedule regular cleanup jobs, use the `crontab` functionality to execute the appropriate command on every server where GPR is installed. The following example schedules the cleanup job for every Saturday at 1:00 am:

```
echo "0 1 * * Sat (docker container prune -f; docker volume prune -f; docker network prune
-f)" ) | crontab -
```

In an HA environment, Genesys recommends that you perform the cleanup on each node in turn.

If you need to configure your logging settings to avoid unacceptable log file sizes, see the following information:

- The LOG_LEVEL environment variable
- Configure AICS Log Settings

(Optional) Configuring AICS for Large Datasets

A "large" dataset is one that contains more than 1.5 million rows/250 columns. No more than 100 of the columns should contain high-cardinality values. Genesys recommends that you adjust your dataset to stay within these size limits.

Reconfiguring the GUNICORN_TIMEOUT Parameter

To accommodate a large dataset, you might need to configure the `GUNICORN_TIMEOUT` environment variable, which is located in the **.../<installation_directory>/scripts/tango.env** configuration file. The current default value is `600` seconds.

Correcting an 413 (Request Entity Too Large) NGINX Error

> **Important**
>
> The NGINX container was removed in release 9.0.015.03.

1. Open the **nginx.conf** file.
2. Increase the value for the `client_max_body_size` parameter to `3g`.
3. Restart NGINX by entering the following command:

```
$ docker restart nginx
```

(Optional) Change the Default Port for AICS

> **Important**
>
> If you change default port for AICS you have to make sure that port is opened to anyone who needs to access AICS APIs or UI.

If you are using NGINX, you must also change the port in your NGINX configuration. This section explains how to change the port for the Predictive Routing (Tango) application only.

> **Important**
> The NGINX container was removed in release 9.0.015.03.

Stop the application by running the following command:

```
$ bash stop.sh
```

Edit the **scripts/docker-compose.yml** file:

1. Locate the **tango-no-nginx** label.
2. Replace the listening port number with the new port number.

For example, to replace the default port, 443, by port 9090., replace `ports: "443:3031"` with `ports: "9090:3031"`.

Start the application by running the following command:. Note that you start the application without NGINX (there is no `-n` flag).

```
$ bash start.sh
```

Use your browser to check that the application is running on the new port (`9090`).

(Optional) Turn on SSL/ HTTPS on NGINX

> **Important**
> The NGINX container was removed in release 9.0.015.03.

To turn on SSL and HTTPS on NGINX, perform the following steps:

1. Stop the application using the following command:
   ```
   $ bash scripts/stop.sh
   ```
2. Create a certificate or add the certificate and key using a command in the following syntax:
   ```
   $ openssl req -newkey rsa:2048 -nodes -keyout server.key -x509 -days 365 -out
   server.crt openssl dhparam -dsaparam -out dhparams.pem 4096
   ```
3. Update the **docker-compose.yml** file using the following commands:
   ```
   nginx:
   image: nginx:1.11.9-alpine
   container_name: nginx
   restart: always
   ports:
      - 80:80
      - 443:443
   volumes:
      - ./nginx-ssl.conf:/etc/nginx/nginx.conf
   ```

```
    - ./server.crt:/etc/nginx/server.crt
    - ./server.key:/etc/nginx/server.key
    - ./dhparams.pem:/etc/nginx/dhparams.pem
```

4. Uncomment (remove the pound sign from) the entire second section of the **nginx.conf** file. This sections contains the SSL configuration.
5. To enable HTTPS on NGINX, replace the following line in the **nginx.conf** file:
   `proxy_set_header X-Forwarded-Proto $scheme;`
   with: `proxy_set_header X-Forwarded-Proto https;`
6. Restart AICS using the following command. This is required to make the changes take effect:
   `$ bash scripts/start.sh -n</source>`
7. Verify that you can access Predictive Routing via HTTPS by opening the following URL in your browser:
   `https://<SERVER_IP_ADDRESS>/`

(Optional) View Container Disk Usage

AICS uses persistent storage for two containers: Tango and Mongo. This storage configuration is defined in the `docker-compose.yml` file. Depending on your environment and its demands, you might need to change its configuration.

Disk usage might vary depending of the size of the organization and the use, but as a general rule, use a dedicated mount point at least for the **Mongo** container, because it is the fastest-growing directory.

Tango Container

- **Host directory: /opt/SP/jop/temp/Medallia**
- Container directory: **/data/medallia**
- Usage: Contains the outcome of the call based on a survey.

- **Host directory: /opt/SP/jop/temp/GIM**
- Container directory: **/data/gim**
- Usage: Genesys Info Mart interaction records: interactions in the contact center, name of the agent, and so on.

- **Host directory: /opt/SP/jop/temp/CRM**
- Container directory: **/data/crm**
- Usage: CRM contains caller and customer profile information, products they own, tenure, and so on.

Mongo Container

- **Host directory: /datadir**
- Container directory: **/data/db**
- Usage: Stores MongoDB data.

For extra information on Docker volumes, see Using Docker Volumes

(Optional) Back Up Your Data

Genesys recommends that you back up necessary data, especially MongoDB data.

- The Disk Usage section offers a general discussion of the directories. See the Mongo Container section in Disk Usage to determine which directories should be backed up (the **Host directory**, for example) .

> ### Important
> For extra information about MongoDB backups: Backing Up MongoDB.

(Optional) Map a Local Volume to a Container

You can map local directories or files into any of the containers used by the application: tango, workers, mongo, minio, or nginx.

To create the mapping follow these steps:

1. Update the **IP_JOP_PRR_<version_number>_ENU_linux/scripts/docker-compose.yml** file.
2. Edit the corresponding service section by adding a new line on the volumes declaration.
3. to make your changes take effect, stop and then restart the application, using the flags that may apply for starting the application.
   For example, to mount an existing local directory named `/some_local_directory` into the tango container at `/custom_mount_point` configure the volume would as follows:

```
volumes:
  - /opt/SP/jop/temp/Medallia:/data/medallia
  - /opt/SP/jop/temp/GIM:/data/gim
  - /opt/SP/jop/temp/CRM:/data/crm
  - /some_local_directory:/custom_mount_point
```

> ### Important
> - In releases earler than 9.0.015.03, the tango container can be started with or without NGINX. NGINX support was removed in release 9.0.015.03. It has two declaration options in the **docker-compose.yml** file: `tango` and `tango_no_nginx`. You should update both to avoid confusion.
> - Additional information can be found at https://docs.docker.com/compose/compose-file/compose-file-v2/#volumes

Install into an Existing AICS Deployment

It is easy to install a different version of AICS on your target server. You can use the steps here to install either a newer or an older version of AICS.

> **Important**
>
> There is downtime during this process but no data is lost. Executing this script only upgrades services and does not stop or upgrade MongoDB.

> **Important**
>
> Review the Upgrade Notes section of the Release Notes for all releases later than your starting release, including your target release. Follow any procedures specified for the interim releases, such as running scripts. If there is no Upgrade Notes section, or the section is empty, no additional steps are required for the associated release. The following AICS releases *do* require special upgrade procedures:
>
> - 9.0.007.00
> - 9.0.007.01
> - 9.0.007.03
> - 9.0.011.00
> - 9.0.013.01
> - 9.0.014.00
> - 9.0.014.02

To perform the upgrade:

1. If you have custom values for any environment variables, make a copy of the **tango.env** file before you start your upgrade. For more about the environment variables, see Set Values for Environment Variables, above.
2. Copy the **IP_JOptPlatform_<*version_number*>_ENU_dockerlinux.zip** file to the desired installation directory. Genesys recommends that you use the PR_USER home directory as destination for the AICS installation package.
3. Unzip the **IP_JOptPlatform_<*version_number*>_ENU_dockerlinux.zip** file, using the Linux `unzip` command, to access the **IP_JOP_PRR_<*version_number*>_ENU_linux.tar.gz** repository file.
4. To unpack the **IP_JOP_PRR_<*version_number*>_ENU_linux.tar.gz** repository file, execute the following command:

```
$ tar -xvzf IP_JOP_PRR_<version_number>_ENU_linux.tar.gz
```

After unpacking the new version of AICS in the PR_USER home directory, you will have multiple

different subdirectories named **IP_JOP_PRR_<*version_number*>_ENU_linux**. For example you might have two subdirectories:

- **IP_JOP_PRR_<*old_version_number*>_ENU_linux**
- **IP_JOP_PRR_<*new_version_number*>_ENU_linux**

1. Assuming you are installing *new_version* of the application and removing *old_version*, execute the following commands in the **IP_JOP_PRR_<*new_version_number*>_ENU_linux** directory:

   ```
   $ bash scripts/install.sh
   ```

   ```
   $ bash scripts/upgrade_gpr_services.sh
   ```

2. Configure any environment variables you require, using one of the following methods:
   - Paste the copy you made of your previous **tango.env** file over the new one.
   - To preserve any newly-added environment variables, open the new **tango.env** file and edit it.

Your updated version of AICS should now be ready for use.

(Optional) Uninstall AICS

The procedure given in this section should be used only on single-server deployments.

> **Important**
>
> If you need to remove AICS from an HA environment, contact Genesys Customer Care for assistance.

To entirely remove AICS, enter the following commands:

1. `$ bash IP_JOP_PRR_xyz/scripts/stop.sh # stop GPR`
2. `$ rm -rf IP_JOP_PRR_xyz # delete GPR installation`
3. `$ sudo docker system prune -a --volumes`

   or if this fails

   ```
   $ sudo docker system prune -a
   ```

4. `$ sudo rm -rf /datadir/*`

AICS should now be entirely removed from your environment.

Deploying in High Availability Environments

Both AI Core Services (AICS) and Agent State Connector (ASC) support high availability (HA).

High availability (HA) is configured differently for each Predictive Routing component:

- AI Core Services (AICS) uses a multi-server architecture. It can be installed at a single site, or in a multi-site architecture. Genesys recommends that you install AICS on three or five servers. More servers mean higher availability: with three servers, the system can survive the failure of only one machine; with five servers, the system can survive the failure of two machines.

> **Important**
>
> - AICS is installed in Docker containers. Genesys does not ship Docker as a part of AICS. You must install Docker in your environment before you can load the AICS containers.
> - You might need an active internet connection to download additional libraries when installing Docker.

- Agent State Connector (ASC) is deployed in warm-standby mode, with primary and backup servers.
- The URS Strategy Subroutines and Composer Subroutines components run as part of your routing solution, and therefore use the HA architecture established for that solution.

HA for AICS

The HA deployment and operating information for AICS is divided into the following sections:

- Hardware Requirements
- Installing HA AICS - Single Data Center Architecture
- Installing HA AICS - Multiple Data Center Architecture
- Set Values for Environment Variables
- Load Balancing and HTTPS Configuration for HA AICS
- Required Ports for AICS Servers
- Scaling the AICS Deployment (jumps to the Scaling AI Core Services topic in this *Guide*)
- Cleaning Up Disk Space
- Installing into an Existing HA AICS Deployment
- Checking the Logs for the AICS Containers (jumps to the Logging topic in this *Guide*)

- Troubleshooting an AICS HA Deployment
- (Optional) Backing Up Your Data
- (Optional) Installing AICS on a Kubernetes Cluster
- (Optional) Mapping a Local Volume to a Container

Hardware Requirements

- AICS HA requires a cluster of at least three servers. Genesys recommends that you deploy an odd number of servers to be used for hosting highly available AICS system.
- Every server must meet the preconditions stated in Target Server Requirements and Recommendations (in the AI Core Services Single-Host Deployment topic in this *Guide*). This will be verified during installation.
- All servers must have networking set up between them, with the ports opened as specified in Required Ports for AICS Servers.
- All servers must have synchronized system clocks. You can use Network Time Protocol (NTP) for this.
- On every target server, port 3031 must be reachable by the load balancer.
- On every target server, you **MUST** create a separate disk partition for storing MongoDB data. Mount this partition as `/datadir`. The partition size depends on your expected data usage, but must be at least 50 GB. For disk partitioning, use standard Linux tools. The `/datadir` partition **MUST** exist before you install GPR and the user who is executing the GPR installation should have write access to the partition. Preliminary Step: Create a Separate Disk for the MongoDB Database explains how to check the free space in your **mongodb** directory.
- You must have at least 50 GB free disk space on the root partition.

Important

If you are running VMWare VXLAN, you might encounter a port conflict between VMWare VXLAN and Docker, both of which require port 4789. If you encounter this issue, Genesys recommends that you use a networking application such as Weave Net to manage networking among Docker containers. For additional information, consult the documentation for the respective products:

- For the Docker Swarm port requirements: Use swarm mode routing mesh
- For VMWare VXLAN port requirements: Ports and Protocols Required by NSX
- For Weave Net: Introducing Weave Net

Installing HA AICS - Single Data Center Architecture

Important

- The following instructions enable you to set up a **new** AICS HA deployment in a single data center. If you already have a single-server deployment of AICS installed, contact Genesys Customer Care for help migrating to an HA architecture.
- If you need to uninstall AICS from an HA environment, contact Genesys Customer Care for assistance.

Installation Procedure

Important

- Some installation steps require you to know the hostnames of the target servers. You can run the command `hostname` on each server in the cluster to get the hostnames. This document uses the terms *node-1-hostname*, *node-2-hostname*, *node-3-hostname*, and so on, to stand in for the real hostnames of the servers. You must use actual hostnames when executing the example commands shown in the following sections.
- All scripts for installing and operating AICS in an HA setup are located in the **IP_JOP_PRR_<*version_number*>_ENU_linux/ha-scripts/** directory.

1. **Copy the installation binary file:**
   Copy the ***.tar.gz** file to every server in the cluster. Make sure you follow recommendations about the user PR_USER and the installation location described in single-host installation.
2. **Unpack the installation binary file:**

   Unpack the file on every server in the cluster. To unpack, follow these steps:

   1. Copy the **IP_JOP_PRR_<*version_number*>_ENU_linux.tar.gz** installation binary file to the desired installation directory. Genesys recommends that you use the PR_USER home directory as the destination for the AICS installation package.
   2. From a command prompt, unpack the file using the following command to create the `IP_JOP_PRR_<version_number>_ENU_linux` directory:

      ```
      tar -xvzf IP_JOP_PRR_<version_number>_ENU_linux.tar.gz
      ```

   3. **Create a Docker Swarm cluster.**
      AICS uses Docker Swarm technology to ensure high availability of all its components. In order for AICS to be deployed in highly available manner, you must properly format the Docker Swarm cluster on your target servers.

      1. On the target server with the hostname *node-1-hostname*, execute following command to

initiate the Docker Swarm cluster:

```
docker swarm init
```

> **Important**
>
> If the system has multiple IP addresses, specify the **--advertise-addr** parameter so the correct address is chosen for communication between all nodes in the cluster. If you do not specify this parameter, an error similar to the following is generated: `Error response from daemon: could not choose an IP address to advertise since this system has multiple addresses on different interfaces (10.33.181.18 on ens160 and 178.139.129.20 on ens192) - specify one with --advertise-addr`.

The following is an example of the command to initiate the Docker Swarm cluster, specifying the address that is advertised to other members of the cluster:

```
docker swarm init --advertise-addr YOUR_IP_ADDRESS
```

You can also specify a network interface to advertise the interface address, as in the following example:

```
docker swarm init --advertise-addr YOUR_NETWORK_INTERFACE
```

2. Still on the node with the hostname *node-1-hostname*, execute the following command:

   ```
   docker swarm join-token manager
   ```

   The output of this command should look similar to the following:

   ```
   docker swarm join --token
   SWMTKN-1-4d6wgar0nbghws5gx6j912zf2fdawpud42njjwwkso1rf9sy9y-
   dsbdfid1ilds081yyy30rof1t 172.31.18.159:2377
   ```

3. Copy this command and execute it on **all** other nodes in cluster. This ensures that all other nodes join the same cluster and coordinates AICS deployment.
4. Now execute following command on the node with the hostname *node-1-hostname* in order to verify that cluster has been properly formed and that you can continue with installation:

   ```
   docker node ls
   ```

   The output of this command **MUST** show you all target servers in the cluster (*node-1-hostname*, *node-2-hostname*, ..., *node-X-hostname*). If you do not see a complete list of servers, do not proceed with AICS installation. The following is an example of output where all nodes joined the cluster and are all reachable:

   ```
   ID              HOSTNAME  STATUS  AVAILABILITY  MANAGER  STATUS
   vdxn4uzuvaxly9i0je8g0bhps *node-1-hostname Ready Active Leader
   908bvibmyg9w87la6php11q96 node-2-hostname Ready Active Reachable
   ersak4msppm0ymgd2y7lbkgne node-3-hostname Ready Active Reachable
   shzyj970n5932h3z7pdvyvjes node-4-hostname Ready Active Reachable
   zjy3ltqsp3m5uekci7nr06tlj node-5-hostname Ready Active Reachable
   ```

5.

Label MongoDB nodes in the cluster:

Follow the steps below to define your MongoDB nodes:

1. Decide how many MongoDB instances to install in your deployment. You must install odd number of MongoDB instances, and no fewer than three. A higher number means higher availability.

   > **Important**
   > Only one MongoDB instance can run per target server.

2. On the server with the host name *node-1-hostname*, execute following command to see all the nodes currently in the cluster:

   ```
   docker node ls
   ```

3. Choose the servers where MongoDB instances will run. In single data center deployment it does not matter which servers you choose as long as they have fast disks (SSD) and enough disk space.
   The examples assume you chose the servers with the host names *node-1-hostname*, *node-2-hostname*, and *node-3-hostname* to run MongoDB instances.

4. Label the selected nodes appropriately. To do this, execute following commands on *node-1-hostname*:

   ```
   docker node update --label-add mongo.replica=1 $(docker node ls -q -f
   name=node-1-hostname)
   docker node update --label-add mongo.replica=2 $(docker node ls -q -f
   name=node-2-hostname)
   docker node update --label-add mongo.replica=3 $(docker node ls -q -f
   name=node-3-hostname)
   ```

   For a cluster with five MongoDB instances, you would also run these two additional commands (and you would have to have at least five servers in the cluster). Follow the established pattern to label additional nodes if you are using more than five.

   ```
   docker node update --label-add mongo.replica=4 $(docker node ls -q -f
   name=node-4-hostname)
   docker node update --label-add mongo.replica=5 $(docker node ls -q -f
   name=node-5-hostname)
   ```

6. **Label the Worker nodes in the cluster:**

   Decide how many workers you want to run and on which servers.

   - The minimum number of servers marked to run worker instances is two, but you can have more for increased scalability and high availability. This configuration is verified during AICS installation.
   - Each worker container scales independently and you can have multiple instances of the same worker type running on the same server.
   - Workers can be co-located with other containers (such as MongoDB).

   1. Execute following commands on the node with the hostname *node-1-hostname* to ensure that worker instances will run on nodes *node-1-hostname*, *node-2-hostname*, and *node-3-hostname*:

      ```
      docker node update --label-add worker=true $(docker node ls -q -f
      ```

```
name=node-1-hostname)
docker node update --label-add worker=true $(docker node ls -q -f
name=node-2-hostname)
docker node update --label-add worker=true $(docker node ls -q -f
name=node-3-hostname)
```

You can choose to label more nodes and make them available to run worker instances. You cannot label fewer than two nodes with `worker = true`

7. **Label the MinIO nodes in the cluster:**
   You should always label at least one node to run MinIO container. MinIO container is used for faster dataset uploads. We recommend to label two nodes to run MinIO.

   1. Execute following commands on the node with the hostname *node-1-hostname* to ensure that a MinIO instance will run on one of nodes *node-1-hostname* or *node-2-hostname*:

      ```
      docker node update --label-add minio=true $(docker node ls -q -f
      name=node-1-hostname)
      docker node update --label-add minio=true $(docker node ls -q -f
      name=node-2-hostname)
      ```

      - There is always only one MinIO instance running and it only runs on one of the properly-labeled nodes.

   2. Find on what node MinIO container is running by executing following command:

      ```
      docker service ps minio_server_minio --format {{.Node}}
      ```

   3. Find the public IP address of the MinIO node and make sure that the S3_ENDPOINT configuration parameter in **IP_JOP_PRR_<*version_number*>_ENU_linux/conf/tango.env** is configured in the following way:

      ```
      S3_ENDPOINT=https://PUBLIC_IP_OF_NODE_WHERE_MINIO_CONTAINER_RUNS:9000
      ```

      - The MinIO container can be co-located with other containers (such as MongoDB or workers).

8. **Note the Tango instances:**
   There is automatically one Tango instance running on every node (server) in the cluster. As you expand the cluster, new Tango instances are installed and started on the newly-created nodes.

9. **Install AICS in HA mode:**
   Your Docker Swarm cluster is now ready for AICS installation.

   1. To make the Docker images needed by AICS available on every server in the cluster, execute the following command on **every** server in the cluster:

      ```
      bash ha-scripts/install.sh
      ```

      If you are managing your MongoDB deployment externally, run the **install.sh** script with the **-externalMongo** flag, as follows:

      ```
      bash ha-scripts/install.sh -externalMongo
      ```

   2. To initialize the HA AICS deployment and start the application, execute the following command. This command also sets the password for your default user, super_user@genesys.com. Replace the variable `<'my_password'>` in the command below with a strong password, and record it securely for future reference.

```
cd ha-scripts; bash start.sh -l -p <'my_password'>
```

10. **Access AICS in HA mode:**
    Once your fully-installed AICS deployment has started up correctly, you can access AICS by using the IP address of any server in the cluster on port 3031 as: https://<IP_ADDRESS>:3031

> **Important**
>
> Genesys recommends that you install a load balancer in front of the cluster to make it easier to access AICS. See Load Balancing for HA AICS for details.

Installing HA AICS - Multiple Data Center Architecture

> **Important**
>
> The following instructions enable you to set up a **new** AICS HA deployment in a multiple data center environment. If you already have a single-server deployment of AICS installed, contact Genesys Customer Care for help migrating to an HA architecture.

The basic procedure for installing AICS in multiple data centers is the same as installing AICS in single data center. However, when deploying AICS in an environment with multiple data centers, there are some considerations and requirements in addition to those for a single data center.

- Before starting, ensure that you have a fast LAN/WAN that connects all of the servers and that all ports are open.
- Plan to spread all instances of the AICS components (Workers, MongoDB, Tango, MinIO) across your data centers to ensure that AICS continues to operate correctly if a single data center fails. ***This is most important for servers running MongoDB.***

Special Considerations for MongoDB Instances

- Spread labels across the data centers when labeling servers to run MongoDB replica set members.

> **Important**
>
> The AICS installation procedure does **not** validate whether MongoDB instances are spread across data centers. Failing to ensure this even distribution can compromise overall availability of the AICS deployment.

- Every data center should have similar hardware capacity (RAM, CPU, disk).

- **No data center should have a majority of the MongoDB servers running in it when using three data centers.**

Using Only Two Data Centers

You can use only two data centers when installing AICS in HA mode, but this reduces overall availability of AICS. In this scenario, one data center always has the majority of the MongoDB servers running in it. If that data center fails, the second data center goes into read-only mode. You must then execute a manual recovery action, using the following procedure:

Execute Manual Recovery

To recover if your system enters read-only mode:

1. Find the current status of the MongoDB cluster by entering the following command:

   ```
   docker exec -it $(docker ps -qf label=com.docker.swarm.service.name=mongo_mongo3)
   mongo --ssl --sslCAFile /etc/ssl/mongodb.pem --sslAllowInvalidHostnames --eval "for
   (i=0; i<rs.status().members.length; i++) { member = rs.status().members[i];
   print(member.name + \" : \" + member.stateStr) }"
   ```

   For example, you might enter:

   ```
   [pm@hostname ha-scripts]$ docker exec -it $(docker ps -qf
   label=com.docker.swarm.service.name=mongo_mongo3) mongo --ssl --sslCAFile /etc/ssl/
   mongodb.pem --sslAllowInvalidHostnames --eval "for (i=0;
   i<rs.status().members.length; i++)="" {="" member="rs.status().members[i];"
   print(member.name="" +="" \"="" :="" member.statestr)="" }"<="" tt="">
   ```

   And receive back the following:

   ```
   MongoDB shell version: 3.2.18

   connecting to: test

   mongo_mongo1:27017 : SECONDARY

   mongo_mongo2:27017 : SECONDARY

   mongo_mongo3:27017 : PRIMARY

   [pm@node-3 ha-scripts]$
   ```

   The primary MongoDB node is `mongo_mongo3`. The following command shows the number of members in the MongoDB cluster:

   ```
   rs.status().members.length;
   ```

2. Remove any unreachable MongoDB members. If necessary, use the following command to change to the primary node:

   ```
   com.docker.swarm.service.name=mongo_mongo3
   ```

3. Run the following command on the primary MongoDB node to recover the MongoDB cluster:

   ```
   docker exec -it $(docker ps -qf label=com.docker.swarm.service.name=mongo_mongo3)
   mongo --ssl --sslCAFile /etc/ssl/mongodb.pem --sslAllowInvalidHostnames --eval
   "members = rs.status().members; cfgmembers = rs.conf().members; for
   (i=members.length; i>0; i--) { j = i - 1; if (members[j].health == 0) {
   ```

```
cfgmembers.splice(j,1) } }; cfg = rs.conf(); cfg.members = cfgmembers;
printjson(rs.reconfig(cfg, {force: 1}))"
```

For example, you might enter:

```
[pm@hostname ha-scripts]$ docker exec -it $(docker ps -qf
label=com.docker.swarm.service.name=mongo_mongo3) mongo --ssl --sslCAFile /etc/ssl/
mongodb.pem --sslAllowInvalidHostnames --eval "members = rs.status().members;
cfgmembers = rs.conf().members; for (i=members.length; i>0; i--) { j = i - 1; if
(members[j].health == 0) { cfgmembers.splice(j,1) } }; cfg = rs.conf(); cfg.members
= cfgmembers; printjson(rs.reconfig(cfg, {force: 1}))"
```

And receive back the following:

```
MongoDB shell version: 3.2.18

connecting to: test

{ "ok" : 1 }

[pm@node-3 ha-scripts]$
```

The minority members in the reachable data center can now form a quorum, which returns the running data center to read-write mode.

For other useful commands, including commands for checking node status and removing non-functional nodes, see Troubleshooting Your HA AICS Deployment, below.

Set Values for Environment Variables

This section lists environment variables that must or should be configured for optimal GPR performance, and the recommended values. Adjust these values as necessary, based on your specific environment.

> ### Warning
> The **tango.env** file, which contains the environment variables, is overwritten when you perform a software upgrade. Before upgrading, save a copy of the **tango.env** file and refer to it to reset your variables. Note that if you simply overwrite the new **tango.env** file with your existing one, any environment variables added in the new release are removed.

Environment variables are defined in the **IP_JOP_PRR_<*version_number*>_ENU_linux/conf/tango.env** file. The same file is used for both single node and HA deployments.

To add a new variable:

1. Create a new line in the **tango.env** file.
2. Add the variable and its value, using the following format:
   <NEW_ENV_VAR>=value

> **Important**
>
> - Do not use quotes for string parameters.
> - Remove trailing spaces.

Changes take effect on restart of the tango container (run the `bash scripts/restart.sh command` command). In an HA environment, with multiple instances of the containers running, restart is performed sequentially (a rolling restart), so that there is no downtime of the GPR application.

Configurable Environment Variables

- **ADD_CARDINALITIES_EVERY_N_RECORDS** - When you append data to an Agent or Customer Profile via the API, cardinalities are computed only for the appended data portion and only when the number of agents or customers set in the ADD_CARDINALITIES_EVERY_N_RECORDS parameter is reached. The results of computation are added to the already-stored cardinality values. This significantly improves speed when loading new data by avoiding simultaneous recomputations on the full data collection when there are multiple frequent appends done in small batches.enables you to specify how many appended records are added to an Agent or Customer Profile before GPR recalculates cardinalities. The default value is `1000`.
  - Notes:
    - This functionality is available only when you use the Predictive Routing API. If you append using the Predictive Routing application interface, all cardinalities are recalculated.
    - Full automatic computation happens only once, when an Agent or Customer Profile is uploaded the first time for schema discovery.
    - You can force recomputation of cardinalities on the full Agent or Customer Profiles collection using the POST **compute_cardinalities** API endpoint. For details, see the *Predictive Routing API Reference*. (This file requires a password to open it. Contact your Genesys representative if you need access.)
- **AUTOGENERATE_INDEXES** - Instructs GPR to create indexes on all Datasets, Agent Profile schemas, and Customer Profile schemas. By default, set to `True`.

  > **Important**
  >
  > Genesys strongly recommends you to leave the default value for this variable.

- **HOST_DOMAIN** - Use this variable to specify the public IP address or host name used for your deployment. The value should be one of the following, depending on your environment type:
  - For single-server deployments, specify the public IP address or the host name of the host where GPR is deployed.
  - For high availability (HA) deployments, specify the IP address of your load balancer.
- **LOG_LEVEL**
  - **INFO** - Informational messages that highlight the progress of the application: **LOG_LEVEL**=`INFO`.

This setting is recommended for production deployments.

  - **DEBUG** - Fine-grained informational events that are most useful to debug the application: **LOG_LEVEL**=DEBUG. This setting should be used only for short periods of time because it can fill the disk.

- **LOGIN_MESSAGES** enables you have the Predictive Routing application display a custom message on the login screen.
  - When you enter this message, make sure that all special characters are properly escaped. *Special characters* are ones not part of the standard English alphabet, such as symbols, letters with umlauts, cedillas, and other such marks, and letters from other alphabets, such as the Greek or Cyrillic alphabets.
  - To simplify the task of converting characters, Genesys recommends an online conversion tool, such as **https://www.freeformatter.com/html-escape.html**.
  - For example, make the following substitutions:
    - & becomes &
    - < becomes <
    - > becomes >
    - " becomes "
    - ' becomes '
- **OMP_NUM_THREADS** (required for releases prior to 9.0.011.00; in releases 9.0.011.00 and higher, this parameter is set automatically)
  - Genesys recommends that you set the value to OMP_NUM_THREADS=1 for the best performance.
  - If you do not specify a value, GPR spawns one thread for each core it detects in your environment. The system assumes it can use all available cores for tasks such as analysis and model training, leaving no CPU resources for other processes running on the same machine, such as reading/writing to the database. The result is an overall slowdown of the application. Set this variable to allow the operating system to properly distribute CPU threads among the various running processes.
- **S3_ENDPOINT** - (Mandatory) The endpoint for the Minio container, introduced in AICS release 9.0.013.01 for Dataset uploads and expanded to Agent and Customer Profile uploads in AICS 9.0.015.03. Specifies the public IP address or domain name of the server where AI Core Services is installed, followed, optionally, by the port number.
  - The port number must always be 9000, which is the mandatory port value allocated for the Minio container.
  - In HA environments, locate the server on which the Minio container is running and use the public IP address or the domain name of that server. For example:

    For an IP address - `S3_ENDPOINT=https://<public_ip_address>:9000`

    For a domain name - `S3_ENDPOINT=https://<your_domain_name>:9000`
  - The S3_ENDPOINT value must always use the HTTPS protocol. If you do not configure this variable, the **start.sh** script generates a warning message and stops deploying AICS.
- (Optional) **GUNICORN_TIMEOUT**
  - Adjust the timeout if you need to accommodate a large Dataset. The current default value is `600` seconds.

Load Balancing and HTTPS Configuration for HA AICS

Once AICS has been installed and started, you can access it using the IP address of any node in the cluster on port 3031. To enable load balancing:

1. Your load balancer should have its health-check functionality turned on.
2. The load balancer should check for HTTP code `200` to be returned on `https://IP:3031/login`.

> **Important**
>
> - Genesys recommends a third-party highly-available load balancer, such as F5, to ensure all requests to AICS platform are spread evenly across all nodes in the AICS cluster.
> - If you need SSL, set it up on the third-party load balancer.
> - If you are using a domain name instead of a numeric IP address, configure the S3_ENDPOINT environment variable in the **tango.env** file as follows: **S3_ENDPOINT**= `https://<your_domain_name>:9000`

Configure HTTPS in an HA Environment

In a high availability (HA) environment, follow the instructions provided in the documentation for your load balancer. In an HA environment, you do not need to deploy the certificates on the individual nodes.

Unusual HTTP/S Deployment Scenarios

- Default local certificate - Genesys ships AICS with a default local certificate. You can use that local certificate to access the GPR web application for internal testing purposes. Note the following points when using the local certificate:
  - The browser displays a Not Secure connection warning.
  - You **cannot** use the default certificate to configure connections from Agent State Connector or the Subroutines components to AICS.
- Self-signed certificate - *For lab environments only* - You can use OpenSSL to generate a self-signed certificate. You can use this self-signed certificate to configure secure connection between AICS and the other GPR components, as explained in the instructions for configuring HTTPS for ASC and the URS Strategy Subroutines. Generate a self-signed certificate by executing a command following the format in the following example:

```
$ openssl req -new -newkey rsa:4096 -days 365 -nodes -x509 -subj "/C=US/ST=US/L=US/
O=IT/OU=IT Department/CN=<ip address of the server where GPR is deployed>" -keyout
tango.key -out tango.crt
```

- HTTP connections in test environments - In AICS release 9.0.015.04 and higher, you can optionally configure HTTP connections.

> **Warning**
>
> HTTP connections are supported only in test environments. Genesys strongly recommends using the default HTTPS configuration in production environments and in lab environments that contain sensitive data. Genesys is not responsible for any potential damage and/or data loss if the solution is implemented without the recommended security practices and protocols.

To use HTTP connections, perform the following steps:

1. Comment out the following lines in the **ha-scripts/swarm/tango-swarm.yml** file *on every node*:

   ```
   #       - ../../conf/tango.key:/data/ssl/tango.key
   #       - ../../conf/tango.crt:/data/ssl/tango.crt
   ```

2. Save your changes.
3. Restart GPR by running the following command on any node:

   ```
   $ bash ha-scripts/restart.sh
   ```

   You do not need to restart each node separately. Running the restart command on one node restarts the entire system.

4. On the load balancer in front of your Docker Swarm cluster, change *https* to *http* in your load balancer configuration. For example, make a change similar to the following:
   Change

   https://your_node_ip:3031

   to

   http://your_node_ip:3031

Using the NGINX Load Balancer

> **Important**
>
> The NGINX container was removed from AICS in release 9.0.013.01.

In releases through 9.0.012.01, Genesys shipped the NGINX load balancer as part of AICS. *It is intended for use only in prototype scenarios.*

> **Important**
>
> The NGINX load balancer is a single point of failure and should not be used in production deployments.

To use the NGINX, follow the procedure below:

1. Edit the `ha-scripts/nginx/nginx.conf` file by putting the IP addresses of all nodes in your cluster into the `upstream tango` section using syntax such as IP1:3031, IP2:3031, IP3:3031. For example, your command might look similar to the following:

   ```
   upstream tango {
        server 18.220.11.120:3031;
        server 18.216.235.201:3031;
        server 13.59.93.192:3031;
        }
   ```

2. Execute the following command in order to start the NGINX container:

   ```
   bash ha-scripts/nginx/start.sh
   ```

3. Verify that you can access AICS by pointing your browser to IP address where NGINX is running.

To stop NGINX, run the following command:

```
bash ha-scripts/nginx/stop.sh
```

To fix a `413 (Request Entity Too Large)` NGINX error, follow these steps:

1. Open the **nginx.conf** file.
2. Increase the value for the **client_max_body_size** parameter to 3g.
3. Restart NGINX using the command:

```
docker restart nginx
```

Required Ports for AICS Servers

The following ports are those required for communication between all target servers in the cluster. Note that some ports are specific to high availability (HA) environments (such as the Docker swarm port), while others apply to all deployments.

Component	Protocol	Port Number	Type	Description
Docker	TCP	2377	Inbound/Outbound	Cluster management communications
Docker swarm	TCP/UDP	7946	Inbound/Outbound	Required for Docker Swarm for communication among nodes
Docker swarm	UDP	4789	Outbound/Inbound	For overlay network traffic
MongoDB	TCP	27017	Inbound/Outbound	Default port for MongoDB
Tango container	TCP	3031	Inbound	Required to access the Predictive Routing API and Predictive Routing

Component	Protocol	Port Number	Type	Description
				web application
MinIO container	TCP	32646	Inbound/Outbound	Default port for MinIO
SSH	TCP	22	Inbound/Outbound	Required to access all target servers using SSH

To open a port, use the following syntax:

```
firewall-cmd --zone=public --add-port=<''port_number''>/<''protocol''> --permanent
```

Important

If you are running VMWare VXLAN, you might encounter a port conflict between VMWare VXLAN and Docker, both of which require port 4789. If you encounter this issue, Genesys recommends that you use a networking application such as Weave Net to manage networking among Docker containers. For additional information, consult the documentation for the respective products:

- For the Docker Swarm port requirements: Use swarm mode routing mesh
- For VMWare VXLAN port requirements: Ports and Protocols Required by NSX
- For Weave Net: Introducing Weave Net

Clean Up Disk Space

Starting in release 9.0.013.01, GPR performs automatic cleanup processes which should maintain an adequate amount of free disk space. However, if you are running an earlier version of AICS, or are running 9.0.013.01 or higher and continue to encounter disk space problems, refer to the instructions in this section.

You might encounter performance issues if you do not clean up Docker data that is no longer required. The Docker `prune` command enables you to clean up your Docker environment. The Docker user documentation provides a detailed discussion of the `prune` command and how to use it to clean up images, containers, and volumes; see Prune unused Docker objects.

Important

The clean-up process does not affect normal GPR operation. It does not require downtime, there is no need to restart any component, and performance is unaffected.

Clean-Up Procedure

Genesys recommends that you use the following commands to remove unnecessary Docker data:

```
docker container prune -f
docker volume prune -f
docker network prune -f
```

To schedule regular cleanup jobs, use the `crontab` functionality to execute the appropriate command on every server where GPR is installed. The following example schedules the cleanup job for every Saturday at 1:00 am:

```
echo "0 1 * * Sat (docker container prune -f; docker volume prune -f; docker network prune
-f)" ) | crontab -
```

In an HA environment, Genesys recommends that you perform the cleanup on each node in turn.

If you need to configure your logging settings to avoid unacceptable log file sizes, see the following information:

- The LOG_LEVEL environment variable
- Configure AICS Log Settings

Installing into an Existing HA AICS Deployment

It is easy to install a different version of AICS on your target servers. Use the standard procedure described below to install either a newer or an older version of AICS. For some releases, listed below, you must also run additional scripts to complete the upgrade.

Special Upgrade Procedures

Some releases require special upgrade scripts or procedures. These procedures appear in the Upgrade Notes section of the RN for that release.

- Review the Upgrade Notes section of the Release Notes for *all* releases later than your starting release, including your target release.
- If special upgrade scripts are required for any releases between your current version and your target version, they are all included in the IP for your target version.
- Follow any procedures specified for the interim releases, such as running scripts.
- If there is no Upgrade Notes section, or the section is empty, no additional steps are required for the associated release.
- The following AICS releases *do* require special upgrade procedures:
  - 9.0.007.00
  - 9.0.007.01
  - 9.0.007.03

- 9.0.011.00
- 9.0.013.01
- 9.0.014.00
- 9.0.014.02

Standard Upgrade Procedure

Follow the steps in this section to perform the standard upgrade:

> **Important**
>
> The standard process described below requires downtime, but does not result in loss of data. The upgrade script updates only the various services running in the Tango container and the Workers containers. It does not stop or upgrade MongoDB.

1. If you edited your **tango.env** file, save a copy of it in a separate location. Also save any other files you might have customized, such as the **docker-compose.yml** file and the contents of your **/datadir** directory.
2. Copy the new AICS release package (the ***.tar.gz** file) to all servers in the cluster. Use the same user and procedure as if you are installing AICS for the first time. All the recommendations about the user who performs the installation and operates AICS still apply.
3. After unpacking the new version of AICS in the PR_USER home directory that contains *all* target servers, you will have multiple different subdirectories named **IP_JOP_PRR_<*version_number*>_ENU_linux**. For example you might have two subdirectories:
   - **IP_JOP_PRR_<*old_version_number*>_ENU_linux**
   - **IP_JOP_PRR_<*new_version_number*>_ENU_linux**
4. Assuming you are installing *new_version* of the application and removing *old_version*, execute the following command in the **IP_JOP_PRR_<*new_version_number*>_ENU_linux** directory on *all* target servers:

   ```
   bash ha-scripts/install.sh
   ```

5. Then in any one of the servers, execute the following command in the **IP_JOP_PRR_<*new_version_number*>_ENU_linux** directory:

   ```
   bash ha-scripts/upgrade_gpr_services.sh
   ```

   This command executes the upgrade of Tango (AICS) on all nodes in the cluster, one by one, and rolls back the change if there is a problem. There is no downtime during this upgrade, and no data loss.
6. To restore your custom environment values, paste the copy you made of your previous **tango.env** file over the new one, as well as any other files you might have customized, such as the **docker-compose.yml** file and the contents of your **/datadir** directory.

Your upgrade should now be complete.

Troubleshooting a AICS HA Deployment

The following sections offer information that can help you identify issues without your deployment.

Handling Server Failure

If a server (node) restarts, the HA deployment recovers automatically as long as the server keeps its previous IP address and the data on the disk is not corrupted.

The following command identifies a non-working node as **unreachable** node:

```
docker node ls
```

If a server needs to be decommissioned and replaced with new one, the following manual step is necessary to preserve the health of the cluster. After shutting down the server that is to be decommissioned, execute the following two commands, where **NODE_ID** is the unique node identifier of the server to be decommissioned:

```
docker node demote <NODE_ID>
docker node rm <NODE_ID>
```

After this, you can add a new server to your environment. Label it the same way as the decommissioned server and execute the procedure for joining that server to the cluster as described in Installation Procedure, above.

Handling Failover

When a server hosting MongoDB and the AICS application (the Tango container) experiences a failover, a certain number of API requests to AICS might fail during the few seconds it takes for the system to recover. The routing strategy attempts to resend any failed request, but Agent State Connector (ASC) does not have this capability. As a result, there is a risk of a small data loss.

Note that error messages appear in the logs for both MongoDB and the AICS application when a failover occurs.

Health Checks for Your Deployment

To check the health of your Predictive Routing HA deployment, perform the following steps:

1. Verify that all nodes are up and running. On any node in the cluster, execute the following command:

   ```
   docker node ls
   ```

   You should receive output similar to the following:

   ```
   [pm@hostname ~]$ docker node ls
   ID HOSTNAME STATUS AVAILABILITY MANAGER STATUS
   mc0bgyueb3c0h9drsy3j0i2ty node-1-hostname Ready Active Leader
   ```

```
vm1csljly66vwguxzaz8ly98r *node-2-hostname Ready Active Reachable

z2vlnldcyh0y57jwns0bz9jxe node-3-hostname Ready Active Reachable
```

All nodes should be reachable.

2. Check that all services are running by executing the following command on any node in the cluster:

```
docker service ls
```

You should receive output similar to the following:

```
[pm@hostname ~]$ docker service ls

ID NAME MODE REPLICAS IMAGE PORTS

jzjitn8lp78t mongo_mongo1 replicated 1/1 mongo:3.2

iqntp5eabfnw mongo_mongo2 replicated 1/1 mongo:3.2

whw05twosi9s mongo_mongo3 replicated 1/1 mongo:3.2

1jp3sgt16czw tango_tango global 3/3 jop_tango:2017_12_12_15_17

hu3kvkzxn88r workers_workers replicated 2/2 jop_tango:2017_12_12_15_17
```

- The important column here is REPLICAS.
- The Tango service should always be global and reachable on port 3031 on every node in cluster.
- The MongoDB service is replicated. The value 1/1 in the REPLICAS column for MongoDB indicates that a replica exists for each instance. See Checking the Health of MongoDB (below) for how to check health of MongoDB database.
- The Workers service is replicated and should show as many replicas as there are nodes labeled with the Workers label. See Label the Worker Nodes in the Cluster (above) for how to label nodes.

Checking the Health of MongoDB

All the commands listed below should show your MongoDB cluster with one PRIMARY instance and all other instances should be healthy SECONDARY instances.

- To check the health of the MongoDB cluster while logged into node with hostname *node-1-hostname* execute following command on *node-1-hostname*:

```
[pm@node-1-hostname ~]$ docker exec -it $(docker ps -qf
label=com.docker.swarm.service.name=mongo_mongo1) mongo --ssl --sslCAFile /etc/ssl/
mongodb.pem --sslAllowInvalidHostnames --eval 'rs.status()'
```

- To check the health of MongoDB cluster while logged into node with hostname *node-2-hostname* execute following command on *node-2-hostname*:

```
[pm@node-2-hostname ~]$ docker exec -it $(docker ps -qf
```

```
label=com.docker.swarm.service.name=mongo_mongo2) mongo --ssl --sslCAFile /etc/ssl/
mongodb.pem --sslAllowInvalidHostnames --eval 'rs.status()'
```

- To check the health of MongoDB cluster while logged into node with hostname *node-3-hostname* execute following command on *node-3-hostname*:

  ```
  [pm@node-3-hostname ~]$ docker exec -it $(docker ps -qf
  label=com.docker.swarm.service.name=mongo_mongo3) mongo --ssl --sslCAFile /etc/ssl/
  mongodb.pem --sslAllowInvalidHostnames --eval 'rs.status()'
  ```

  Similarly, you can check the health of MongoDB cluster from any other node where a MongoDB replica is running.

- For example, if you send the following commands:

  ```
  docker exec -it $(docker ps -qf name=mongo_mongo) mongo --ssl --sslCAFile /etc/ssl/
  mongodb.pem --sslAllowInvalidHostnames --eval "rs.status()" | grep "stateStr"
  ```

  You should receive a response similar to the following:

  ```
  "stateStr" : "SECONDARY",
  "stateStr" : "SECONDARY",
  "stateStr" : "PRIMARY",
  "stateStr" : "SECONDARY",
  "stateStr" : "SECONDARY",
  [pm@node-1 IP_JOP_PRR_gpr_rc_ENU_linux]$
  ```

Other Useful Commands

Here are few more useful commands to troubleshoot MongoDB:

To find out the status of all members in the replica set, use the following command:

```
docker exec -it $(docker ps -qf label=com.docker.swarm.service.name=mongo_mongo3) mongo --ssl
--sslCAFile /etc/ssl/mongodb.pem --sslAllowInvalidHostnames --eval "rs.status().members"
```

To remove an unreachable member, execute the following command (this has to be repeated for each unreachable member in a failed data center):

```
docker exec -it $(docker ps -qf label=com.docker.swarm.service.name=mongo_mongo3) mongo --ssl
--sslCAFile /etc/ssl/mongodb.pem --sslAllowInvalidHostnames --eval 'rs.remove("HOST:PORT")'
```

(Optional) Backing Up Your Data

This section applies specifically to backing up and restoring in an HA environment. For instructions to back up and restore MongoDB in a single-site/single-server AICS deployment, see Backing Up and Restoring Your Data.

Although HA greatly reduces the likelihood of data loss, Genesys recommends that you back up your data to safeguard it. This section explains how to back up and restore your data in an HA environment.

> **Important**
>
> All MongoDB backup and restore operations should be performed on the PRIMARY MongoDB instance.

Using SSL with MongoDB

The procedure below is for MongoDB with SSL enabled. *Genesys recommends that you use SSL.*

- To use SSL, add the `--ssl` parameter to your commands.
  In test environments, you can optionally add `--sslAllowInvalidCertificates` following the `--ssl` parameter.

  In test environments ONLY, if you need to maintain an environment without SSL connections, omit the `--ssl` and `--sslAllowInvalidCertificates` parameters.

Backing Up

On every server where MongoDB is running, there is one important directory:

- The `/data/db` directory in every MongoDB container is mapped to the `/datadir` directory on the server file system.

Use the `mongodump` command from inside the container to back up your MongoDB data, using the following command:

```
mongodump --ssl --out /data/db/`date +"%m-%d-%Y"`
```

This command backs up all databases in the `/data/db/<date +"%m-%d-%Y">` directory located in the container. For example, you might back up the `/data/db/08-18-2019` directory.

The backed-up data is located in the `/datadir/<date +"%m-%d-%Y">` directory on the server host computer. For the example backup command above, the output would be located in the `/datadir/08-18-2019` directory.

Restoring

In order to restore data you must first make data files available in the appropriate directory on the server host computer.

Use the following command inside of the container:

```
mongorestore --ssl --drop /data/db/''PATH_TO_SPECIFIC_BACKUP_DIRECTORY''
```

For example, you might run the command:

```
mongorestore --ssl --drop /data/db/08-18-2019
```

For extra information about backing up MongoDB and data preservation strategies, see the following topic on the MongoDB site: https://docs.mongodb.com/manual/core/backups/.

(Optional) Installing AICS on a Kubernetes Cluster

The following instructions provide optional deployment and configuration procedures that guide you through setting up AI Core Services (AICS) on Kubernetes. These instructions assume that you have already installed Kubernetes in your environment. Genesys supplies scripts to orchestrate AICS on the Kubernetes cluster.

For information about Kubernetes and how to deploy it, see the Kubernetes web site.

System and Architecture Requirements

In addition to the standard set of system requirements, the following apply specifically to AICS running on a Kubernetes cluster:

- Kubernetes version 1.10 or higher.
- Helm installed on the master node of the Kubernetes cluster.
- At least four nodes comprising the cluster, with the following roles:
  - 1 *master* node with minimum of 4 CPUs and 8 GB RAM.
  - 3 *worker* nodes each with a minimum of 8 CPUs and 16 GB RAM.
- AICS has specific CPU and RAM requirements depending on your environment. Use the Sizing Worksheet to determine what hardware resources you need for your environment. When running AICS on a Kubernetes cluster, keep in mind that Kubernetes also has CPU and RAM requirements.
- Kubernetes configured to use either local storage or Dynamic Volume Provisioning on all nodes, including the *master* node.

Installing AICS on the Kubernetes Cluster

To install AICS on Kubernetes, perform the following procedures:

1. Guided by the instructions in Installation Procedure (above), upload and unpack the **IP_JOP_PRR_<*version_number*>_ENU_linux.tar.gz** installation binary file to all Kubernetes nodes. When you unpack this file, it creates the **IP_JOP_PRR_<*version_number*>_ENU_linux** directory.
2. On the host that is the master node of the Kubernetes cluster, navigate to the **IP_JOP_PRR_<*version_number*>_ENU_linux** directory.
3. Locate and modify the following two files to include the version of AICS you are deploying. Do this by replacing `jop_tango:CHANGE_ME` in each of the two files with `jop_tango:$VERSION`, which can be found in the **jop.version** file in the same folder.
   - **helm/gpr/values.yml**
   - **helm/worker/values.yml**

     The AICS component was previously known as JOP. These files retain the old naming convention.
4. On each Kubernetes *worker* node (these are the Kubernetes *worker* nodes, not to be confused with AICS worker containers), navigate to the **IP_JOP_PRR_<*version_number*>_ENU_linux** directory and then run the following command to load the AICS Docker images:

```
bash ha-scripts/install.sh -s
```

Labeling Nodes

Run the followng commands from the Kubernetes cluster *master* node.

1. Check the existing labels on Kubernetes nodes:

   ```
   sudo kubectl get nodes --show-labels
   ```

2. For every node where you want an AICS worker container to run, execute:

   ```
   sudo kubectl label nodes <node-name> worker=true
   ```

3. For every node where you want a Tango container to run, execute:

   ```
   sudo kubectl label nodes <node-name> gpr-apps=true
   ```

Deploying AICS Using Helm with Local Storage

To use local storage for the MinIO and MongoDB data store, execute the following script on master node to deploy AICS using Helm charts. This deploys the Tango container and the various AICS worker containers:

```
bash kubernetes/deploy-gpr-services.sh
```

Deploying AICS using Helm with Dynamic Volume Provisioning

The following optional command can be used to provide details for the storage class and the type of provisioner. When you run the command, replace <provisioner_name> with an actual name of a supported provisioner, such as glusterfs.

```
bash kubernetes/deploy-gpr-services.sh -s <provisioner_name> -m mongodb-ssd -c minio-ssd
```

This ensures that the appropriate storage class is selected when making a persistent volume claim.

Upgrading

Follow the steps below to upgrade the Tango container and the Workers containers.

Upgrading the Tango Container Using Helm

1. Download the new AICS IP, unpack it, and navigate to the **IP_JOP_PRR_<*version_number*>_ENU_linux** directory for the *new* IP.
2. Manually modify the following two files to include the version of AICS you are deploying. Do this by replacing `jop_tango:CHANGE_ME` in each of the two files with `jop_tango:$VERSION`, which can be found in the **jop.version** file in the folder for the *new* IP.
   - **helm/gpr/values.yml**
   - **helm/worker/values.yml**
3. On each Kubernetes *master* and *worker* node, navigate to the **IP_JOP_PRR_<*version_number*>_ENU_linux** directory for the *new* version and then run the following command to load the new AICS Docker images:

```
bash ha-scripts/install.sh -s
```

4. On the master node, run the following command:

```
helm upgrade --install gpr -f ./helm/gpr/values.yaml ./helm/gpr
```

- To check the upgrade history of the AICS Tango container, run the following command:

```
helm history gpr
```

- To roll back the upgrade of the AICS Tango container, run the following command:

```
helm rollback gpr <revision_number>
```

Upgrading AICS Worker Containers Using Helm

To upgrade the Worker containers, run the following commands:

```
helm upgrade --install worker-analysis --set workerDeploymentName=worker-
analysis,workerTopic=analysis -f /helm/worker/values.yaml /helm/worker
 helm upgrade --install worker-dataset-upload --set workerDeploymentName=worker-dataset-
upload,workerTopic=dataset_upload -f /helm/worker/values.yaml /helm/worker
 helm upgrade --install worker-model-training --set workerDeploymentName=worker-model-
training,workerTopic=model_training -f /helm/worker/values.yaml /helm/worker
 helm upgrade --install worker-purging --set workerDeploymentName=worker-
purging,workerTopic=purging -f /helm/worker/values.yaml /helm/worker
```

To list the containers deployed using Helm, run the following command:

```
helm ls
```

To check the upgrade history of the AICS Worker containers, run the following command:

```
helm history <worker_name>
```

To roll back the upgrade of the AICS Worker containers, run the following command:

```
helm rollback <worker_name> <revision_name>
```

(Optional) Mapping a Local Volume into a Container

Local directories or files can be mapped on any of the containers user by the application in an HA deployment: tango, workers. or mongo.

> **Tip**
> An HA deployment in Production mode should not use NGINX.

To mount a volume, update the file corresponding to the desired container to a local directory or file by editing the volumes declaration:

- tango: <IP_JOP_PRR_<*version_number*>_ENU_linux/ha-scripts/swarm/tango-swarm.yml
- mongo: <IP_JOP_PRR_<*version_number*>_ENU_linux/ha-scripts/swarm/mongo-swarm5.yml / <IP_JOP_PRR_<version_number>_ENU_linux/ha-scripts/swarm/mongo-swarm.yml
- workers: <IP_JOP_PRR_<*version_number*>_ENU_linux/ha-scripts/swarm/worker-swarm.yml

> **Important**
>
> Mapping a directory or file on a node makes it available only on that host. It does not create or imply any type of file replication.

To mount a local directory, follow the format presented in the following example:

- To mount /some_local_directory, into /custom_mount_point in the mongo container on node-1, edit the <IP_JOP_PRR_<*version_number*>_ENU_linux/ha-scripts/mongo-swarm.yml file as follows:

```
volumes:
   - mongodata1:/data/db
   - mongoconfig1:/data/configdb
   - ../conf/mongodb.pem:/etc/ssl/mongodb.pem
   - /some_local_directory:/custom_mount_point
```

To make the changes take effect restart the application:

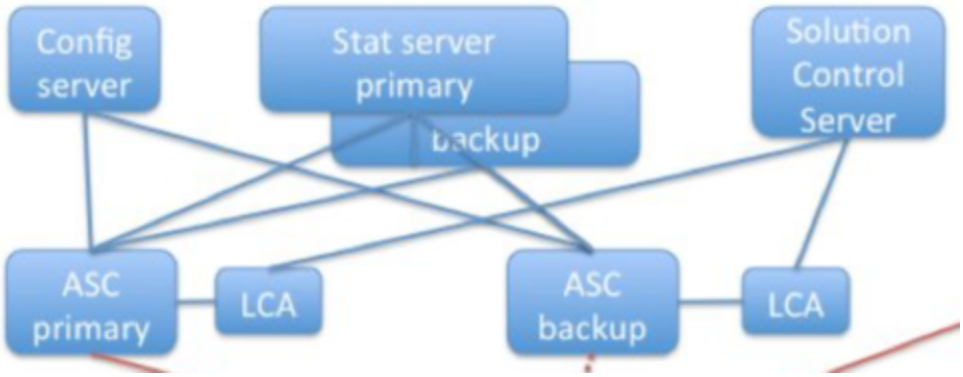
```
bash <IP_JOP_PRR_<''version_number''>_ENU_linux/ha-scripts/restart.sh
```

> **Important**
>
> Additional information can be found at https://docs.docker.com/compose/compose-file/compose-file-v2/#volumes

HA for ASC

Agent State Connector (ASC) has a standard primary-backup warm-standby high availability configuration. The backup server application remains initialized and ready to take over the operations of the primary server. It maintains connections to Configuration Server and Stat Server, but does not send agent profile updates to AICS.

To configure a primary-backup pair of ASC instances, create two ASC Application objects. Open the **Server Info** tab for the backup ASC set warm standby as the redundancy mode. When Local Control Agent (LCA) determines that the primary ASC is unavailable, it implements a changeover of the backup to primary mode.



Important

If the Stat Server instance you are using for Predictive Routing is release 8.5.100.10 or higher, you must set the value for the accept-clients-in-backup-mode configuration option in the Stat Server `Application` object to no to ensure normal backup switchover between ASC instances.

Scale AI Core Services

Correct use of scaling enables you to efficiently adapt your environment to changing conditions. The first step in increasing the size of your AI Core Services (AICS) deployment is to add new servers and configure them. You can then allocate instances of the various containers among the new servers. As with the initial deployment, use special consideration when planning how to distribute your instances of MongoDB.

> **Important**
> There is no need to shut down your AICS deployment. You can add servers while AICS is running.

Add New Servers

When GPR is deployed in high availability (HA) mode you can expand that deployment by adding additional hardware to the cluster.

To add servers to an existing cluster, follow these steps:

1. Complete the Installation steps and unpacking steps on the new servers to make them available to the AICS deployment.
2. On the node with hostname *node-1-hostname* execute the following command:

   ```
   docker swarm join-token manager
   ```

   The output of this command should look something like this:

   ```
   docker swarm join --token
    SWMTKN-1-4d6wgar0nbghws5gx6j912zf2fdawpud42njjwwkso1rf9sy9y-
    dsbdfid1ilds081yyy30rof1t 172.31.18.159:2377
   ```

3. Copy this command and execute it on the new servers. This adds the new nodes to your existing cluster.
4. On the new servers, execute the following command:

   ```
   bash ha-scripts/install.sh
   ```

You can now scale the services to start using the new servers.

Scaling Individual GPR components

GPR consists of multiple individual components, each with different responsibilities. These components are packaged as Docker images and executed as Docker containers. Every component of GPR can be scaled vertically (configuring already-running GPR Docker containers to use more of the existing hardware resources) or horizontally (adding new hardware and then creating additional GPR containers). Each GPR component scales independently of the others.

The sections below describe each component of GPR, when to scale it, and how to scale it both vertically and horizontally.

Scaling Tango

The Tango container processes all scoring requests and exposes the GPR APIs to the external world. The single Tango container deployed per host during installation can use as many CPUs and as much RAM as you can provide, enabling you to scale vertically to whatever extent your environment requires. The Tango container does not need to scale horizontally.

- In a single-host deployment, only one Tango container is created.
- In an HA deployment, there is one Tango container per host.

Scaling MongoDB

In single-host deployments, you can have only one MongoDB instance and it can only be scaled vertically. In HA deployments, you can have 3 or 5 MongoDB instances running on separate servers and these instances can each be scaled vertically. Genesys does not recommend you to scale MongoDB beyond this point.

> **Important**
>
> Since MongoDB, which stores all your data, is a critical part of GPR, Genesys recommends that you take time to size your MongoDB containers with deliberate care from the beginning, (during installation) rather than adding resources to your existing deployment, because later changes require downtime.

Scaling MongoDB Vertically

MongoDB should be scaled vertically when you have enough hardware but the current MongoDB resource allocation is acting as a bottleneck. By default each MongoDB container can use up to 2 CPUs and up to 8GB of RAM, whether running on a single host or in an HA deployment.

Starting in MongoDB 3.2, WiredTiger is the default storage engine for MongoDB. The `wiredTigerCacheSizeGB` parameter should be configured to use 50% of the maximum memory assigned to the MongoDB container. By default, this is 4 GB.

Single-Server Environments

Scaling MongoDB vertically in a single-host deployment requires downtime.

1. Stop GPR, using the following command:

   ```
   bash stop.sh
   ```

   - To add more CPUs to the MongoDB instance, change the value of the `cpus` setting, found in the **IP_JOP_PRR_<*version_number*>_ENU_linux/scripts/docker-compose.yml** file under the `mongo` container.
   - To add more RAM to the MongoDB instance, change the value of the `mem_limit` setting, found in the **IP_JOP_PRR_<*version_number*>_ENU_linux/scripts/docker-compose.yml** file under the `mongo` container.
   - To change the amount of memory allocated to the wiredTiger storage engine for the MongoDB instance, change the value of the `wiredTigerCacheSizeGB` setting, found in the **IP_JOP_PRR_<*version_number*>_ENU_linux/scripts/docker-compose.yml** file under the `mongo` container.

2. Restart GPR, using the following command:

   ```
   bash start.sh
   ```

HA Environments

Scaling MongoDB vertically in an HA deployment requires downtime.

1. Stop GPR, using the following command:

   ```
   bash stop.sh
   ```

   - To add more CPUs to the MongoDB instances, change the value of the `cpus` setting, found in the **IP_JOP_PRR_<version_number>_ENU_linux/ha-scripts/swarm/mongo-swarm.yml** file under the `mongo1`, `mongo2`, `mongo3` containers and, if you are using five containers, the `mongo4` and `mongo5` containers.
   - To add more RAM to the MongoDB instances, change the value of the `memory` setting, found in the **IP_JOP_PRR_<version_number>_ENU_linux/ha-scripts/swarm/mongo-swarm.yml** file under the `mongo1`, `mongo2`, `mongo3` containers and, if you are using five containers, the `mongo4` and `mongo5` containers.
   - To change the amount of memory allocated to the wiredTiger storage engine for each MongoDB instance, change the value of the `wiredTigerCacheSizeGB` setting, found in the **IP_JOP_PRR_<version_number>_ENU_linux/ha-scripts/swarm/mongo-swarm.yml** file under the `mongo1`, `mongo2`, `mongo3` containers and, if you are using five containers, the `mongo4` and `mongo5` containers.

2. Restart GPR, using the following command:

   ```
   bash start.sh
   ```

Scaling MongoDB Horizontally

You can only have three or five MongoDB instances in an HA deployment. You can scale MongoDB

horizontally only if you currently have three MongoDB instances in the replica set.

1. If you have not already done so, complete the steps in Adding New Servers (above) to provision additional hardware capacity.
2. Label the newly created servers so that they can run additional MongoDB instances. Use the procedure given in Label MongoDB Nodes in the Cluster (above).
3. Restart GPR, using the following command:

```
bash start.sh
```

Scaling MinIO

In release 9.0.013.01 and higher, MinIO is used for dataset uploads. By default, GPR creates one MinIO container for each host, each using two CPUs. There is no need to scale MinIO either vertically or horizontally.

Scaling Workers

GPR includes a number of different containers that run workers, each of which performs a different task in an asynchronous manner. As a result, you can scale the workers of the different types as needed in your environment.

> **Important**
> The exact number and types of containers vary depending on your release. In earlier releases, more of the functionality is performed in the Tango container; in later releases, some of these functions are split into separate containers.

Model Training Workers

Model training workers are responsible for executing model-training jobs.

Scaling Vertically

Model-training workers can be scaled vertically to reduce the time required for individual model-training jobs.

The model-training algorithms cannot use more than four CPUs, which is the default value. If you choose to allocate fewer CPUs, make the changes in the following location (this applies to both single-server and HA deployments):

- The `cpus` setting in the `model_training` section of the **IP_JOP_PRR_<*version_number*>_ENU_linux/scripts/docker-compose.yml** file.

Adding extra RAM to the model-training jobs can speed up the execution of individual jobs. The more RAM you allocate to each model-training container, the faster the execution. Model-training containers use as much RAM as is available on the server where they are deployed.

Scaling Horizontally

Adding more model-training containers ensures that more models can be trained in parallel. Jobs that can not be executed immediately are queued for execution. You can scale model-training containers horizontally, in either HA or single-host deployments, assuming you have enough hardware for the newly created containers.

By default, GPR creates one model-training container that uses up to four CPUs and as much RAM as is available. With this configuration, you can perform only one model-training job at a time. All other jobs are queued.

To scale model-training jobs horizontally, perform the following steps:

1. Stop GPR using the following command:

   ```
   bash stop.sh
   ```

   - For a single-server environment, change the value of the `NUM_OF_MODEL_TRAINING_WORKERS_INSTANCES` configuration variable, found at the beginning of the **IP_JOP_PRR_<*version_number*>_ENU_linux/scripts/start.sh** script.
   - For an HA environment, change the value of the `NUM_OF_MODEL_TRAINING_WORKERS_INSTANCES` configuration variable, found at the beginning of the `IP_JOP_PRR_<version_number>_ENU_linux/ha-scripts/start.sh` script.

2. Restart GPR using the following command:

   ```
   bash start.sh
   ```

Dataset Upload Workers

In releases prior to 9.0.013.01, the dataset-upload workers are responsible for uploading Datasets as well as loading data to MongoDB. (In release 9.0.013.01 and higher, the initial upload is performed by MinIO.)

Scaling Vertically

Dataset-upload workers can be scaled vertically to reduce the time required for individual Dataset upload jobs. The Dataset-upload containers use as many CPUs as you allocate. By default, this value is set to two CPUs.

To change this number, edit the `cpus` setting in the `dataset_upload` section of the appropriate file for your environment:

- For single-server deployments: **IP_JOP_PRR_<*version_number*>_ENU_linux/scripts/docker-compose.yml**
- For HA deployments: **IP_JOP_PRR_<*version_number*>_ENU_linux/ha-scripts/swarm/worker-swarm.yml**

Adding extra RAM to the dataset-upload jobs can speed up the execution of individual jobs. The more RAM you allocate to each dataset-upload container, the faster the execution. Dataset-upload containers can use as much RAM as is available on the server where they are deployed.

Scaling Horizontally

Adding more dataset-upload containers ensures that more Datasets can be uploaded in parallel. Jobs that cannot be executed immediately are queued for execution. You can scale dataset-upload containers horizontally regardless of whether GPR is deployed in an HA or a single-server environment. The only requirement is that there must be enough hardware for the newly created containers.

By default, GPR creates one dataset-upload worker for each container, each using up to two CPUs and as much RAM as is available. With this configuration, you can upload only one dataset in parallel. All other jobs are queued.

In order to scale dataset-upload jobs horizontally, perform the following steps:

1. Stop GPR using the following command:

   ```
   bash stop.sh
   ```

   - For a single-server environment, change the value of the `NUM_OF_DATASET_UPLOAD_WORKERS_INSTANCES` configuration variable, found at the beginning of the **IP_JOP_PRR_<*version_number*>_ENU_linux/scripts/start.sh** script.
   - For an HA environment, change the value of the `NUM_OF_DATASET_UPLOAD_WORKERS_INSTANCES` configuration variable, found at the beginning of the `IP_JOP_PRR_<version_number>_ENU_linux/ha-scripts/start.sh` script.

2. Restart GPR using the following command:

   ```
   bash start.sh
   ```

Analysis Workers

Analysis containers are used for various analysis jobs, supporting creation of the Lift Estimation report and the Feature Analysis report, among others. By default, GPR creates one analysis container for a single-host deployment and two analysis containers for an HA deployment. Each container uses up to two CPUs (the default setting) and as much RAM as is available.

In a single-host deployment, by default, you have only one container, which means you can only run one analysis job at a time. All other analysis jobs are queued. To run analysis jobs in parallel, perform the following steps:

1. Stop GPR using the following command:

   ```
   bash stop.sh
   ```

2. Change the value of the `NUM_OF_ANALYSIS_WORKERS_INSTANCES` configuration variable, found in the **IP_JOP_PRR_<*version_number*>_ENU_linux/scripts/start.sh** script from 1 to 2.
3. Restart GPR using the following command:

   ```
   bash start.sh
   ```

In an HA deployment, by default, you have two analysis containers running, which means you can run two analysis jobs at the same time. All other analysis jobs are queued. To change this value, perform the following steps:

1. Stop GPR using the following command:

```
bash stop.sh
```

2. Change the value of the `NUM_OF_ANALYSIS_WORKERS_INSTANCES` configuration variable, found in the **IP_JOP_PRR_<*version_number*>_ENU_linux/ha-scripts/start.sh** script, from 2 to your desired number.
3. Restart GPR using the following command:

```
bash start.sh
```

To increase the processing speed for each analysis job, increase the number of CPUs the analysis container can use.

To change this value, perform the following steps:

1. Stop GPR using the following command:

```
bash stop.sh
```

2. Edit the `cpus` setting in the `analysis` section of the appropriate file for your environment:

- For single-server deployments: **IP_JOP_PRR_<*version_number*>_ENU_linux/scripts/docker-compose.yml**
- For HA deployments: **IP_JOP_PRR_<*version_number*>_ENU_linux/ha-scripts/swarm/worker-swarm.yml**

1. Restart GPR using the following command:

```
bash start.sh
```

Purge Workers

By default, GPR creates one purge container for each host, each using at most one CPU and at most two GB RAM.

There is no need to scale this type of worker either vertically or horizontally.

Deploy Agent State Connector

Agent State Connector (ASC) connects to Configuration Server and (in release 9.0.015.04 and higher, optionally) to Stat Server. It retrieves changes to Person and Agent Group configuration objects and updates to agent login data. It sends that information to the AI Core Services (AICS), which uses it to update agent profiles, agent availability, and (optionally) agent statistics.

Deploying ASC has three main phases:

- Create and configure the ASC `Application` object in your Genesys configuration application.
- Install Agent State Connector.
- Configure HTTP/HTTPS Connections and (Optional) Configure ASC to Use Shift-JIS Encoding.

Environment Assumptions
The instructions in this section assume that you are creating new `Application` objects under the Environment folder, in either a single-tenant or multi-tenant configuration environment. To create `Application` objects under a particular Tenant folder in a multi-tenant configuration environment, replace the word *Environment* with the name of your Tenant folder in the configuration instructions.

> **Important**
> In a multi-tenant environment, configure one ASC instance for each Tenant.

Configure the ASC Application

Create and Configure the ASC Application Object

1. Import the Application Template
2. Create an ASC Application Object
3. Configure the General tab
4. Configure the Server Info tab
5. Configure the Start Info tab
6. Configure the Options tab - The configuration options control many aspects of ASC behavior, including the ability to monitor Stat Server statistics, to control threading and timeouts, and to specify whether to create an Agent Profile schema automatically.
7. Configure the Tenants tab
8. Configure the Connections tab

Import the Application Template

Before you can configure an Application object for ASC, you must import its Application template. The Application template provides a majority of the configuration options, as well as the default values for them. You can use this Application template to create as many Application objects of the same type as you need.

> **Important**
>
> For an explanation of how to use Genesys Administrator Extension to import the Application template and to create a new Application object, see the Genesys Administrator Extension Help file, which is directly available from Genesys Administrator Extension user interface, as well as from the link given here.

1. On the Configuration Manager window in Genesys Administrator Extension, select the **Environment > Application Templates** folder.
2. From the **File** menu, select **Import Application Template**.
3. In the **Look In** box, click the down arrow.
4. Browse to the IP for Agent State Connector and open the **TEMPLATES** folder.
5. Select the template file for Agent State Connector; it is called '*AgentStateConnector_900.apd*.
6. Click **Open** to open the **Properties** dialog box for the template.
7. Make any changes that you require.
8. Click **OK** to save the template and close the **Properties** dialog box.

The next step is to configure an ASC Application object.

Create an ASC Application Object

After you import the application template, you can create and configure an Application object for ASC by using the Configuration Manager tab in Genesys Administrator.

1. On the Configuration Manager window, select the **Environment > Applications** folder.
2. From the **File** menu, select **New > Application**.
3. From the available application templates in the **Browse** dialog box, select the template that you imported for ASC.
4. On each of the Application tabs, enter the settings appropriate for your environment, as explained in detail in the section that follow.
5. Click **OK** to save your changes, then close the **Properties** dialog box.

Configure the General tab

In the **Properties** dialog box, click the **General** tab, and then enter a name for this application.

Configure the Server Info tab

Click the **Server Info** tab, and then specify the following properties:

- Host—Select or enter the name or IP address of the computer on which you want to install and/or run this server.
- Port—Enter the value `0` (zero) for the port number.

> **Important**
> In IPv6 deployments, you cannot set the IP address of the host—only IPv4 addresses can be set for the host. Therefore, enter the *name* of the host instead.

Configure the Start Info tab

> **Tip**
> The properties you define here are updated automatically during the installation procedure.

Click the **Start Info** tab, and then specify the following properties:

- Working Directory—Enter the full path to the directory from which the application starts.
- Command Line—<Path_to_JDK_installation_folder>\bin\java.exe
- Command Line Arguments—Add the following startup arguments that will be used in **AgentStateConnector.bat** (Windows) or **AgentStateConnector.sh** (unix) environments:

```
Dcom.genesyslab.platform.commons.log.loggerFactory=log4j -jar agent_state_connector-
<AGENT_STATE_CONNECTOR_VERSION>-jar-with-dependencies.jar -host <cfg server host>
-port <cfg server port> -app <ASC Application object>
```

  - For additional information about command-line parameters, see Starting and Stopping the ASC Application.

Configure the Options tab

Click the **Options** tab, and then specify or change the values of the configuration options, as suitable for your deployment.

> **Important**
>
> - If you are deploying ASC 9.0.015.04 or higher and plan not to connect to Stat Server, see Configure ASC Without Stat Server for the required configuration options settings.

- For information about the entire set of ASC configuration options, see ASC Configuration Options.
- For recommendations and tips on what values to use for certain of the ASC configuration options, see Guidelines for Configuration Options Values in the *Operations: Agent State Connector* topic.

(Optional) Configure ASC to Monitor Statistics

1. Make sure the statistics you want to monitor are configured in Stat Server. StatAgentOccupancy is not a default statistic and requires you to set it up manually. For instructions, see Create a Custom Stat Server Statistic, below.
2. On the ASC `Application` object **Options** tab, create a new section, named **statistics**. (All other ASC options are configured in the **default** section.)
3. In the **[statistics]** section, create a new option with the same name as the statistic you want to monitor. For example, StatAgentOccupancy.
   In the 9.0.012.00 release, the only supported statistic is StatAgentOccupancy.
4. Set the option value to the correct TimeProfileName for the specified statistic, as specified in the Stat Server `Application` object. For StatAgentOccupancy, set the value to `SinceLogin`.
5. Save the new option.
6. Check that the value for the **timebased-statistic-interval** is appropriate for your environment. The default value is `60` seconds.
7. Restart ASC to have the changes take effect.

The new statistic is also saved as part of the Agent Profile schema.

Create a Custom Stat Server Statistic

To create custom statistics, including the StatAgentOccupancy statistic, use the following procedure:

1. In the Stat Server `Application` object, create a new configuration section with the name of the desired statistic. For example, **StatAgentOccupancy**.
2. Create the following options within the new **[StatAgentOccupancy]** section, and set them to the specified values:

- Category=RelativeTimePercentage
- MainMask=CallDialing, CallRinging, AfterCallWork, CallInbound, CallOutbound, CallInternal, CallConsult, CallUnknown
- RelMask= Monitored, LoggedIn, OnHook, WaitForNextCall, OffHook, CallDialing, CallRinging, NotReadyForNextCall, AfterCallWork, OfflineWorkType2, BreakType1, BreakType2, CallOnHold, NotUsed, NotUsed, ASM_Engaged, ASM_Outbound, CallInbound, CallOutbound, CallInternal, CallConsult, CallUnknown
- Objects=Agent, GroupAgents
- Subject=AgentStatus

3. Save your changes in the Stat Server `Application` object.
   For additional information on statistics configuration in Stat Server, see Statistic Configuration Options in the *Stat Server User's Guide*.
4. Check whether the necessary TimeProfile value is configured in the **[TimeProfiles]** section on the Stat Server `Application` object. For StatAgentOccupancy, you must configure the **SinceLogin** time profile type. If it does not exist:
   1. Open the **[TimeProfiles]** section.
   2. Create a new option with name SinceLogin,SinceLogin. All options in the **[TimeProfiles]** section have the name format *<TimeProfileName>,<Type>*.
   3. Leave the option value empty.
   4. Save changes in Stat Server `Application` object.
      For additional information on time profiles in Stat Server, see TimeProfiles Section in the *Stat Server User's Guide*.
5. Restart Stat Server.

Log Options

- Configure both ASC-specific log options and common log options in the log-related configuration sections. For option descriptions, see Log Options and the *Framework Configuration Options Reference Manual*.

Configure the Tenants tab

> **Tip**
> The Tenants tab is displayed only in a multi-tenant environment.

- Click the **Tenants** tab, and then click **Add** to add all tenants that this ASC application will serve. ASC only monitors Person objects that are associated with the tenants you specify.
- If this ASC instance is required to monitor the objects that are configured under the **Environment** folder, assign the **Environment** tenant among the other tenants.

Configure the Connections tab

Click the **Connections** tab, and then add the following connections:

- Configuration Server
- Stat Server (optional in release 9.0.015.04 and higher; if you do not add a Stat Server to the **Connections** tab, agent availability data is taken from Universal Routing Server)
- Message Server

Configure ASC Without Stat Server

In ASC release 9.0.015.04 and higher, configure the following settings to operate without Stat Server:

1. Do not add Stat Server from the **Connections** tab. If Stat Server was added previously, remove it.
2. Specify the following configuration option values on the Predictive_Route_DataCfg Transaction List object:
   - **use-action-filters** = `false`
   - **login-status-expression** = `&((loginStatus>0&loginStatus<23)|loginStatus>23)`
   - **use-login-status** = `true`

High Availability

In high availability (HA) environments using primary and backup pairs of servers, the servers listed on the **Connections** tab are handled as *primary*. To specify the backup servers for any primary servers, open the `Application` object for the primary server and add the backup server on the primary server's **Server Info** tab.

Install Agent State Connector

You can install ASC on either a Windows system or a Linux 64-bit system.

Prerequisites

- You have created and configured an ASC `Application` object in the interface you use for configuration, as described above.
- Configuration Server is installed and running in your environment.
- You have identified the following parameters, which you need to configure the connection to Configuration Server:
  - ASC Host name: By default, this is the host name of the machine on which you install ASC.
  - Configuration Server Host name.
  - Network port: Configuration Server network port.

- User name: Configuration Server user name.
- Password: Configuration Server password.
- Installation path: Full path to the ASC installation directory.

Installing on Windows

To install, perform the following steps:

1. Do one of the following:
   - Insert the ASC CD into the CD-ROM drive of the machine on which you want to install ASC.
   - Download the ASC IP to the desired location on the target machine.
2. Navigate to, and open, the **.../windows** directory.
3. Double-click the **setup.exe** file, and then follow the directions in the installation wizard.

Installing on RedHat Linux 7 64-Bit

Linux-specific prerequisite:

- Install tar and gunzip.

To install, perform the following steps:

1. Install the C runtime libraries using the following command:
   `yum install glibc.i686`
   - **Troubleshooting Notes:**
     - If the C runtime libraries are not available, the following error message appears: `./Perl: /lib/ld-linux.so.2: bad ELF interpreter: No such file or directory`.
     - If you see the following error during installation, copy the 32-bit versions of **libgcc_s.so.1** and **libstdc++.so.6** to the **.../lib/** directory: `./cfgutility: error while loading shared libraries: libstdc++.so.6: cannot open shared object file: No such file or directory`.
2. Insert the ASC CD into the CD-ROM drive of the machine on which you want to install ASC; or, download the ASC IP to the desired location on the target machine.
3. Unzip the installation file using the following command:
   `tar -xvzf ip_<version_number>.tar.gz`
4. Update the **install.sh** script to work on 64-bit systems by making the following changes:
   - Replace `./gunzip` with `gunzip`.
   - Replace `tar_name=./tar` with `tar_name=tar`.
5. Run **install.sh** and follow the instructions to set up ASC, using the configuration parameters you gathered.

Configure HTTP/HTTPS Connections

You should already have configured HTTPS for AI Core Services before starting the following procedure.

1. Import the tango.crt file from the ***<GPR_IP_version>*/conf** folder to the java keystore. For example, you might enter a command similar to the following:

```
keytool -import -alias gpr-ssl -keystore /usr/java/jdk1.8.0_171-amd64/jre/lib/security/cacerts -file tango.crt -storepass
 "changeit"
```

2. Depending on your version of ASC and your platform, perform the following steps.
   - For ASC release 9.0.015.00 and higher, HTTP/S configuration is done using only the **jop-base-url** configuration option:
     In both Linux and Windows environments:
     - Change the value of the URL entered in the **jop-base-url** configuration option so that it specifies `https` or `http`, as desired.
       For example, to configure ASC to use HTTPS, your new option value might look similar to the following:

       `https://fce-u0009.us.int.genesyslab.com/api/v2.0`.
   - For Agent State Connector 9.0.014.01 and lower, HTTP/S configuration is done using only the USE_HTTP environment variable:
     In Linux environments:
     - To use HTTPS, set the USE_HTTP environment variable to `" "` (an empty string).
     - To use HTTP, set the USE_HTTP environment variable to `true`.

     In Windows environments:
     - To use HTTPS, remove the USE_HTTP environment variable entirely.
     - To use HTTP, set the USE_HTTP environment variable to `true`.
3. Restart ASC.

(Optional) Configure ASC to Use Shift-JIS Encoding

By default, ASC uses UTF-8 encoding. To configure ASC to use Shift_JIS encoding, perform the manual configuration steps specified for your environment:

Linux

1. Navigate to the directory where you installed ASC.
2. Open the **AgentStateConnector.sh** file.
3. Locate the following line:
   ```
   JVMPARAMS="-server -Xmx2g -Xms2g -Xss512k"
   ```
4. At the end of the line, add the following:
   ```
   -Dfile.encoding=Shift_JIS
   ```
   The line should now read:
   ```
   JVMPARAMS="-server -Xmx2g -Xms2g -Xss512k -Dfile.encoding=Shift_JIS"
   ```
5. Save the **AgentStateConnector.sh** file.
6. Restart ASC.

Windows

1. Navigate to the directory where you installed ASC.
2. Open the **AgentStateConnector.bat** file.
3. Locate the following line:
   SET JVMPARAMS="-server -Xmx2g -Xms2g -Xss512k"
4. At the end of the line, add the following:
   `-Dfile.encoding=Shift_JIS`

   The line should now read:

   ```
   SET JVMPARAMS="-server -Xmx2g -Xms2g -Xss512k -Dfile.encoding=Shift_JIS"
   ```
5. Save the **AgentStateConnector.bat** file.
6. Restart ASC.

Configuration Options

Genesys Predictive Routing uses configuration options to enable you to specify certain behaviors. Options relating to the AI Core Services (AICS) and the strategy subroutines are configured in a Transaction List object, configured in Genesys Administrator in the following location: **PROVISIONING > Routing/eServices > Transactions > List Objects > DEV > AgentScoring**.

> **Important**
>
> Before release 9.0.009.01, AI Core Services (AICS) was known as Journey Optimization Platform (JOP).

Agent State Connector (ASC) has its own `Application` object, where you configure options relating to specifically to ASC functionality.

- Predictive_Route_DataCfg Transaction List Object Options. Some functionality has multiple options controlling the desired behavior:
  - Agent Occupancy Options
  - Agent Holdout Options
  - Dynamic Interaction Priority Options
- Agent State Connector Configuration Options
  - ASC Log Options

Predictive_Route_DataCfg Transaction List Object Options

ab-test-time-slice
context-id-key
default-agent-score
emergency-scoring-token
format-as-map
global-map-timeout
jop-api-key
jop-auth-url
jop-logging-url
jop-password
jop-scoring-url
jop-username
login-status-expression
log-to-api
max-score

orig-connid-key
overload-control-timeout
prr-mode
scoring-token-expiration
send-user-event
udata-keys-to-exclude
use-action-filters
use-crm-query
use-double-selection
use-login-status
use-setreadycondition
vq-for-reporting

Agent Occupancy Options

Agent occupancy is the percentage of time that an agent is working while logged in, a service objective that can be specified when building a staffing forecast. Agent occupancy data is taken from Stat Server by URS using the SData function. Stat Server collects agent occupancy data using the StatAgentOccupancy statistic. The routing strategy filters agents by occupancy in the ScoreIdealAgent callback subroutine. The agent occupancy results are used to sort the agents in the target agent group; over-occupied agents drop down lower in the sorted list.

agent-occupancy-factor
agent-occupancy-threshold
use-agent-occupancy

Agent Holdout Options

Agent hold-out enables you to have an interaction wait a specified time, even when an agent has become available, if the available agent is has a low score for the interaction and there is a chance a better-matched agent might become available within the configured time window.

initial-threshold-timeout
score-base-threshold
threshold-relaxation-step
threshold-relaxation-timeout

Dynamic Interaction Priority Options

If an interaction has a low score for all targeted agents, it can stay in a queue for a long time. To avoid such situations, you can configure a schedule for incremental priority increases. The schedule is set once for each interaction processed by GPR. The following options control interaction priority increments.

> **Important**

> If you already use priority increments for the strategy into which you are inserting the GPR subroutines, you doo not need to configure these options. If you are using priority increments only for predictive routing, use the following options to configure it.

priority-increment
priority-init-interval
priority-interval
set-dynamic-priority

ab-test-time-slice

Specifies the length, in seconds, of the periods of time when Predictive Routing and skill-based routing are alternately turned on when you have set the **prr-mode** configuration option to `ab-test-time-sliced`. Genesys recommends that you do not set the value of this option to less than `600` seconds in a production environment.

- Configured in: Predictive_Route_DataCfg List object, **[default-predictor]** or **[<predictor_name>]** section
- Default value: 1741
- Valid values: Any positive integer
- Changes take effect: Immediately

context-id-key

Specifies the name of the user data key containing an ID for the current interaction, using which the Predictive Routing scoring engine can retrieve a record from an internal database of customer profiles (CRM database) and use features from the record to compute agents scores for the interaction.

To incorporate customer profile data into models for matching the agents, a copy of the CRM database must be uploaded into JOP before you train a predictor model. The URS ActivatePredictiveRouting subroutine attaches a `context_id` key to the scoring request body and provides the value of the user data key defined by this option as the `context_id` value.

If the returned customer ID is empty or you set the option value to `ANI`, the interaction ANI is used.

- Configured in: Predictive_Route_DataCfg List object, **[default-predictor]** or **[<predictor_name>]** section
- Default value: `ANI`
- Valid values:
  - `ANI`
  - A valid user data key name holding a customer ID

- Changes take effect: Immediately

default-agent-score

The option specifies the value the ScoreIdealAgent and isAgentScoreGood subroutines should use as the agent score for an interaction for those agents who belong to the target Agent Group but that GPR did not score. For example, an agent might be logged out, or in another status configured as unavailable, until after the scoring request it sent. If such an agent then becomes available before the interaction is routed, GPR assigns that agent the default score.

> ### Important
>
> This option functions differently depending on the release of URS Strategy Subroutines you have deployed:
>
> - In release 9.0.015.00 and higher, gpmAgentScore records the default score assigned to agents GPR did not score. The ScoreIdealAgent subroutine uses this value to sort the scores and the isAgentScoreGood subroutine compares it against any threshold you have configured to determine whether the agent is acceptable.
> - In release 9.0.014.04 and lower, the gpmAgentScore user data KVP always contains the value `0` for such agents. The score specified in this option is used only when URS is sorting the agents in the target group according to their scores.

- Configured in: Predictive_Route_DataCfg List object, **[default-predictor]** or **[<predictor_name>]** section
- Default value: The agent is assigned a score of `0`, which means that the agent is unlikely to receive an interaction from the queue.
- Valid values:
  - `max` - Use the maximum score calculated for an agent in the target agent group.
  - `median` - Use the median score calculated for the target agent group.
  - `global` - Use the average global score for the agents in the target group.
  - `min` - Use the minimum score calculated for an agent in the target agent group.
  - `0` - Use the value `0` as the score.
- Changes take effect: On the next interaction

emergency-scoring-token

Provides an emergency token in the event of continued authentication errors. It is intended for use only in scenarios where the strategy is unable to automatically update the token required to access

the Predictive Routing API.

> **Warning**
> This option should only be used in an emergency situation.

- Configured in: Predictive_Route_DataCfg List object, **[default]** section
- Default value: empty string
- Valid values: Any valid security token string
- Changes take effect: Immediately

format-as-map

The IRD subroutine ActivatePredictiveRouting_v<*version_number*> and the Composer subroutine ActivatePredictiveMatching now support two types of responses to, and score requests to, the Predictive Routing API, either containing both **list** and **list_ranks** fields or just the **list** field.

If set to `true`, the response and the score request to the Predictive Routing API contains two fields, **list** and **list_ranks**. The 'list' field contains a JSON dictionary with agent employee IDs as the keys and agent scores for the current interaction as the values. The **list_ranks** field contains a JSON dictionary with agent employee IDs as the keys and agents ranked according to their scores in the target group as values.

If set to `false`, the response and the score request to the Predictive Routing API contains only the **list** field. The value of this field is a JSON list object, where the items in the list are JSON dictionary objects. Each dictionary item contains the fields: **id** (agent employee ID), **score** (the score that agent has for the current interaction), and **score_type** (the type of a model, local or global, used to compute the score). The list is sorted by agent score in decreasing order.

- Configured in: Predictive_Route_DataCfg List object, **[default]** section
- Default value: `true`
- Valid values: `true, false`
- Changes take effect: Immediately

global-map-timeout

Defines the time period, in seconds, during which supporting information about an interaction (such as the predictor name and ID, the model name and ID, the Predictive Routing operation mode, and the interaction time in queue) are stored in the Universal Routing Server (URS) global map. If option value is set to `0`, the records are stored indefinitely.

- Configured in: Predictive_Route_DataCfg List object, **[default]** section
- Default value: `7200`
- Valid values: (integer) any non-negative integer
- Changes take effect: On the next interaction

> **Important**
>
> To improve URS performance, agent scores are stored in the URS global map with a timeout value of `0` (indefinitely). To remove them, you must call the PrrIxnCleanup subroutine after the interaction has been successfully routed.

jop-api-key

Specifies an access key that is used by the Agent State Connector or the ActivatePredictiveRouting subroutine in URS—depending on where the option is configured—to access the Genesys Predictive Routing API. To obtain the value of this option, open the **Accounts** tab in the Predictive Routing user interface and open your account (or create, to add a new account). The **API key** field appears in the **Account** configuration window. For details, see Settings: Configuring Accounts in the *Genesys Predictive Routing Help*.

- Configured in:
  - Predictive_Route_DataCfg List object, **[default]** section
  - Agent State Connector, **[default]** section
- Mandatory: yes
- Default value: none
- Valid values: Any valid AICS API key
- Changes take effect: After restart

jop-auth-url

Specifies the Genesys Predictive Routing API authentication endpoint URL. This value is the host name of the server on which the AI Core Services (AICS) component is installed followed by `/api/v2.0/authenticate`.

- Configured in: Predictive_Route_DataCfg List object, **[default]** section
- Mandatory: yes
- Default value: none

- Valid values: (string) A valid AICS authentication endpoint URL, in the following format: `<aics_server_host_name>/api/v2.0/authenticate`
- Changes take effect: immediately

jop-logging-url

Defines the URL for logging the interaction routing score log and outcome results to the Predictive Routing web application REST API.

- Configured in: Predictive_Route_DataCfg List object, **[default]** section
- Default value: none
- Valid values: (string) any valid URL
- Changes take effect: On the next interaction

jop-password

Specifies a user password valid for use with the Genesys Predictive Routing.

- Configured in:
  - Predictive_Route_DataCfg List object, **[default]** section
  - Agent State Connector, **[default]** section
- Mandatory: yes
- Default value: none
- Valid values: (string) The password for any valid Predictive Routing user
- Changes take effect: After restart

jop-scoring-url

The ActivatePredictiveRouting strategy subroutine in URS uses the URL defined by this option as the HTTP address to send scoring requests to AI Core Services (AICS) Scoring REST API. This URL should be the value for the **jop-base-url** option with `<predictor_name>/score` appended.

You can locate the predictor ID in messages returned from the GPR API or in the browser URL address when you are in the GPR application with the page for the desired predictor open.

- Configured in: Predictive_Route_DataCfg List object, **[default-predictor]** or **[<predictor_name>]** section

- Default value: none
- Valid values: (string) a valid AICS scoring endpoint URL + a valid predictor ID
- Changes take effect: On the next interaction processed

jop-username

Specifies a user's username to access Genesys Predictive Routing.

- Configured in:
  - Predictive_Route_DataCfg List object, **[default]** section
  - Agent State Connector, **[default]** section
- Mandatory: yes
- Default value: none
- Valid values: (string) Any valid email address registered with Predictive Routing.
- Changes take effect: After restart

login-status-expression

If you set the value of the **use-login-status** option to `true`, the value of the **login-status-expression** option is added to the action_filters expression in the ActivatePredictiveRouting_v3 subroutine when the scoring request is created.

- Configured in: Predictive_Route_DataCfg List object, **[default-predictor]** or **[<predictor_name>]** section
- Default value: no default value
- Valid values:
  - &((loginStatus>0&loginStatus<23)|loginStatus>23) - Instructs the scoring engine to evaluate scores for those agents identified as part of the target group by a skill expression or an Agent Group name who are logged into the voice channel.
  - &(loginStatus=4|loginStatus=9) - Instructs the scoring engine to evaluate scores for those agents identified as part of the target group by a skill expression or an Agent Group name who are ready to accept an interaction, or have status AfterCallWork on the voice channel.
- Changes take effect: On the next interaction

log-to-api

Specifies whether logging is enabled to the Predictive Routing application REST API from the routing strategy. If the option value is set to `true`, the context of the interaction is submitted to Predictive Routing when the PrrIxnCompleted subroutine is called, before interaction is routed to an agent.

- Configured in: Predictive_Route_DataCfg List object, **[default]** section
- Default value: `false`
- Valid values:
  - `true`: The context of the interaction is submitted to the Predictive Routing application when the PrrIxnCompleted subroutine is called before the interaction is routed to an agent.
  - `false`: Logging is not enabled.
- Changes take effect: On the next interaction

max-score

Defines the maximum score that an agent can be assigned for an interaction. The value of this option is used by the `ScoreIdealAgent` callback function to re-scale the agent score as the distance from an ideally matched agent for the interaction (assumed by URS to be `0`).

The value you set should correspond to the largest possible value returned by this Predictor from the scoring engine. To function properly, this value must be consistent with the value configured for the Predictor **Score expression** field. Because the GPR scoring engine and URS have different scales, you might need to adjust returned scoring values using the **Score expression** field in the Predictor configuration. See the instructions for how to configure this field in Creating and Updating Predictors in the *Genesys Predictive Routing Help* for more information.

To take advantage of the most precise values, set **max-score** to `10000` and the value for Score expression in the Predictor configuration to `10000 * p_score`. (*p_score* is a term used in the GPR documentation to indicate the raw score returned from the scoring engine. It is not in any way derived from or related to the statistical term *P value*.) For example, if scores range from -4 to 10, use the following **p_score** - `((p_score + 5) / 16) * 100`.

- Configured in: Predictive_Route_DataCfg List object, **[default-predictor]** or **[<predictor_name>]** section
- Default value: `100`
- Valid values: (integer) 1 - `<max>`
- Changes take effect: On the next interaction processed

orig-connid-key

Defines a user data key that the Predictive Routing strategy must attach on initialization. It holds the original connection ID of an interaction, which is used to uniquely identify the interaction for the scoring engine. The ActivatePredictiveRouting subroutine checks for the presence of this key when it starts processing an interaction.

- Configured in: Predictive_Route_DataCfg List object, **[default]** section
- Default value: None
- Valid values: Any valid user data key holding the original interaction connection ID
- Changes take effect: Immediately
- Mandatory: Yes

ConnIDs in Consult Calls
Agents can make consult calls to the route point using GPR to identify target agents. In such cases, your T-Server/SIP Server configuration determines which ConnID is added in the scoring request and recorded in the score log. Depending on your configuration, the ConnID could be from either the main call or the consult call.

The following table explains the different scenarios possible depending on your T-Server/SIP Server option settings.

Option Name	Values	Result for GPR
consult-user-data	separate	The ConnID key should be explicitly attached in both main call and consult call user data *before* the GPR subroutines are invoked. If the keys are attached to both calls, then the GPR score request, and the score log, report the main call and the consult call as separate, each with its respective ConnID.
consult-user-data	inherited joint	The ConnID key must be explicitly attached to the main call by the strategy that invokes the GPR subroutines. The main call and consult call are treated as a unit and both requests log only the ConnID for the main call.

overload-control-timeout

Defines a timeout value that sets the maximum delay, in milliseconds, between the moment when URS receives an Event from T-Server and when URS starts to process the Event in the strategy. If the delay is greater than the value set in this option, Predictive Routing considers the URS application overloaded and temporarily turns off. Once the URS overload ends and the strategy is processing

events within the limit defined by this timeout, Predictive Routing restarts.

- Configured in: Predictive_Route_DataCfg List object, **[default]** section
- Default value: `1000`
- Valid values: Any positive integer
- Changes take effect: Immediately

prr-mode

Specifies whether an instance of Predictive Routing should run as a production instance or as a test instance.

- Configured in: Predictive_Route_DataCfg List object, **[default-predictor]** or **[<predictor_name>]** section
- Default value: `off`
- Valid values:
  - `prod` - All the interactions that pass through the ActivatePredictiveRouting strategy subroutine are processed using Predictive Routing.
  - `off` - No interactions use Predictive Routing.
  - `ab-test-time-sliced` - The periods of time when Predictive Routing and skill-based routing are alternately turned on. The duration of each period is configured in the [<*predictor_name*>].**ab-test-time-slice** configuration option in the Predictive_Route_DataCfg `Transactions List` object.
  - `dry-run` - Predictive Routing scores agents for your interactions, but does not use the scores for routing.
- Changes take effect: Immediately

scoring-token-expiration

If configured, overrides the default token expiration time of 43200 seconds. For example, if set to `3600`, the token expires in the URS memory map in one hour, and a new token is requested from the JOP Scoring Engine.

- Configured in: Predictive_Route_DataCfg List object, **[default]** section
- Default value: `43200`
- Valid values: Any positive integer
- Changes take effect: Immediately

send-user-event

When set to `true`, the routing strategy used with Predictive Routing sends the EventUserEvent TEvent, which includes the following attributes:

- AttributeThisDN with a value indicating the virtual queue where the strategy is executed. This is set in the **vq-for-reporting** option.
- AttributeUserData containing the Predictive Routing-specific key-value pairs which provide the foundation for reports on routing outcomes presented in Genesys Interactive Insights/GCXI.

The KVP data is stored in Genesys Info Mart, and is then available to the Genesys reporting suite and to Predictive Routing, which can use this KVP data to refine predictor and model performance.

For more information on creating reports based on Predictive Routing data, see Deploying: Integrating with Genesys Reporting.

- Configured in: Predictive_Route_DataCfg List object, **[default]** section
- Default value: `false`
- Valid values: `true, false`
- Changes take effect: Immediately

udata-keys-to-exclude

Use this option to exclude unnecessary user data keys from the scoring context.

- Configured in: Predictive_Route_DataCfg List object, **[default]** section
- Default value: no default value
- Valid values: a list of KVP names to be excluded, separated by commas and no spaces
- Changes take effect: On the next interaction

use-action-filters

- Configured in: Predictive_Route_DataCfg List object, **[default-predictor]** or **[<predictor_name>]** section
- Default value: `true`
- Valid values:
  - `true` - URS uses a skill expression or Agent Group names taken from the **action_filters** field in the scoring request.
  - `false` - URS checks with the Stat Server for the target list of agents, as specified in the **login-**

status-expression option, and adds the target Agent IDs to the scoring request.

- Changes take effect: Immediately

> **Important**
>
> If **login-status-expression** is set to `&(loginStatus=4|loginStatus=9)`, indicating that the agents who are in the Ready state or ACW state (for voice calls) are the designated target agents, then the GetActionFilters subroutine uses a custom statistic called RStatGPRAgentsReadyOrACWvoice. This custom statistic is provided in the **object.kvlt** file in the URS Strategy Subroutines IP.

use-crm-query

Option name reserved for future use.

- Configured in: Predictive_Route_DataCfg List object, **[default-predictor]** or **[<predictor_name>]** section
- Default value: `true`
- Valid values:
  - `true`
  - `false`
- Changes take effect: Immediately

use-double-selection

Specifies whether URS uses a double selection mechanism, applying a custom statistic when agents have the same score to select the target agent for an interaction.

If the Predictive Routing routing solution is configured to use the agent hold-out feature (the **use-setreadycondition** option is set to `true`) and the **use-double-selection** option is set to `false`, when two or more agents are in ready state and have the same score for an interaction, the target agent for an interaction is selected at random. If the **use-double-selection** option is set to `true`, URS selects a target agent from a group of agents with equal scores based on a predefined statistic. This is a statistic passed as an argument to the SelectDN function by the routing strategy or one defined in an IRD routing block.

- Configured in: Predictive_Route_DataCfg List object, **[default]** section
- Default value: `false`
- Valid values:

  - `true`: Predictive Routing uses the double selection method.
  - `false`: The double selection method is turned off.
- Changes take effect: Immediately

use-login-status

Set the value of this option to `true` to have the value of the **login-status-expression** option added to the action_filters expression in the ActivatePredictiveRouting_v3 subroutine when the scoring request is created.

> **Important**
>
> Genesys recommends that you set this option to `true` and provide a valid value for **login-status-expression** to reduce the number of agents for whom scores are evaluated. The value `false` should be used only for debug and troubleshooting purposes in a staging environment.

- Configured in: Predictive_Route_DataCfg List object, **[default-predictor]** or **[<predictor_name>]** section
- Default value: `false`
- Valid values: `false, true`
- Changes take effect: On the next interaction

use-setreadycondition

If option is set to `true`, the strategy executes calls to the isAgentScoreGood subroutine, which temporarily removes low-scoring agents from consideration for routing. If option is set to `false`, the strategy does not execute calls to the isAgentScoreGood subroutine.

> **Important**
>
> This option takes effect only when the **prr-mode** option is set to `prod` for the same predictor.

- Configured in: Predictive_Route_DataCfg List object, **[default-predictor]** or **[<predictor_name>]** section
- Default value: false

- Valid values: true, false
- Changes take effect: On the next interaction

vq-for-reporting

Indicates the virtual queue or DN where URS sends the Genesys Predictive Routing (GPR) user event data describing the routing decision made for the interaction. The user event data, in the form of key-value pairs, is attached to EventUserEvent in the AttributeUserData attribute. This should be the same value as AttributeThis DN in the EventUserEvent event.

For more information on creating reports based on Predictive Routing data, see Deploying: Integrating with Genesys Reporting.

- Configured in: Predictive_Route_DataCfg List object, **[default]** section
- Default value: no default value
- Valid values: Any valid virtual queue or DN name
- Changes take effect: Immediately

---

agent-occupancy-factor

If you set the value of the **use-agent-occupancy** option to `true`, and the value of the agent occupancy statistic is higher than the threshold specified in the **agent-occupancy-threshold** option, the ScoreIdealAgent subroutine multiplies the score received for an agent for the current interaction by a coefficient defined by this option.

- Configured in: Predictive_Route_DataCfg List object, **[default]** section
- Default value: `0.5`
- Valid values: Float number between `0.0` and `1.0`
- Changes take effect: On the next interaction

agent-occupancy-threshold

If you set the value of the **use-agent-occupancy** option to `true`, the isAgentScoreGood subroutine compares the value of the occupancy statistic with the value you set in this option. If the occupancy value is higher than the specified threshold, the subroutine multiplies the score received for an agent for the current interaction by a coefficient defined in the **agent-occupancy-factor** option.

- Configured in: Predictive_Route_DataCfg List object, **[default]** section

- Default value: `0`
- Valid values: Any non-negative integer
- Changes take effect: On the next interaction

use-agent-occupancy

The value you set for this option determines whether the isAgentScoreGood subroutine checks for agent occupancy. If you set the value to `true`, the subroutine compares the value of the occupancy statistic with the value you set in the **agent-occupancy-threshold** option. If the occupancy value is higher than the specified threshold, the subroutine multiplies the score received for an agent for the current interaction by a coefficient defined in the **agent-occupancy-factor** value.

- Configured in: Predictive_Route_DataCfg List object, **[default]** section
- Default value: `false`
- Valid values: `true, false`
- Changes take effect: On the next interaction

---

initial-threshold-timeout

Defines a timeout, in seconds, during which the `isAgentScoreGood` URS callback function uses an initial minimum agent score, defined by the **score-base-threshold** option, to match agents to an interaction. After this timeout expires, the minimum score required to allow an agent to handle the interaction is gradually decreased.

- Configured in: Predictive_Route_DataCfg List object, **[default-predictor]** or **[<predictor_name>]** section
- Default value: `0`
- Valid values: (integer) `0` - `<max>`
- Changes take effect: On the next interaction processed

score-base-threshold

This option defines the initial minimum agent score required for an agent to be considered a match for an interaction. After the timeout defined by the **initial-threshold-timeout** option expires, the minimum score required to handle the interaction is gradually decreased. If you set the value to `0`, no initial minimum score is required and agents with any score are considered for an interaction.

- Configured in: Predictive_Route_DataCfg List object, **[default-predictor]** or **[<predictor_name>]**

section

- Default value: `0`
- Valid values: (integer) `0` - `<max>`
- Changes take effect: On the next interaction processed

threshold-relaxation-step

Defines an increment by which, while an interaction remains queued, the minimum agent score required to match the interaction is decreased after each period defined by the value of the **threshold-relaxation-timeout** option, following the initial period defined by the **initial-threshold--timeout** option.

- Configured in: Predictive_Route_DataCfg List object, **[default-predictor]** or **[<predictor_name>]** section
- Default value: 1
- Valid values: (integer) 1 - `<value of the **max-score** option>`
- Changes take effect: On the next interaction processed

threshold-relaxation-timeout

This option defines a timeout, in seconds, after which the minimum agent score required for matching an interaction is decreased by the amount defined by the value of the **threshold--relaxation-step** option.

- Configured in: Predictive_Route_DataCfg List object, **[default-predictor]** or **[<predictor_name>]** section
- Default value: 1
- Valid values: (integer) 1 - `<max>`
- Changes take effect: On the next interaction processed

---

priority-increment

Specifies the increment by which priority is increased each time.

- Configured in: Predictive_Route_DataCfg List object, **[default-predictor]** or **[<predictor_name>]** section
- Default value: 1

- Valid values: (integer) any integer
- Changes take effect: On the next interaction

priority-init-interval

Controls the time interval, in seconds, the strategy waits before starting to increment priority for a queued interaction.

- Configured in: Predictive_Route_DataCfg List object, **[default-predictor]** or **[<predictor_name>]** section
- Default value: `300`
- Valid values: (integer) any non-negative integer
- Changes take effect: On the next interaction

priority-interval

Specifies the time period, in seconds, between priority increments for a queued interaction.

- Configured in: Predictive_Route_DataCfg List object, **[default-predictor]** or **[<predictor_name>]** section
- Default value: `10`
- Valid values: (integer) any integer greater than 5
- Changes take effect: On the next interaction

set-dynamic-priority

Specifies whether dynamic priority interaction handling is enabled and handled in the GPR subroutines. When set to `true` interaction priority is incremented based on the settings configured for the other priority options. When set to `false`, dynamic priority interaction handling is not set by the Predictive Routing subroutines. If dynamic priority parameters are set elsewhere in the strategy, the option must be set to `false`.

- Configured in: Predictive_Route_DataCfg List object, **[default-predictor]** or **[<predictor_name>]** section
- Default value: `false`
- Valid values: `false, true`

- Changes take effect: On the next interaction

Agent State Connector Configuration Options

[default] Section

Configure the following options in the **[default]** section:

agents-batch-size

Defines the maximum number of agent configuration profiles that can be submitted in a single HTTP request to AI Core Services (AICS).

> **Important**
> Increasing this option value might reduce Agent State Connector startup time. However, setting it too high might cause the size of HTTP requests to become greater than 10 Mb, which is the default maximum size for an HTTP request body that AICS accepts by default.

- Configured in: Agent State Connector object, **[default]** section
- Default value: `500`
- Valid values: Integers from `1`–`1000`
- Changes take effect: On restart

auto-schema-discovery

Enables Agent State Connector (ASC), at startup, to check whether an Agent Profile schema is present. If there is no Agent Profile schema uploaded, ASC creates a schema. If an Agent Profile schema has already been uploaded, ASC checks the schema to validate that it is correctly structured. If there is no schema uploaded and ASC cannot create one, or if the uploaded schema is invalid, ASC generates an alarm message and shuts down.

- Configured in: Agent State Connector object, **[default]** section
- Default value: `true`
- Valid values: `true`, `false`
- Changes take effect: On restart

cfg-reading-threads-size

Enables you to specify whether to read agents and groups from Configuration Server using a multi-threading approach.

- Configured in: Agent State Connector object, **[default]** section
- Default value: `100`
- Valid values: Integers from `1`–`2000`
- Changes take effect: On restart

cfg-retry-request-attempts

Used to specify the number of times Agent State Connector (ASC) tries to get updated agent and agent group data from Configuration Server if the first try is unsuccessful.

> **Important**
> Genesys recommends that you do not set the value for this option higher than 5.

- Configured in: Agent State Connector object, **[default]** section
- Default value: 3
- Valid values: Any positive integer between 1 - 10
- Changes take effect: On restart
- Introduced in: 9.0.006.03

confserv-monitoring-reconnect-count

Specifies the maximum number of reconnect attempts to Configuration Server before ASC generates log event 60706, for which you should set an alarm. To be exact, if ASC detects a switchover or disconnection the number of times set in this option during the time period set in the **confserv-monitoring-reconnect-min** option, ASC generates the log event. The cancel event for this alarm should be 60707.

- Configured in: Agent State Connector object, **[default]** section
- Default value: 1
- Valid values: Integers from 10–1000
- Changes take effect: On restart

confserv-monitoring-reconnect-min

Specifies a time interval, in minutes, that ASC uses when monitoring multiple Configuration Server switchover events. If ASC detects as many switchovers or disconnects as specified in the **confserv-monitoring-reconnect-count** during the time period configured in this option, ASC generates log event 60702, for which you should set an alarm.

- Configured in: Agent State Connector object, **[default]** section
- Default value: 1
- Valid values: Integers from 10–1000

- Changes take effect: On restart

filter-by-groups

If the ASC configuration contains non-empty values for the **filter-by-skills** and/or **filter-by-groups** configuration options, ASC subscribes to Stat Server for agent statistics only for the agents included in the specified Agent Groups or those satisfying the configured skill expression. If both options are configured, the agents are subscribed for statistics if they *either* satisfy the skill expression specified in the **filter-by-skills** option *or* are included in one of the Agent Groups specified in the **filter-by-groups** option.

  - This functionality enables you to limit the number of agents monitored by GPR or to use GPR in environments where multiple Stat Servers are deployed to monitor different groups of agents.

- Configured in: Agent State Connector object, **[default]** section
- Default value: no default value
- Valid Values: A comma-separated list of valid Agent Group names
- Changes take effect: On restart

filter-by-skills

If the ASC configuration contains non-empty values for the **filter-by-skills** and/or **filter-by-groups** configuration options, ASC subscribes to Stat Server for agent statistics only for the agents included in the specified Agent Groups or those satisfying the configured skill expression. If both options are configured, the agents are subscribed for statistics if they *either* satisfy the skill expression specified in the **filter-by-skills** option *or* are included in one of the Agent Groups specified in the **filter-by-groups** option.

  - This functionality enables you to limit the number of agents monitored by GPR or to use GPR in environments where multiple Stat Servers are deployed to monitor different groups of agents.

> **Important**
>
> The & (ampersand) and | (pipe) operators are not supported in skill expressions used as the value of the **filter-by-skills** option. The skill expression must include only a single valid skill name.

- Configured in: Agent State Connector object, **[default]** section
- Default value: no default value
- Valid Values: A comma-separated list of valid agent skills

- Changes take effect: On restart

ignore-ascii-characters

Enables you to specify how Agent State Connector (ASC) handles Agent Profile columns with the following unsupported ASCII characters: `[Space], -, <, >`.

- To have ASC remove the specified characters for Agent Profile schema columns, but add the affected columns to the schema, set the option to `true`.
- To have columns with the specified characters entirely omitted from the schema, set the option to `false` (the default value), .

> **Important**
> Columns with other unsupported characters continue to be omitted from the schema. For a complete list of unsupported characters, see Configuring Agent Profiles.

- Configured in: Agent State Connector object, **[default]** section
- Default value: `false`
- Valid Values: `true, false`
- Changes take effect: On restart

ignore-employee-ids

Enables you to instruct Agent State Connector to skip processing for specified employee IDs. For example, the employee configuration might include symbols, such as $, that the database cannot process.

Additional use cases:

- An employee has an unusually large profile, which would create an unacceptable impact on predictive routing performance.
- An agent profile contains some data that produces an error when it is submitted to for predictive routing analysis.

- Configured in: Agent State Connector object, **[default]** section
- Default value: none
- Valid values: Valid employee ID numbers, separated by commas
- Changes take effect: On restart

ignore-person-annex-sections

Specifies which sections on the **Annex** tab of a `Person` configuration object the Agent State Connector (ASC) should skip when uploading the agent profile to AI Core Services. By default, ASC skips the sections related to Genesys Interaction Workspace.

- Configured in: Agent State Connector object, **[default]** section
- Default value: `interaction-workspace,interaction-workspace-recents,interaction-workspace-favorites`
- Valid values: One, or comma-separated list if more than one, valid section names on the Annex of a `Person` configuration object
- Changes take effect: On restart

Use this option to reduce startup time by stopping ASC from loading unnecessary data.

include-person-annex-sections

Specifies which sections on the **Annex** tab of a `Person` configuration object the Agent State Connector (ASC) should take information from when uploading the agent profile to AI Core Services. All unspecified sections are skipped. If both this option and **ignore-person-annex-sections** are configured, ASC disregards the value set for **ignore-person-annex-sections** and loads information only from the sections specified in the **include-person-annex-sections** option.

- Configured in: Agent State Connector object, **[default]** section
- Default value: none
- Valid values: One, or comma-separated list if more than one, valid section names on the Annex of a `Person` configuration object. For example, `interaction-workspace,interaction-workspace-recents,interaction-workspace-favorites`.
- Changes take effect: On restart

Use this option to reduce startup time by preventing ASC from loading unnecessary data.

include-groups

Use this option to specify a list of agent groups for ASC to monitor. This list is a subset of the total list of groups present in agent profiles. ASC ignores all groups except those you list. To monitor all groups, leave the option value empty (the default setting).

For example, you might set the value of this option as follows to have ASC monitor only two groups: `"GROUP1,GROUP2"`

- Configured in: Agent State Connector object, **[default]** section
- Default value: ""
- Valid Values: A comma-separated list of valid agent group names
- Changes take effect: On restart

include-skills

Use this option to specify a list of skills for ASC to monitor. This list is a subset of the total list of skills present in agent profiles. ASC ignores all skills except those you list. To monitor all skills, leave the option value empty (the default setting).

For example, you might set the value of this option as follows to have ASC monitor only two skills: `"CLOSING_AN_ACCOUNT,SALES"`

- Configured in: Agent State Connector object, **[default]** section
- Default value: ""
- Valid Values: A comma-separated list of valid skill names
- Changes take effect: On restart

jop-api-key

Specifies an access key that is used by the Agent State Connector or the ActivatePredictiveRouting subroutine in URS—depending on where the option is configured—to access the Genesys Predictive Routing API. To obtain the value of this option, open the **Accounts** tab in the Predictive Routing user interface and open your account (or create, to add a new account). The **API key** field appears in the **Account** configuration window. For details, see Settings: Configuring Accounts in the *Genesys Predictive Routing Help*.

- Configured in:
  - Predictive_Route_DataCfg List object, **[default]** section
  - Agent State Connector, **[default]** section
- Mandatory: yes
- Default value: none
- Valid values: Any valid AICS API key
- Changes take effect: After restart

jop-base-url

Specifies the common substring of Genesys Predictive Routing API endpoint URLs. This value is the host name of the server on which the AICS component is installed, followed by `/api/v2.0`.

To use HTTPS, specify `https://` in your base URL string.

- Configured in: Agent State Connector object, **[default]** section
- Mandatory: yes
- Default value: none
- Valid values: (string) A valid common substring of AICS endpoint URLs, in the following format:

`https://<aics_server_host_name>/api/v2.0` or `http://<aics_server_host_name>/api/v2.0`

- Changes take effect: After restart

jop-password

Specifies a user password valid for use with the Genesys Predictive Routing.

- Configured in:
  - Predictive_Route_DataCfg List object, **[default]** section
  - Agent State Connector, **[default]** section
- Mandatory: yes
- Default value: none
- Valid values: (string) The password for any valid Predictive Routing user
- Changes take effect: After restart

jop-username

Specifies a user's username to access Genesys Predictive Routing.

- Configured in:
  - Predictive_Route_DataCfg List object, **[default]** section
  - Agent State Connector, **[default]** section
- Mandatory: yes
- Default value: none

- Valid values: (string) Any valid email address registered with Predictive Routing.
- Changes take effect: After restart

jop-update-thread-wait-timeout

Specifies the thread waiting timeout, in milliseconds, applied to the AI Core Services (AICS) subscribe process. This timeout can prevent a polling loop from taking up unacceptable CPU bandwidth at busy periods.

- Configured in: Agent State Connector object, **[default]** section
- Default value: `50`
- Valid values: Any positive integer
- Changes take effect: On restart

reset-jop-on-startup

> **Important**
> This option is removed in Agent State Connector (ASC) release 9.0.006.08 and higher. To delete agents, delete your current agent profile schema in the Predictive Routing application, and then upload an updated schema.

Specifies whether agent profiles are recreated at startup. These profiles are used to query Stat Server about agent statistics.

- Configured in: Agent State Connector object, **[default]** section
- Default value: `true`
- Valid values:
  - `true`—When Agent State Connector (ASC) starts up, it deletes all agent profiles previously stored in the AI Core Services (AICS) database and recreates agent profiles from the Person data from Genesys Configuration Server for the Tenants that ASC monitors.
  - `false`—ASC uses the previously-stored agent profiles.
- Changes take effect: On restart

skip-groups

If this parameter set to `true`, ASC ignores all Configuration Server data about groups and events connected with updates to groups.

Set this option to `true` if the scoring request `action_filters` field contains only the skill expression filters and does not include filters by Agent Group names. ASC then skips reading Agent Group information from Configuration Server, which should significantly reduce ASC initialization time on start up.

- Configured in: Agent State Connector object, **[default]** section
- Default value: `false`
- Valid values: `true, false`
- Changes take effect: On restart
- Introduced in: 9.0.006.03

ss-custom-statistic-name

Use this option to specify the name of a custom statistic that ASC should read from Stat Server. By default, ASC subscribes for CurrentAgentState data.

To use this functionality, you must first configure the custom statistic in Stat Server before you can specify it as the value for the **ss-custom-statistic-name** option. Refer to Create a Custom Stat Server Statistic in the *Predictive Routing Deployment and Operations Guide* for complete instructions.

- Configured in: Agent State Connector object, **[default]** section
- Default value: `CurrentAgentState`
- Valid values: A string consisting of any valid custom statistic name
- Changes take effect: On restart

ss-subscription-timeout

Specifies a timeout, in milliseconds, between each subscription to avoid overloading Stat Server.

> **Important**
>
> - A series of Stat Server switchovers from primary to backup indicates that Stat Server is overloaded. If you see this pattern, increase the value of this option.

- To resolve this issue, you might also need to adjust the value of the **threads-max-size** option.

- Configured in: Agent State Connector object, **[default]** section
- Default value: `10`
- Valid values: Integers from `10`–`1000`
- Changes take effect: On restart

ss-monitoring-reconnect-count

Specifies the maximum number of reconnect attempts to Stat Server before ASC generates log event 60703, for which you should set an alarm. To be exact, if ASC detects a switchover or disconnection the number of times set in this option during the time period set in the **ss-monitoring-reconnect-min** option, ASC generates the log event. The cancel event for this alarm should be 60704.

- Configured in: Agent State Connector object, **[default]** section
- Default value: `4`
- Valid values: Integers from `1`–`10000`
- Changes take effect: On restart

ss-monitoring-reconnect-min

Specifies a time interval, in minutes, that ASC uses when monitoring multiple Stat Server switchover events. If ASC detects as many switchovers or disconnects as specified in the **ss-monitoring-reconnect-count** during the time period configured in this option, ASC generates log event 60701, for which you should set an alarm.

- Configured in: Agent State Connector object, **[default]** section
- Default value: `60`
- Valid values: Integers from `1`–`50000`
- Changes take effect: On restart

stat-srv-ws-conn

> **Important**
> This option has been removed in Agent State Connector (ASC) release 9.0.006.08 and higher. ASC now supports warm standby connections by default.

ASC can now establish a warm standby connection to a primary/backup Stat Server pair.

- Configured in: Agent State Connector object, **[default]** section
- Default value: `true`
- Valid Values:
  - `true`—ASC establishes a warm stand by connection to a primary and backup Stat Server (if a backup Stat Server is configured). On startup, ASC reads the connection parameters for the primary Stat Server from the ASC `Application` object.
  - `false`—ASC establishes a connection only to the primary Stat Server.
- Changes take effect: On restart

timebased-statistic-interval

Specifies the interval (in seconds) between statistic update requests to Stat Server for any statistics you have configured Agent State Connector (ASC) to monitor. See Configure ASC to Monitor Statistics for how to configure ASC and, if necessary, Stat Server. Use this option to ensure that you do not overload Stat Server. Environments with very large agent pools (30,000+ agents) might need to adjust the value of this option.

> **Important**
> In the initial release of this functionality (9.0.012.01), the only supported statistic is StatAgentOccupancy.

- Configured in: Agent State Connector object, **[default]** section
- Default value: `60` (seconds)
- Valid Values: Any positive integer
- Changes take effect: On restart
- Introduced In: 9.0.012.01

threads-max-size

Specifies the maximum number of threads used to subscribe for agent updates from Stat Server using a multithreading approach. Adjust the value for this option as needed in your environment to ensure that you do not overload Stat Server.

> **Important**
>
> - A series of Stat Server switchovers from primary to backup indicates that Stat Server is overloaded.
> - To reduce load on Stat Server *decrease* the value of this option.
> - To resolve this issue, you might also need to adjust the value of the **ss-subscription-timeout** option.

- Configured in: Agent State Connector object, **[default]** section
- Default value: `100`
- Valid Values: Any positive integer up to `2000`
- Changes take effect: On restart

[log] Section

Configure the following options in the **[log]** section:

standard
all
verbose

standard

Specifies the outputs to which an application sends the log events of the Standard level. The log output types must be separated by a comma when more than one output is configured. For example:

```
standard = stderr, network
```

- Configured in: Agent State Connector Application object, **[log]** section
- Mandatory: yes
- Default value: `stdout`
- Valid values: (string)

  - `stdout`—Log events are sent to the Standard output (`stdout`).
  - `stderr`—Log events are sent to the Standard error output (`stderr`).
  - `network`—Log events are sent to Message Server, which can reside anywhere on the network. Message Server stores the log events in the Log Database.
  - `memory`—Log events are sent to the memory output on the local disk. This is the safest output in terms of the application performance.
  - `<filename>`—Log events are stored in a file with the specified name. If a path is not specified, the file is created in the application's working directory.
- Changes take effect: Immediately

all

Specifies the outputs to which an application sends the log events of the `all` level. The log output types must be separated by a comma when more than one output is configured. For example:

```
all = stdout, logfile
```

- Configured in: Agent State Connector Application object, **[log]** section
- Mandatory: yes
- Default value: `stdout`
- Valid values: (string)
  - `stdout`—Log events are sent to the Standard output (`stdout`).
  - `stderr`—Log events are sent to the Standard error output (`stderr`).
  - `network`—Log events are sent to Message Server, which can reside anywhere on the network. Message Server stores the log events in the Log Database.
    Setting the **all** log level option to the `network` output enables an application to send log events of the Standard, Interaction, and Trace levels to Message Server. Debug-level log events are neither sent to Message Server nor stored in the Log Database.
  - `memory`—Log events are sent to the memory output on the local disk. This is the safest output in terms of the application performance.
  - `<filename>`—Log events are stored in a file with the specified name. If a path is not specified, the file is created in the application's working directory.
- Changes take effect: Immediately

verbose

Determines whether a log output is created. If it is, specifies the minimum level of log events generated. The log events levels, starting with the highest priority level, are Standard, Interaction, Trace, and Debug.

- Configured in: Agent State Connector Application object, **[log]** section
- Mandatory: yes
- Default value: `standard`
- Valid values: (string)
  - `all`—All log events (that is, log events of the Standard, Trace, Interaction, and Debug levels) are generated.
  - `debug`—The same as `all`.
  - `trace`—Log events of the Trace level and higher (that is, log events of the Standard, Interaction, and Trace levels) are generated, but log events of the Debug level are not.
  - `interaction`—Log events of the Interaction level and higher (that is, log events of the Standard and Interaction levels) are generated, but log events of the Trace and Debug levels are not.
  - `standard`—Log events of the Standard level are generated, but log events of the Interaction, Trace, and Debug levels are not.
  - `none`—No output is produced.
- Changes take effect: Immediately

Start and Stop All GPR Components

Start and Stop AICS

Starting and stopping AICS differs somewhat depending on whether you are running a single-server deployment or an HA deployment. Click the link for the appropriate procedure:

AICS Running on a Single Server

This section assumes that you have completed all prerequisite steps to deploy Docker and to unpack and install AICS. For instructions, see Deploy AI Core Services on a Single Host.

To start the AICS application, run the following commands:

```
$ cd IP_JOP_PRR_<version_number>_ENU_linux/scripts/
$ bash start.sh
```

The **start.sh** command does the following:

- Ensures that no previous instance of the application is running. If an instance is already running, the new start process shuts down.
- Exports all the required environment variables.
- Starts the mongodb container service.
  As soon as the mongodb service is available and running, **start.sh** starts the remaining AICS containers.

To stop the AICS application, run the following commands:

```
$ cd IP_JOP_PRR_<version_number>_ENU_linux/scripts/
$ bash stop.sh
```

The **stop.sh** command shuts down AICS, starting with the worker containers and progressing to the other containers after they have stopped.

To restart the AICS application, run the following commands:

```
$ cd IP_JOP_PRR_<version_number>_ENU_linux/scripts/
$ bash restart.sh
```

The **restart.sh** command performs the same functions as the **stop.sh** command followed by the **start.sh** command.

AICS Running in HA Mode

This section assumes that you have completed all prerequisite steps to deploy Docker, unpack and install AICS, and import the AICS Docker images to the various nodes in the Docker Swarm cluster. For instructions, see Deploy in High Availability Environments.

To start the AICS application, run the following commands:

```
$ cd IP_JOP_PRR_<version_number>_ENU_linux/ha-scripts/
$ bash start.sh
```

The **start.sh** command does the following:

- Ensures that no previous instance of the application is running in the Docker Swarm cluster. If an instance is already running, the new start process shuts down.
- Exports all the required environment variables.
- Validates the cluster before proceeding to start the included nodes.

To stop the AICS application, run the following commands:

```
$ cd IP_JOP_PRR_<version_number>_ENU_linux/ha-scripts/
$ bash stop.sh
```

The **stop.sh** command shuts down AICS on every node in the Docker Swarm cluster.

To restart the AICS application, run the following commands:

```
$ cd IP_JOP_PRR_<version_number>_ENU_linux/ha-scripts/
$ bash restart.sh
```

The **restart.sh** command performs the same functions as the **stop.sh** command followed by the **start.sh** command.

> ### Important
>
> - The **start.sh** and **restart.sh** commands should be executed on a node inside the Docker Swarm cluster that is running the mongodb service. If you run these commands on a node without the mongodb service, it fails and generates a helpful error message.
> - You can execute the **stop.sh** command on any node in the Docker Swarm cluster.

Autostart AICS

To auto-start the application cluster on server reboot add the following lines to the **/etc/rc.d/rc.local** file:

```
# Adds execution permissions to rc.local
sudo chmod u+x /etc/rc.d/rc.local

# Makes sure docker has stated
sudo echo "sleep 10" >> /etc/rc.d/rc.local

# starts the containers on boot
sudo echo "bash /home/pm/IP_JOP_PRR_<version_number>_ENU_linux/scripts/start.sh" >>/etc/rc.d/
rc.local
```

Restart the server and verify that the application is running.

Note: You may need to change the parameters passed to `start.sh`.

Start and Stop ASC

All Agent State Connector instances can be started and stopped from Genesys Administrator. For instructions on starting and stopping from Genesys Administrator, see the System Dashboard topic in the *Genesys Administrator Help*.

To stop ASC manually from the command line, use the appropriate one of the following procedures:

Start ASC Manually on Linux

To start ASC from the command line, open a terminal window on the host machine and enter the following commands:

```
cd <agent state="" connector="" working="" directory=""><br />./AgentStateConnector.sh -host
<host_name> -port <port_number> -app
<application_name></application_name></port_number></host_name></agent>
```

You can start the Agent State Connector by entering `./run.sh`.

Start ASC Manually on Windows

To start ASC from the command line:

- Open a terminal window on the host machine and run the **`AgentStateConnector.bat`** script.

To start ASC deployed on Windows host from Genesys Administrator:

- Configure on the "Server Info" tab the following parameters:
  - Command Line: `<path to="" jdk="" bin="" folder="">\java.exe</path>`
  - Command Line Arguments: `-jar agent_state_connector-<version>-jar-with-`

```
dependencies.jar -host <host_name> -port <port_number> -app
"<application_name>"</application_name></port_number></host_name></version>
```

- Start the ASC using "Start" button on the application page.

Stop ASC Manually on Linux

If you need to stop ASC from the host using the command line, enter the following:

```
<br />ps ax | grep <asc_application_name><br />kill -9 <shell_script_pid><br />kill -9
<pid_of_java_process_running_asc></pid_of_java_process_running_asc></shell_script_pid></asc_application_name>
```

> **Important**
>
> For an emergency stop, follow the same procedure given here.

Stop ASC Manually on Windows

If ASC is running as an application—not as a Windows Service—stop it using the following procedure.

- From the application's console window, press **CTRL+C**.
- From Genesys Administrator, use "Stop" button on the ASC application page.

Start and Stop Use of the Strategy Subroutines

To turn on Predictive Routing in your routing strategy:

1. Open the Predictive_Route_CfgData Transaction List object.
2. Set the **prr-mode** option to any value *except* `off` in all sections that define predictors, and also in the **[default-predictor]** section.

To turn off Predictive Routing in your routing strategy:

1. Open the Predictive_Route_CfgData Transaction List object.
2. Set the **prr-mode** option to `off` in all sections that define predictors, and also in the **[default-predictor]** section.

How Does GPR Score Agents?

GPR scores agents based on the data uploaded to the Agent and Customer Profiles or, if you are using the GPR API, on data passed in the API score request as part of the context parameter. If both are present, data from the API request takes priority over data from the Agent and Customer Profiles.

This topic explains how GPR handles various scoring scenarios, depending on your environment and your configuration settings.

Default Agent Scores

If an agent belongs to the target Agent Group but GPR does not score the agent, the isAgentScoreGood and ScoreIdealAgent subroutines assign a score for that agent according to the value set for the **default-agent-score** configuration option.

For agents who have a default score assigned, the following KVPs reflect that value:

- gpmAgentScore, which records the value specified in the **default-agent-score** option if the agent who handled the interaction had the default score. If the the AICS scoring engine calculated a score for the agent, gpmAgentScore reports the calculated score value.
- gpmDefaultAgentScore, which records the value specified in the **default-agent-score** option.
- gpmDefaultScoreUsed, which indicates whether the selected agent was assigned the default score.
- gpmDefaultScoredAgents, which records the number of agents assigned the default agent score.

Example 1

Agents A and B log in after the scoring request is made and are each assigned the default score, which is 40. Agent A receives the interaction. The related KVPs have the following values:

- gpmAgentScore = 40
- gpmDefaultAgentScore = 40
- gpmDefaultScoreUsed = 1
- gpmDefaultScoredAgents = 2

Example 2

GPR assigns Agent C a score of 80. The default score is 40. No agents are assigned the default score. The related KVPs have the following values:

- gpmAgentScore = 80
- gpmDefaultAgentScore = 40
- gpmDefaultScoreUsed = 0
- gpmDefaultScoredAgents = 0

Score Adjustment

The GPR subroutines enable you to adjust agent scores using an occupancy factor when URS sorts them. You control the agent occupancy setting in the **agent-occupancy-factor** configuration option. Scores adjusted using an agent occupancy factor are recorded in the gpmAdjustedAgentScore KVP.

Example

GPR returns a score of 80 for Agent A. The **agent-occupancy-factor** option value is `0.5`. If this agent selected to receive the interaction, the agent score KVPs have the following values:

- gpmAdjustedAgentScore = 40
- gpmAgentScore = 80

> **Important**
> This adjusted score is used only for sorting the agent scores in the ScoreIdealAgent subroutine. The adjusted agent score is used in the isAgentScoreGood subroutine to compare the agent score with the configured threshold. The actual returned score is used.

Threshold Scores

To implement the agent holdout feature, GPR checks the score returned for the agent against the threshold value configured in the **score-based-threshold** option. URS calls the isAgentScoreGood subroutine to suppress routing to an agent who is in ready state if this agent does not provide an acceptable match for the interaction. This is used in conjunction with relaxation thresholds to target better-matched agents preferentially, expanding the pool of agents if the best-matched agents are unavailable.

- See Agent Holdout Options for the complete list of options used for agent holdout and threshold settings.

Score Relaxation Timeouts

By design, URS checks the threshold relaxation ("awakens") at two-second intervals. As a result, the minimum *real-world* value for **threshold-relaxation-timeout** is 2 because the threshold relaxation is checked only every two seconds. Even though the default value for the **threshold-relaxation-timeout** option is 1, URS applies the threshold relaxation only at two-second intervals.

When the **initial-threshold-timeout** value has elapsed, the minimum score required to allow an agent to handle the interaction is reduced by the configured relaxation step. This relaxation step can be applied multiple times, depending on the option settings you specify.

Example 1

GPR returns a score of 50 for Agent A. The threshold and relaxation options have the following values:

- score-based-threshold = 55
- initial-threshold-timeout = 2
- threshold-relaxation-timeout = 4
- threshold-relaxation-step = 4

Agent A is selected after 6 seconds. The related KVPs have the following values:

- gpmAgentScore =50
- gpmInitialScoreThreshold = 55
- gpmFinalScoreThreshold = 47

URS attempts for Agent A	Threshold Value	Result
Interaction queued and scoring completed in the same second	55 (initial threshold value)	agent score (50) < threshold (55)
after 2 seconds	51 (first relaxation applied, 55-4)	agent score (50) < threshold (51)
after 4 seconds	51 (no change from previous step)	agent score (50) < threshold (51)
after 6 seconds	47 (second relaxation applied, 51-4)	agent score (50) > threshold (47); interaction routed to agent

Example 2

GPR returns a score of 30 for Agent B. The threshold and relaxation options have the following values:

- score-based-threshold = 40
- initial-threshold-timeout = 5
- threshold-relaxation-timeout = 2
- threshold-relaxation-step = 5

Agent A is selected after 8 seconds. The related KVPs have the following values:

- gpmAgentScore =30
- gpmInitialScoreThreshold = 40
- gpmFinalScoreThreshold = 30

URS attempts for Agent B	Threshold Value	Result
Interaction queued and scoring completed in the same second	40 (initial threshold value)	agent score (30) < threshold (40)
after 2 seconds	40 (no change from previous step)	agent score (30) < threshold (40)
after 4 seconds	40 (no change from previous	agent score (30) < threshold (40)

URS attempts for Agent B	Threshold Value	Result
	step)	
after 6 seconds (initial timeout is 5 seconds, but relaxation is applied only when URS awakens)	35 (first relaxation applied, 40-5)	agent score (30) < threshold (35)
after 8 seconds	30 (second relaxation applied, 35-5)	agent score (30) = threshold (30); interaction routed to agent

Example 3

GPR returns a score of 35 for Agent C. The threshold and relaxation options have the following values:

- score-based-threshold = 40
- initial-threshold-timeout = 5
- threshold-relaxation-timeout = 1 (no value specified, default value used)
- threshold-relaxation-step = 1 (no value specified, default value used)

Agent C is selected after 10 seconds. The related KVPs have the following values:

- gpmAgentScore =35
- gpmInitialScoreThreshold = 40
- gpmFinalScoreThreshold = 34

URS attempts for Agent C	Threshold Value	Result
Interaction queued and scoring completed in the same second	40 (initial threshold value)	agent score(35) < threshold (40)
after 2 seconds	40 (no change from previous step)	agent score (35) < threshold (40)
after 4 seconds	40 (no change from previous step)	agent score (35) < threshold (40)
after 6 seconds (initial timeout is 5 seconds, but relaxation is applied only when URS awakens)	38 first and second relaxation applied: • first relaxation after initial 5 seconds • second relaxation after next 1 second	agent score (35) < threshold (38)
after 8 seconds	36 (third and fourth relaxations applied, 38 - 2)	agent score (35) < threshold (36)
after 10 seconds	34 (fifth and sixth relaxations applied, 36 - 2)	agent score (35) > threshold (34); interaction routed to agent

How the Availability Status of Agents in the Target Agent Group is Determined

The source for agent availability information depends on your release of Agent State Connector (ASC).

Agent State Connector release 9.0.015.04 and higher

The connection between ASC and Stat Server is optional. If you do not specify a Stat Server in the ASC `Application` object **Connections** tab, URS provides agent availability information. It checks with Stat Server on the agent login status of the specified target group before making the scoring request, and adds the list of matching agents in the request field 'action_filters'.

The GetActionFilters subroutine reads the login statuses specified to be available for routing from the **login-status-expression** configuration option.

- To use the new functionality, set the value of the **use-action-filters** configuration option, introduced in Ai Core Services release 9.0.015.03, to `false` (the default value is `true`, which maintains the same functionality as in previous releases).

Sample scoring request showing a list of agents employee IDs in the field `action_filters`, where `POC0x` strings indicate IDs of agents with a required login status:

```
{
"token":"<api_token>",
"format_as_map":"true",
"context_id":"3600",
"log_request":"true",
"action_filters":"employeeId in [\"POC01\",\"POC02\",\"POC03\",\"POC04\"]",
"context":
{
    "PR_TYPE":"Gold",
   "PR_LANG":"French"
}
}
```

Important

- This architecture increases the load on URS by approximately 15%. Use the Sizing Guide to verify that you have sufficient URS bandwidth available.
- Only alphanumeric characters, spaces, and underscores are supported in the names of Stat Server `Application` objects. Names including other special characters cause a malformed scoring request.

Agent State Connector release 9.0.015.01 and lower

All agent status changes (such as login, logout, ready, and so on) are synced to AICS by ASC, which receives the status updates from Stat Server. When a scoring request is made, the target group

names include the login statuses (expression) that mark an agent as available for routing in the action filters field. GPR returns the scores only for the agents present in the target group that matching the specified login status.

Sample scoring request showing the use of action filters with a skill expression and a login status expression, where all agents with the skill 'SkillFrench', having a skill level higher than 10, and who are currently logged in, are selected:

```
{
"token":"<api_token>",
"format_as_map":"true",
"context_id":"3600",
"log_request":"true",
"action_filters":"Skill_French>10 & ((loginStatus>1 & loginStatus<23) | loginStatus>23)",
"context":
{
    "PR_TYPE":"Gold",
   "PR_LANG":"French"
}
}
```

(Optional) Store Scoring Data in the GPR Log File

You can configure the GPR subroutines to log the interaction context and the scoring response details to the GPR API when the **log-to-api** configuration option is set to `true`.

> ### Important
>
> Score log functionality requires the following releases of GPR components:
>
> - AICS 9.0.015.03 and higher
> - URS Strategy Subroutines 9.0.015.00 and higher

With two exceptions, the same values are stored in the score log and in the Genesys Info Mart tables. The exceptions are the following, which appear in the score log but not in the Info Mart database:

- gpmAgentID - Genesys Info Mart requires only the Agent DBID to retrieve detailed agent information during aggregation.
- ConnID - Genesys Info Mart requires only the CallUUID to retrieve detailed interaction information, including the ConnID, during aggregation.

To support real-time reporting in Genesys Pulse, all GPR KVPs are added in two places:

- In user data, which is in the outer body of the score log request.
- In the context field of the score log request, along with other user data typically included in the context field.

Integrate with Genesys Reporting includes a complete list of all KVPs stored in the Genesys Info Mart database.

Configure GPR to Log Scoring Details

To enable score logging, configure the following:

1. Add the following environment variables to the AICS **tango.env** file:
   - LOGS_COLLECTOR_ENABLED = `True`
   - SCORE_LOG_BACKUPCOUNT = `30` (The maximum number of backup copies of the compressed log files.)
   - SCORE_LOG_FILENAME = `/var/log/gpr/supportability-tool-logs/score_logs.log` (The path where the score log files are kept. The log file is mounted to **/datadir/supportability-tool-logs/** on the host running the Tango container.)
2. Restart the Tango container.

Clean Up Scoring Logs

GPR doesn't remove score logs automatically or clean up the score logs associated with a Predictor when that Predictor is deleted. To delete unneeded score logs from MongoDB, AI Core Services release 9.0.015.03 and higher includes the **/scr/gpr/scripts/clean_score_logs.py** script, which is located in the tango container. If you are running a release lower than 9.0.015.03, you can copy the script into the **/scripts** directory and run it successfully.

The commands required to run the **clean_score_logs.py** script depend on your version of AICS:

For release 9.0.013.01 and higher, run the script using the following commands:

```
$ docker exec -ti tango /bin/bash
$ cd /src/gpr/scripts
$ MODE=prod python3.6 clean_score_logs.py <parameters>
```

For release 9.0.012.01 and lower, run the script using the following commands:

```
$ docker exec -ti tango /bin/bash
$ cd /src/gpr/scripts
$ MODE=prod python clean_score_logs.py <parameters>
```

In an HA deployment, execute the script on any node in the Docker Swarm cluster that is running the mongodb service.

Determining the Correct Parameters for the clean_score_logs.py Script

The parameters to be passed with the script to clean up unneeded score logs depend on how the score log is structured.

- When you pass an API POST /score_log request that includes a Predictor ID, GPR stores the score log in a collection named using the following format:

`scorelog_<predictor_id><predictor_name_without_spaces-or_tabs>`.
To delete such a collection, run the script *without* parameters. The script automatically detects collections that are left after the corresponding predictors were deleted and deletes them.

- When you pass an API POST /score_log request that does *not* include a Predictor ID, GPR stores the score log in a *default* account-specific collection named `scorelog_default_nopredictor_<account_id>`. See Deleting From Default Score Log Collections (below) for specific instructions.
- The optional `--dry_run` parameter instructs the script not to clean up the score log files, but only to display how many collections/records are going to be deleted.

Deleting From Default Score Log Collections

The exact format of the default score logs depends on the Universal Routing Server (URS) strategy configuration. To remove score logs for deleted Predictors from the default collection, define the path to the Predictor ID in the score log body structure and pass it to the script as a separate parameter. If you do not specify the path, the script cannot automatically detect which Predictor data to delete.

The following examples show different score log structures in the default collection.

- Note that the examples assume you are using AICS 9.0.013.01 or higher. If you are running an earlier version, adapt the commands as explained at the beginning of the Clean Up Score Logs section (above).

1. To delete logs formatted as in Example 1 and Example 2 (below) run the script with the parameter `--predictor_key=p_id`, as shown in the following command:

   ```
   python3.6 src/gpr/scripts/clean_score_logs.py --predictor_key=p_id
   ```

2. To delete logs formatted as in Example 3 (below) run the script with the parameter `--predictor_key=context.gpmPredictorId`, as shown in the following command:

   ```
   python3.6 clean_score_logs.py --predictor_key=context.gpmPredictorId
   ```

Example 1

```
{
    "_id" : ObjectId("5d0c94e341c24c30bdc970ff"),
    "param" : "1",
    "p_id": "5ae0cb37e2d22d221dc07c5b",
    "created_at_slog" : ISODate("2019-06-21T08:27:15.722Z")
},
```

Example 2

```
{
    "_id" : ObjectId("5d0c94e341c24c30bdc97101"),
    "param" : "2",
    "p_id": "5ae0cb37e2d22d221dc07c5b",
    "created_at_slog" : ISODate("2019-06-21T08:33:12.107Z")
},
```

Example 3

```
{
    "context_id" : "3600",
    "prrPredictor" : "",
    "context" : {
```

```
        "gpmMedianScore" : "9",
        "last_name" : "A",
        "gpmPredictorId" : "5ad6802b8615b3002c8985d0",
        "gpmAgentScore" : "0",
        "sex" : "M",
        "seniority" : "NEW",
    },
    "created_at_slog" : ISODate("2019-05-17T08:20:11.503Z
}
```

Sample Score Log Output

The following sample shows what you might receive in response to a request for score log details from the API. For details, see the *Predictive Routing API Reference*. (Requires a password for access. Please contact your Genesys representative if you need to view this document.)

```
{
"prrPredictor": "manual_test",
 "gpmMode": "prod",
 "gpmStatus": "agent-surplus",
 "gpmPredictor": "manual_test",
 "gpmPredictorId": "5cb07deeb3fc5800397fb8f4",
 "gpmModel": "manual_model",
 "gpmModelId": "5cb07e11c0eb2f00353b6151",
 "gpmResult": "1",
 "gpmGlobalScore": "0",
 "gpmMedianScore": "21",
 "gpmMaxScore": "72",
 "gpmMinScore": "9",
 "gpmTargetSize": "4",
 "gpmUse": "1",
 "gpmCustomerFound": "1",
 "CALLID": "01VI5E6TOSED503B16OAHG5AES000004",
 "START_TS": "1559063048",
 "gpmRouteAttemptId": "1",
 "context_id": "3600",
 "gpmAgentScore": "72",
 "gpmAgentRank": "1",
 "gpmScoreAboveMedian": "1",
 "gpmPredictorType": "Service",
 "gpmDefaultAgentScore": "21",
 "gpmDefaultScoredAgents":"4",
 "gpmDefaultScoreUsed": "0",
 "gpmInitialScoreThreshold": "50",
"gpmFinalScoreThreshold": "50",
 "gpmAdjustedAgentScore": "72",
 "gpmSuitableAgentsCount": "2",
 "gpmGlobalScoreCount": "0",
 "gpmAgentID" : "POC01",
 "ConnID": "006c02dcefa47049"
 "gpmAgentDBID": "104",
 "gpmWaitTime": "6",
 "context":
 {
       "PR_TYPE": "Gold",
       "PR_LANG": "French",
       "RPVQID": "0016FD27SOEDB5861EOAHG5AES000004"
 }
}
```

Integrate with Genesys Routing

Genesys Predictive Routing (GPR) provides two packages of subroutines that integrate into your Genesys routing environment. They perform the agent scoring and best-match selection functions, enabling you to fine-tune your routing to take advantage of the rich data on agents, customers, and interactions available to you in your environment.

- To understanding routing interaction flows and how interactions are selected using GPR, see Routing Scenarios Using GPR.
- The two subroutines options are the following:
  - URS Strategy Subroutines - for environments routing with Universal Routing Server (URS) and Interaction Routing Designer (IRD).
  - Composer Subroutines - for environments designing routing flows with Composer/Orchestration Server (ORS).

> **Important**
>
> The Composer Subroutines extend the functionality of the URS Strategy Subroutines so that you can use Composer and ORS to manage your routing. However, they are a wrapper around the URS Strategy Subroutines, not an entirely separate set of subroutines. As a result, *they require that you also install URS and IRD*.

Additional Information

- For a description of the Designer routing block for Predictive Routing and how to configure it, see Predictive Routing Block (Genesys Engage cloud users only).
- If you would like to evaluate Genesys Predictive Routing for use with schedule-based routing (using Genesys Workforce Management), service-level routing, or business-objective routing, contact Genesys Professional Services for a review of your routing environment.

Routing Scenarios Using GPR

To deploy the GPR subroutines, you modify your IRD strategies or Composer workflows o incorporate them. Rather than picking the Agent with required skills who has been available longest, or using simple Agent Group routing, Predictive Routing predicts the best results for a specific interaction, based on customer intent or other relevant information.

This topic presents the following information:

- A high-level view of a Predictive Routing interaction flow
- How the URS Strategy Subroutines work together to score agents and identify a routing target
- Routing scenarios using Predictive Routing, explaining how URS ranks agents by score

> **Important**
>
> - If you would like to evaluate Genesys Predictive Routing for use with schedule-based routing (using Genesys Workforce Management), service-level routing, or business-objective routing, contact Genesys Professional Services for a review of your routing environment.
> - If your environment uses multiple URS instances receiving interactions from a single T-Server, the only criterion used to select the next interaction for routing is priority.
> - This topic assumes that you are using a virtual agent group (VAG) as the target for your routing. If you route using a skill expression to identify your targets, convert it to a VAG string expression using the IRD functions MultiSkill or CreateSkillGroup before passing the resulting string as an argument to the ActivatePredictiveRouting_v3 subroutine. See Using Agent Skills for Ideal Agent Selection in the *Supplement to the Universal Routing 8.1 Reference Manual* for more information.

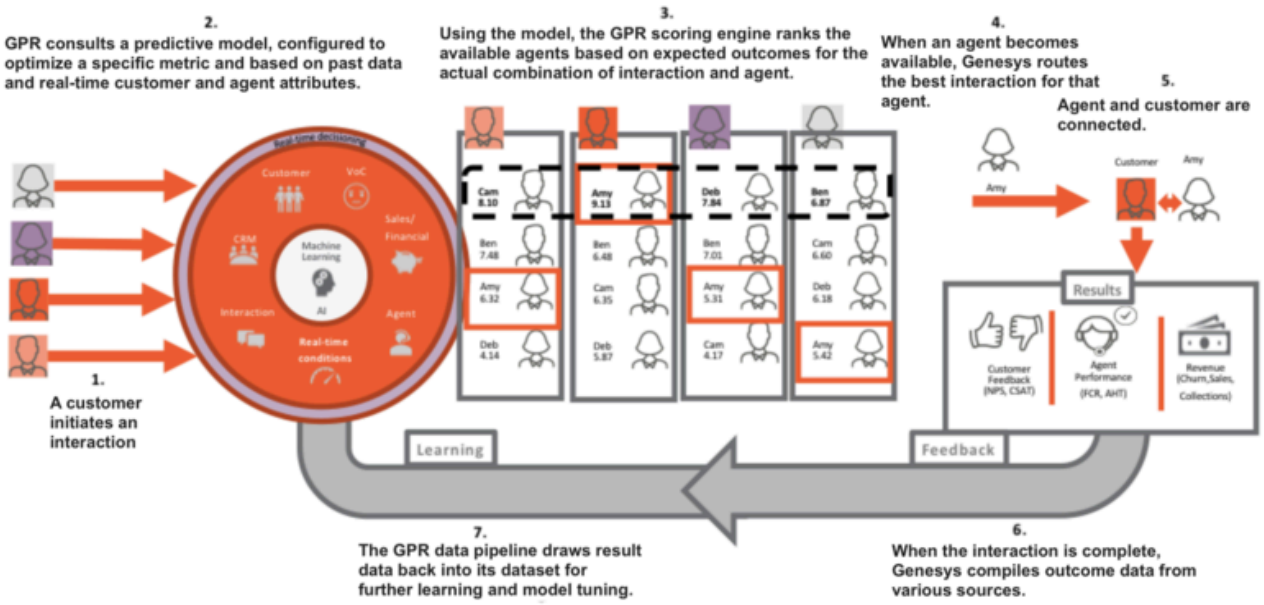
High-Level Predictive Routing Interaction Flow

The graphic below shows a very general interaction flow using Predictive Routing. Refinements to the flow depend greatly on details of your environment. Key aspects that differ in various environments:

- Your data - That is, the interaction types supported and the applications that might have relevant information. Genesys Info Mart is a key data source, but CRM systems and other applications in your environment can also provide important data. See Prepare Your Data for more information.
- Your pre-routing data flow - This depends on the interaction type and the exact architecture in your environment. For example, is this a chat interaction or a call? Do you use an IVR, and if so, what information do you attach?
- The Genesys routing solution you are using - Predictive Routing supports routing with IRD/URS and with

Composer/ORS/URS.

- Your reporting solution for Predictive Routing - Whether you are using GCXI, Genesys Pulse, or another solution to present the data stored in Genesys Info Mart.



How the Strategy Subroutines Work

The following sequence provides a basic overview of the way the various GPR subroutines work together to evaluate agent scores and determine the best match given the currently available agents and the currently waiting interactions.

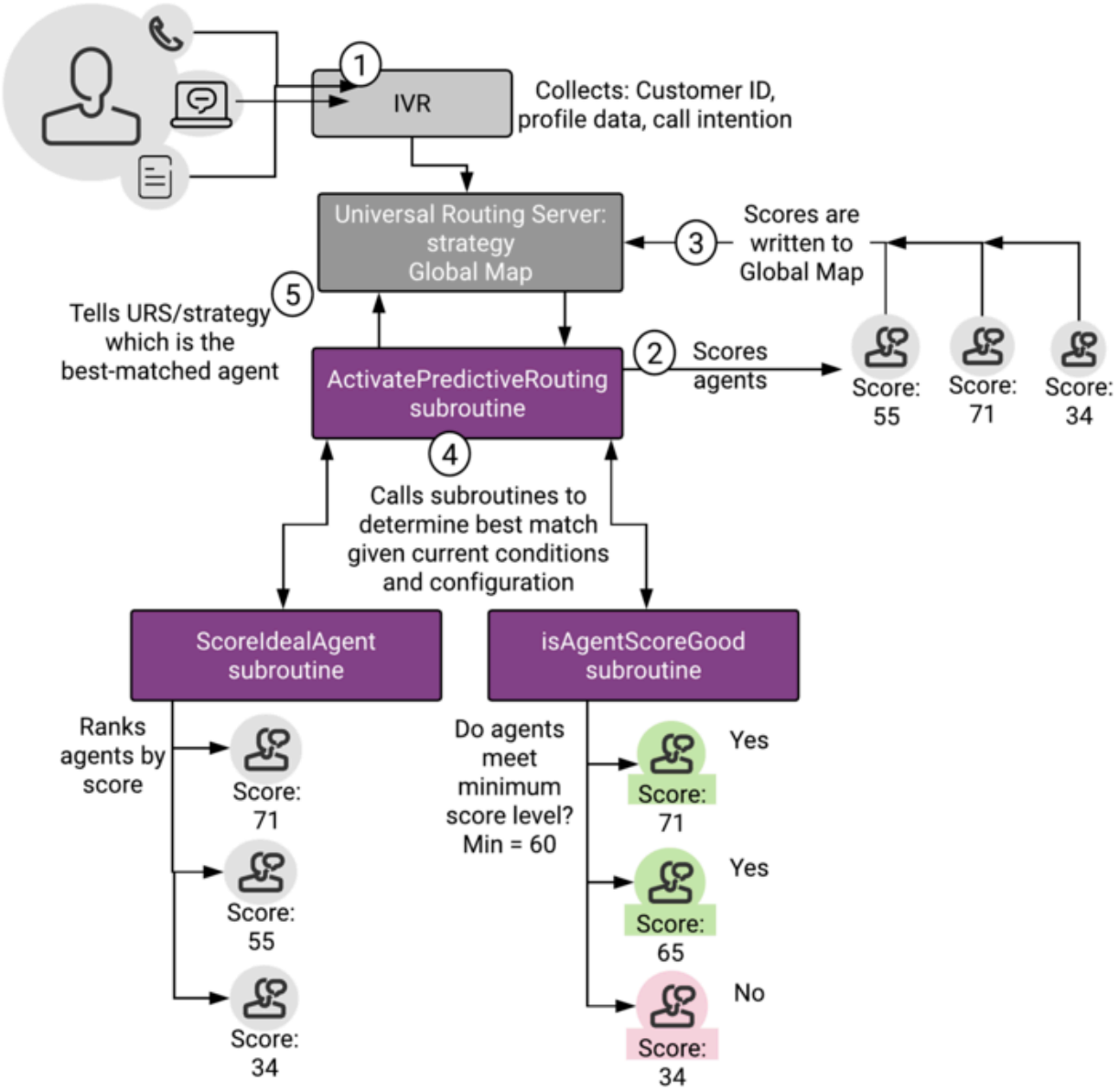
1. ActivatePredictiveRouting retrieves agent scores are retrieved from AI Core Services via REST API request and stores them in the global map in URS memory. The name of the map is the interaction ConnectionId (the original ID, if the interaction is a consult). This map contains pairs of agent employee IDs as the keys and their scores for the interaction as values. ActivatePredictiveRouting calls the SetIdealAndReadyCondition subroutine for further interaction processing.
2. SetIdealAndReadyCondition processes the different modes of Predictive Routing. It calls the SetIdealAgent IRD function to schedule the execution of the URS callback subroutines. It calls the ScoreIdealAgent subroutine to facilitate interaction queueing according to their scores, and calls SetReadyCondition (if enabled) to call the isAgentScoreGood subroutine.
   The parameters for these callback subroutines are retrieved and verified before any URS request is invoked to enable the callbacks.
3. After establishing a list of potential targets based on the target expression (Skill, Agent Group, and so on) SetIdealAndReadyCondition then executes the ScoreIdealAgent callback subroutine.
4. ScoreIdealAgent retrieves the scores for the potential target agents from the global map set in Step 1.
5. When an agent becomes ready, URS executes the isAgentScoreGood subroutine to determine whether

that target is acceptable. If you enabled agent hold-out, URS executes the isAgentScoreGood subroutine when an agent becomes ready, which determines whether that agent reaches the specified threshold score. If not, URS waits for a configured timeout period, then checks whether any agent now satisfies the adjusted threshold value. See How Does GPR Score Agents? for a detailed discussion of how agent hold-out routing works.

6. Once the isAgentScoreGood subroutine locates an available agent who scores above the current threshold, it sends the target details to URS to initiate routing.
7. URS calls the GPRIxnCompleted subroutine as a custom step from a Routing Block in the strategy. It collects the Predictive Routing outcome for the successfully routed interaction (the DBID of the agent to whom it was distributed, the score of the agent, other interaction statistics relevant for the Predictive Routing performance) and prepares the user data for Predictive Routing reporting.
8. URS calls the GPRIxnCleanup subroutine from both the success and failure exits from the Routing block, or if the interaction is abandoned. The purpose of the subroutine is to publish the Predictive Routing reporting user data and to clean up the ScoreIdealAgent and isAgentScoreGood callback subroutines. GPRIxnCleanup publishes reporting data in two ways:
   - It sends a UserEvent containing the user data relevant for Predictive Routing to T-Server/SIP Server, from which is enters the Genesys historic reporting solution flow.
   - It can submit the same data AI Core Services via REST API request where it is stored in the score_log and can be retrieve using an API request.

Important

If your routing strategy uses the SelectDN and SuspendDN IRD functions instead of a Routing Block, consult with Genesys Professional services about how the ActivatePredictiveRouting, GPRIxnCompleted and GPRIxnCleanup subroutines can be integrated into your strategy.



Routing Scenarios Using Predictive Routing

When you are using Predictive Routing to route interactions, there are two main scenarios that affect how this matching plays out:

- **Agent Surplus** - There are relatively few interactions, which means there could be a number of high-score agents available. You can configure a minimum threshold so that, if the agents available are not

very highly ranked, the strategy keeps the interaction in queue until a better-scoring agent becomes available.

- **Interaction Surplus** - There are many interactions, so that most agents are busy and it might be more difficult to find an ideal agent for each interaction. In such a scenario, you can have agents matched to the interaction for which they have the highest probability of getting a positive result.

Agent Surplus Flow

In this case there are agents logged in and in the Ready state who can respond to interactions immediately. From a Virtual Agent Group that is defined by skill expression, URS first tries to route an interaction to an agent with the best score, using the following process to match agents and interactions:

1. An interaction arrives at the routing strategy, which has a target group of agents.
2. The ActivatePredictiveRouting subroutine sends a request to the Predictive Routing scoring server via HTTP request.
3. Predictive Routing returns scores for each agent in the target group based on the criteria you selected in the active model.
4. The ActivatePredictiveRouting subroutine updates a global cache in URS memory, which keeps agent scores for all interactions. When URS tries to route the current interaction to the agent group, it sorts the agents according to their scores, in descending order, and routes to the agents with the best score first.

When URS takes an interaction from the queue:

1. URS calls the ScoreIdealAgent subroutine, which reads the agent scores in the target group from global map and ranks the agents by score.
2. URS calls the IsAgentScoreGood subroutine, which selects the available agent with the highest score, assuming the agent has a score high enough to be selected for this interaction.
   In an agent-surplus scenario, it is typically not a problem to route to an agent with a good score. For scenarios where this is not the case, see **Interaction Surplus Flow**, below.
3. URS calls the PrrIxnCompleted subroutine, which updates user data with the scoring result for storage in Genesys Info Mart.
4. URS calls the PRRLog macro, which logs the result in the URS log file.

Interaction Surplus Flow

This scenario covers situations when all agents are already busy handling interactions and new interactions are queued. When one of the agents becomes ready, the system selects the interaction for which the agent has the best score. This is not necessarily the interaction that has been in the queue longest.

When interactions are waiting, URS uses a number of criteria to decide the order in which it directs the interactions to the best target. In general, URS uses the following hierarchy:

1. Interaction priority.

2. Best agent score.

> **Important**
>
> In scenarios where both scored and unscored interactions might have the same priority, scoring is disregarded for all the interactions and the selection is based on the next differentiating criterion, time.

3. Time in queue, which can be based on age of interaction or time in queue and can incorporate predicted wait time.
4. Interaction ID (URS selects the interaction with the lowest—oldest—ID). This is a rarely-used "tie-breaker" criterion.

Using gpmStatus and gpmSuitableAgentsCount to Monitor Your Routing

gpmStatus and gpmSuitableAgentsCount are KVP values written in the Genesys Info Mart database when an interaction is routed using the GPR subroutines. (You can also retrieve the values by using the GPR API to query the score log.)

- gpmStatus indicates whether there was an agent-surplus or an interaction-surplus condition when the interaction was routed.
- gpmSuitableAgentsCount indicates the number of agents who have scores returned from AICS greater than or equal to the initial threshold value when the scoring response is received. If gpmSuitableAgentsCount is `0`, then no agents have eligible scores compared with the threshold value, so the interaction must wait for a higher-scoring agent to become available or for the next threshold relaxation step

These KVP values, when analyzed for different interactions over a representative day or week period can help you understand your contact center traffic and GPR performance. The following table indicates certain scenarios and how to interpret them.

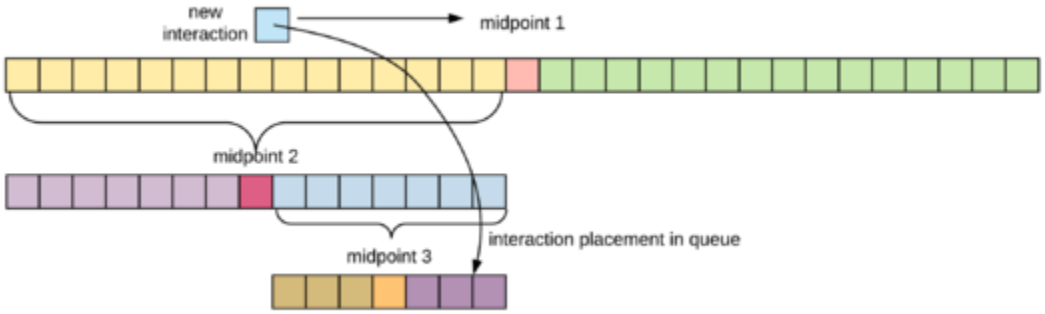
KVP Values	Inference
gpmStatus = caller-surplus gpmSuitableAgentsCount > 0	Your GPR Model is returning useful scores with relation to the configured routing threshold, but agent staffing is not adequate to produce satisfactory wait times.
gpmStatus = agent-surplus gpmSuitableAgentsCount > 0 or consistently a very small number	Analyze why the scores GPR returns are not meeting the configured threshold. You might need to retrain your Model, adjust the scoring expression, or reduce the threshold level.

How Interactions are Sorted within the Queue

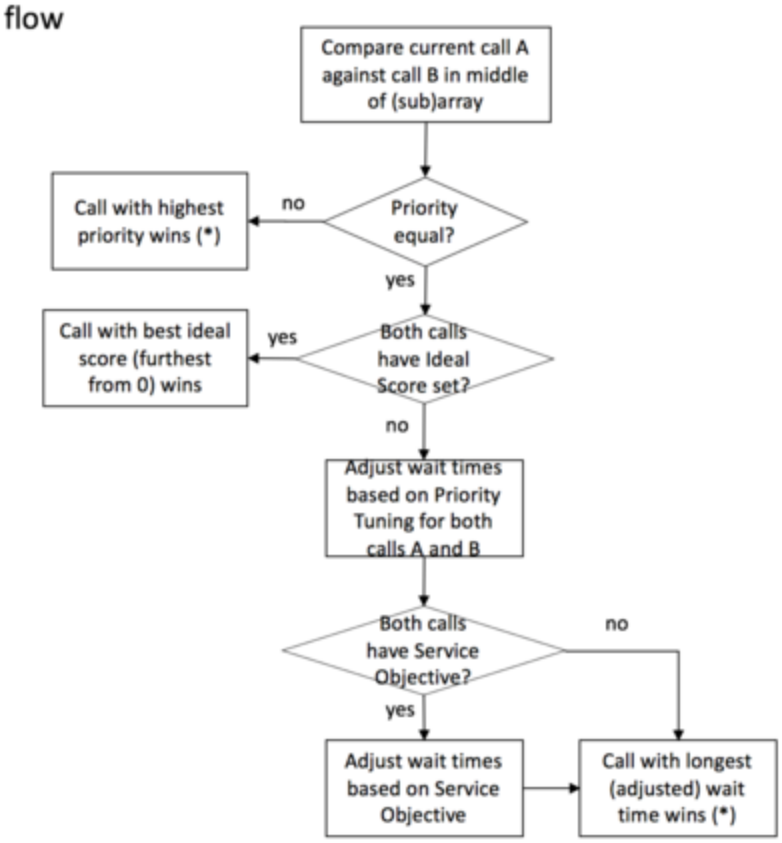
As each interaction comes in, it is scored, and then assessed relative to the interaction at the midpoint of the existing array of interactions. Does it come before or after the mid-point?

> **Important**
>
> The order in which interactions are prioritized is called an *array* here. This is not equivalent to a queue. These interactions might be from multiple queues, each of which is submitting interactions for URS sorting and routing.

After this decision, URS compares the new interaction against the midpoint within the selected region. Each time URS evaluates the interaction, it is assigned to a smaller region with the total array, always relative to the midpoint of the previous region.



The sorting decision tree:



Example 1

Three calls arrive at a contact center:

- C1 - priority = 1, agent score = 0.3, timestamp = 0:00, URS ID = 1
- C2 - priority = 1, agent score = n/a, timestamp = 0:05, URS ID = 2
- C3 - priority = 1, agent score = 0.6, timestamp = 0:10, URS ID = 3

1. C1 arrives first and is placed into the empty array.
2. C2 arrives. URS compares it with the middle (in this case, only) call in the array, C1.
   The priority is equal and, because C2 has no agent score, URS moves to the next decision criterion.
3. C2 has a shorter wait time, so is put behind C1.
   With only two calls in the array, no further comparison is needed.
4. C3 arrives. URS compares it against the "middle" entry of the array, C1.
   The priority is equal. C3 has better score (further from 0), so URS puts it in front of C1.

Outcome: C3, C1, C2 (the example assumes that interactions are taken from the left end of the array)

Example 2

Five calls arrive at a contact center and are placed in either the Predictive Routing queue or a

conventional queue:

- C1 - priority = 1, agent score = n/a, timestamp = 0:00, URS ID = 1
- C2 - priority = 1, agent score = 0.5, timestamp = 0:05, URS ID = 2
- C3 - priority = 1, agent score = n/a, timestamp = 0:10, URS ID = 3
- C4 - priority = 1, agent score = 0.75, timestamp = 0:15, URS ID = 4
- C5 - priority = 1, agent score = 0.95, timestamp = 0:20, URS ID = 5

1. C1 arrives first. URS places it into the empty array.
2. C2 arrives. URS compares it with the middle (in this case, only) call in the array, C1.
   The priority is equal and, because C1 has no agent score, URS moves to the next decision criterion.
3. C2 has a shorter wait time, so is put behind C1. (In this example,
   Current order: C1 C2 (the example assumes that interactions are taken from the left end of the array)

   With only two calls in the array, no further comparison is needed.
4. C3 arrives. URS compares it against the "middle" entry of the array, C2.
   The priority is equal and, because C3 has no agent score, URS moves to the next decision criterion.
5. C3 has a shorter wait time, so is put behind C2.
   Current order: C1 C2 C3
6. C4 arrives. URS compares it against the middle entry of the array, C2.
   The priority is equal. C4 has a better score (further from 0), so URS places it before C2.
7. Now URS must determine whether C4 should be before or after C1, which is also before C2.
   The priority is equal and, because C3 has no agent score, URS moves to the next decision criterion.
8. C4 has a shorter wait time (a more recent timestamp), so URS places it behind C1.
   Current order: C1 C4 C2 C3
9. C5 arrives. URS compares it against the "middle" entry of the array, C2.
   The priority is equal. C5 has a better score (further from 0), so URS places it before C2.
10. Now URS must determine whether C5 should be before or after C4, the "middle" call in the section of the array before C2.
    The priority is equal. C5 has a better score, so URS places it before C4.
11. Now URS must determine whether C5 should be before or after C1.
12. C5 has a shorter wait time, so URS places it behind C1.
    Final order: C1 C5 C4 C2 C3

Using Agent Hold-Out

Agent hold-out enables you to have an interaction wait a specified time, even when an agent has become available, if the available agent is has a low score for the interaction and there is a chance a better-matched agent might become available within the configured time window. The interaction flow is as follows:

1. URS calls the IsAgentScoreGood subroutine, which determines whether any of the available agents meet the threshold for handling the interaction.

2. If available agents have low scores for this interaction and the interaction spent only a short time in the queue, URS waits for a better agent to become ready.
3. The minimum acceptable score required for an agent for the interaction is gradually reduced, so if no higher-scored agent becomes available, the lower-scored agent might finally be given the interaction.

After that determination occurs, the remainder of the flow is the same as that given in the agent-surplus flow above. Use the relevant Predictive_Route_DataCfg Transaction List Object configuration options to set up the priority increments.

Dynamic Interaction Priority Increments

To avoid having interactions lingering in a queue for an excessive amount of time, URS can trigger an escalation in interaction priority after a time delay that you set. To speed up interaction handling, you can incrementally relax the minimum skill level required for agents to handle the interaction or expand the pool of agents to consider.

Each time a routing strategy tries to route an interactions, it calls the ActivatePredictiveRouting subroutine. After each failed routing attempt, the strategy checks how long the interaction has been waiting in the queue and, if the time in queue is above a certain threshold, it routes the interaction to the next available agent, no matter their score for the interaction.

Use the relevant Predictive_Route_DataCfg Transaction List Object configuration options to set up the priority increments.

Deploy the URS Strategy Subroutines

Genesys Predictive Routing (GPR) provides subroutines components that are integrated with your Genesys Routing solution. The subroutines are placed within an existing strategy, where they add agent scoring and best-match functionality that enables you to fine-tune the routing of a specific interaction to the agent who can best handle it, based on the KPIs you want to optimize.

This topic explains how to use the pre-configured strategy subroutines with Interaction Routing Designer (IRD) and Universal Routing Server (URS). This topic includes the following information:

- List of subroutines and other files included in the IP, with brief explanations of what they do.
- Deployment procedures.
- See Routing Scenarios Using GPR for a discussion of how the subroutines handle agent-interaction matching in various scenarios.

For information on how to deploy the Composer Subroutines, see Composer, Orchestration Server (ORS), and URS.

Out-of-the-Box IRD/URS Strategy Subroutines

The IP for the URS Strategy Subroutines component contains the following strategy subroutine RBN files as well as the **object.kvlt** file and the **Predictive_Route_Data_Template.cfg** file. (The RBN files are listed in alphabetical, not logical, order.)

- **ActivatePredictiveRouting_v3**: Called from a routing strategy when the conditions you specify are met. This subroutine automatically calls additional subroutines that score agents, rank the scored agents, and evaluate whether there is an agent available to handle the interaction based on agent score. Referred to simply as *ActivatePredictiveRouting* in the remainder of this topic.
- **GetActionFilters**: Called from the ActivatePredictiveRouting subroutine to parse the Virtual Agent Group string representation. In release 9.0.015.00, the GetActionFilters subroutine was enhanced to identify the list of agents matching the target skill group along with the configured login status expression. This information is also reported in the action filters of the scoring request. This functionality is invoked only when the **use-action-filters** configuration option is set to `false`.
- **GetScoringAuthToken**: Called from the ActivatePredictiveRouting subroutine to authenticate with the AI Core Services REST API.
- **GPRIxnCleanup**: Starting in URS Strategy Subroutines 9.0.015.00, performs score log and UserEvent (KVP) distribution (previously done in the GPRIxnCompleted subroutine). This attached UserEvent data includes details required for Predictive Routing performance analysis, such as gpmStatus, which indicates whether, when the interaction was routed, there was an agent-surplus or an interaction-surplus situation. Also cancels the execution of the ScoreIdealAgent and isAgentScoreGood callback subroutines if an attempt to route an interaction with Predictive Routing times out. The strategy then tries to route the interaction using skill-based routing.
  - Starting in URS Strategy Subroutines 9.0.015.00, this subroutine requires you to configure certain parameters. See Configure GPRIxnCleanup for instructions.

  - This subroutine must be called from the default port of the last Routing block used for Predictive Routing. You must also set the **Clear targets** flag in that Routing block. Alternatively, you can configure the strategy to loop back to the same Routing block again after calling the GPRIxnCleanup subroutine.

- **GPRIxnCompleted**: Computes the values for the GPR KVPs based on the selected agent scores and updates the URS global map. In releases prior to URS Strategy Subroutines 9.0.015.00, also attaches Predictive Routing–specific keys to interaction user data (done in the GPRIxnCleanup subroutine in URS Strategy Subroutines 9.0.015.00 and later). This attached data includes details required for Predictive Routing performance analysis, such as gpmStatus, which indicates whether, when the interaction was routed, there was an agent-surplus or an interaction-surplus situation. Links to the interaction ID in Genesys Info Mart. (Named PRRIxnCompleted in earlier releases.)
- **GPRIxnSetup**: (Introduced in URS Strategy Subroutines 9.0.015.00) Creates the interaction global map in URS memory with the ConnectionID as the key name; adds the Genesys Info Mart KVPs and initializes them with the default values; sets the gpmMode parameter to `off` and gpmResult to 15 (Predictive Routing is turned off or not used for this interaction). See Configure GPRIxnSetup for complete configuration instructions and how to use this subroutine.
- **isAgentScoreGood**: URS calls this subroutine to suppress routing to an agent who is in ready state if this agent does not provide an acceptable match for the interaction. You can use this in conjunction with relaxation thresholds to target better-matched agents preferentially, expanding the pool of agents if the best-matched agents are unavailable.
- **ScoreIdealAgent**: Called by URS before routing an interaction to an Agent Group or Virtual Agent Group at the time URS invokes the SelectDN function. This subroutine sorts the agents within the target group according to their scores, in decreasing order. It can also rescale the agent score to the range accepted by URS, if necessary.
- **ScoreIdealAgentDefault**: Used either when the agent scores for the current interaction are unknown or when running GPR in dry-run mode.
- **SetIdealAndReadyCondition**: Called from ActivatePredictiveRouting subroutine. This is a wrapper subroutine around scheduling the calls to subroutines such as ScoreIdealAgent and isAgentScoreGood, which are executed outside the main interaction processing flow. Implements the GPR operation modes controlled by the **prr-mode** option.
- **SetIdealAndReadyConditionforOCS**: Required if you are using the Composer Workflow Subroutines.

Other Files

- **objects.kvlt**: A text file containing objects required by the strategy subroutines, including the Print_Log_Message macro. You must import this file as well as the RBN file for the strategy subroutines to work correctly.
- **Predictive_Route_Data_Template.cfg**: Template for the Predictive_Route_DataCfg Transaction List object. Stores GPR-related configuration options and values.

Subroutines for Environments Using Dynamic Priority Routing

Release 9.0.014.04 and higher of the URS Strategy Subroutines provide the following subroutines that provide improved dynamic-priority interaction handling:

- **ActivatePredictiveRouting_v3** - This subroutine does the following:
  1. The main routing strategy should set the initial priority for the interaction. The Initialize Variables block takes the initial priority parameter from the skill data `List` object and saves it to the

parBasePriority local variable.

2. After a scoring request is completed and the agent scores are placed into the URS Global Map linked to the interaction, the subroutine executes priority increments. This happens once per interaction.
3. The subroutine verifies whether the **set-dynamic-priority** option is set to `true`. The following steps are executed only on the first routing attempt when the option is set to `true`.
4. The subroutine reads the remaining priority options configured on the Predictive_Route_DataCfg `List` object (**priority-increment**, **priority-init-interval**, and **priority-interval**).
5. The subroutine calls the IncrementPriorityEx function with the required arguments.

Subroutines for Environments Using Non-ASCII Encoding

Release 9.0.014.04 and higher of the URS Strategy Subroutines can support non-ASCII encoding (by default, all GPR components use UTF-8 encoding). The subroutines specified below include enhancements for non-ASCII environments:

- **ActivatePredictiveRouting_v3** - Converts non-ASCII characters to UTF-8 characters before sending scoring requests to the Predictive Routing scoring engine.
- **GPRIxnCompleted** - Converts non-ASCII characters to UTF-8 characters before sending scoring logs.
- **GetScoringAuthToken** - Replaces the StrFormat function with the SetStringKey function. This eliminates an issue with the *~s* character in the StrFormat function, which does not work correctly in a non-ASCII environment.

> **Important**
>
> URS 8.1.400.55 is the minimum required version for GPR-specific IRD subroutines to work in non-ASCII environments.

Installing the URS Strategy Subroutines

The following is a high-level overview of the steps required to deploy the URS Strategy Subroutines:

1. Configure URS to Support Predictive Routing.
2. Import the subroutines. For a list of the subroutines, with descriptions of their functionality, see IRD/URS Strategy Subroutines.
3. Define the entry points in your IRD strategy for the appropriate subroutines.
4. Set appropriate values for the strategy subroutine configuration options, which are located in the Predictive_Route_CfgData Transaction List object.
5. Configure the parameters for the subroutines used in your environment.
6. Test that the subroutines are correctly directing interactions to agents.

The following sections provide detailed instructions for setting up your subroutines.

Configuring URS to Support Genesys Predictive Routing

Perform the following steps to configure URS to work with GPR:

1. Create the **static_strategy** configuration option in the **[default]** section of the URS `Application` object and set its value to `empty`. You can set this option either on the level of individual routing points or on the tenant/URS level. Depending on where you set it, the option works slightly differently. This option takes effect immediately and does not require that you restart URS.
2. Configure the http log for URS to check scoring requests and responses. To do this, set the **verbose** option in the **[web]** section to 3.
3. Set the **run_verbose** option in the **[default]** section to `0`.
4. Set the **vqtime** option in the **[default]** section to `13:2048`.
5. Enable HTTPS support:
   No changes to the Subroutines components are required. However, HTTPS support on Unix-based operating systems requires that you install the Genesys Security Pack on the host running Universal Routing Server (URS). See Installing Genesys Security Pack in the *Genesys Security Deployment Guide* for more information.

   - For instructions, and a general discussion of HTTPS and TLS support among Genesys components, see the *Genesys Security Deployment Guide*.
   - For HTTP/S configuration for URS, see the section "Web Service Connections Using HTTP Bridge" in the *Genesys 8.1 Universal Routing Deployment Guide*, the sections on Web Service Option (p. 687), IRD Web Service Object (p.771), and TLS (p. 782) in the Universal Routing Reference Manual, and HTTP Bridge Updates in the Supplement to the *Universal Routing Reference Manual*.
     - Only simple TLS is supported. For simple TLS, you need to configure only the URS **def_trusted_ca** option.

     If you are using a chain of trusted certificates (intermediate ca and root ca, in pem file format), you can concatenate them, as described in Configuring Multiple Trusted CAs in the *Genesys Security Deployment Guide*.

Strategy Subroutine Integration for URS/IRD

To use the strategy subroutines provided with Predictive Routing, perform the following steps:

1. Navigate to the Export/Import Bar in IRD and import the strategy subroutines from the URS Strategy Subroutines IP.
   For instructions, see the *Export/Import Bar* topic in the *Interaction Routing Designer Help*, which you can open from the Universal Routing landing page.
2. Open the strategy you plan to use with Predictive Routing and place the ActivatePredictiveRouting Call Subroutine object in the desired location. It must be inserted into the routing strategy before the call to a Routing block or a call to the SelectDN function. The graphic below shows the subroutine insertion points.

This subroutine requires the following three input parameters:

- `skill_target`: A STRING indicating the target selected by the URS for an interaction. This must be a virtual agent group.
- `skill_data`: A LIST, which must contain the following two keys:
  - `overflow_timeout`: A STRING defining a period of time in seconds during which URS tries to route the interaction to the current target.
  - `base_priority`: An INTEGER defining the priority value the interaction has before the call to the ActivatePredictiveRouting subroutine.
- `default_skill_data`: A LIST, which must contain the following keys:
  - `AgentScore`: Takes either `Y` or `N` as its value. To activate predictive routing, you must set this parameter to `Y`.
  - `predictor`: Contains the name of the section in the Predictive_Route_DataCfg `Transactions List` configuration object that defines predictor configuration.
  - START_TS: (Used in URS Strategy Subroutines 9.0.015.00 and later. See gpmWaitTime Configuration for details.) The UTC time indicating when an interaction arrived in the queue.
  - `prr-ixn-timestamp`: (Used in URS Strategy Subroutines 9.0.014.04 and earlier. See gpmWaitTime Configuration for details.) Contains a value defining the number of seconds since midnight when the interaction was queued.

3. The isAgentScoreGood subroutine checks for interactions that have been in queue for an unusually long time using the `varQueueTimeSec` parameter, which is set to `600` seconds by default.

> **Important**
>
> - If you are expecting conditions that might result in longer wait times, you might need to set a larger value for this variable. To change the value of this variable, contact Genesys Customer Care for assistance.
> - URS might experience high CPU loads if the timeout is extended beyond 600 seconds and you are using agent hold-out (that is, you have set the value for the **use-setreadycondition** option to `true`).

1. The GPRIxnCompleted subroutine (called PrrIxnCompleted in earlier releases) can be inserted either as a Custom Routing step in the Routing object or immediately in front of the object in your strategy that contains a call to the RouteCall function.
2. In URS Strategy Subroutines releases prior to 9.0.007.00, insert the PrrIxnCleanup subroutine after the green port exit from either the Routing object or the object that contains a call to the RouteCall function.
   NOTE: PrrIxnCleanup is not used in URS Strategy Subroutines release 9.0.007.00 and higher. This update is supported only in environments running URS version 8.1.400.37 and higher.
3. Set values for the configuration options in the Predictive_Route_DataCfg Transaction List Object. Among others, the GPR subroutines send the scoring requests to the URL configured in the **jop-scoring-url** option and score_log requests to the URL configured in the **jop-logging-url** option. The **orig-connid-key** options is required in all deployments to store the original

connection ID, used as a unique identifier, for each interaction.

Configure GPRIxnSetup

The GPRIxnSetup subroutine, introduced in URS Strategy Subroutines 9.0.015.00, does the following:

- Creates the interaction global map in URS memory with the ConnectionID (ConnID) as the key name.
- Adds all the Genesys Info Mart KVPs used in GPR and initializes them with the default values. All the KVP values initialized by the GPRIxnSetup are replaced with actual values when the remaining GPR subroutines are invoked. If the GPR subroutines are not invoked, the score log and the Genesys Info Mart database continue to store the default values set in the GPRIxnSetup subroutine.
- Sets the gpmMode parameter to `off` and gpmResult to 15 (Predictive Routing is turned off or not used for the current interaction). This establishes the initial condition in the contact center and establishes clearly when GPR actively begins to score agents and determine routing targets. As soon as interactions are routed using the GPR subroutines, these KVP values are updated to reflect actual contact center conditions.

To use GPR, set the value of the URS **prestrategy** option to `GPRIxnSetup`.

- If you already specify a subroutine in the **prestrategy** option, you can copy the contents of the GPRIxnSetup subroutine into your existing subroutine.
- For a complete description of the **prestrategy** option, see the Universal Routing Reference Manual.

Configure GPRIxnCleanup

The GPRIxnCleanup subroutine must be called from the default port of the last Routing block used for Predictive Routing. Starting in URS Strategy Subroutines 9.0.015.00, this subroutine can now correctly identify abandoned interactions and interactions in which GPR was unable to route the interaction and add this information to the score log and the Genesys Info Mart gpmResult KVP. The KVP values used are the following:

- gpmResult = 13 - Interaction abandoned
- gpmResult = 14 - GPR routing was unsuccessful

To enable this functionality, configure the GPRIxnCleanup subroutine as follows:

1. When routing succeeds, call GPRIxnCleanup with the parameter `true` from the **default** port of the last Routing block used for Predictive Routing.
2. When routing fails, call GPRIxnCleanup with the parameter `false` from the **red** port of the last Routing block used for Predictive Routing.
3. When an interaction is abandoned, GPRIxnCleanup is called automatically with an empty parameter.

Configure Reporting on Both GPR and Non-GPR Interactions

There are a number of scenarios in which interactions routed using GPR and those routed using alternative methods (non-GPR) might target the same pool of agents:

- Non-GPR interactions might be sent to the same queue as GPR interactions.

- A strategy might send some interactions to GPR and to non-GPR routing, based on specified conditions.
- A different strategy might be routing interactions to the same target agents as the GPR interactions.

URS Strategy Subroutines 9.0.015.00 and later provides support for tracking interactions not routed using GPR. The following KVP values provide the necessary information:

- gpmMode = `off`
- gpmResult = 13 (abandoned) or 14 (routing attempt failed, non-GPR routing used)

> **Important**
>
> If routing fails for a non-GPR interaction, or it is abandoned, GPR does not record these status conditions. For GPR the following is recorded: gpmResult = 15, gpmMode = `off`, gpmMessage = `Predictive Routing is turned off or not used for this interaction`.

To configure your routing environment to handle both GPR and non-GPR routed interactions, perform the following steps:

1. Configure the gprIxnSetup subroutine, which is require to report correctly on blended GPR and non-GPR routing.
2. Call the following subroutines in all strategies routing interactions to agents also targeted from GPR. You can invoke them directly, without going through the ActivatePredictiveRouting_v3 subroutine.
   - GPRIxnCompleted should be called as a custom routing step in all Target Selection blocks or as a Function block before routing interactions to agents using a RouteCall block.
   - GPRIxnCleanup should be called after all Target Selection blocks or RouteCall blocks.

gpmWaitTime Configuration

In release 9.0.015.00 and later, the URS Strategy Subroutines use the value of the START_TS variable to calculate when the interaction arrived in the queue. START_TS is a mandatory variable passed in the default_skill_data List object and stored as a KVP for use in Genesys Info Mart reporting.

- gpmWaitTime is calculated based on the difference between: (the UTC time when the agent is selected) - (START_TS variable value).
- The START_TS variable is also used for measuring periods of time in A/B tests.
- In Genesys Info Mart reporting, the value of the START_TS variable is converted to the DateTimeKey function, used to populate the START_DATE_TIME_KEY column in the GPM_FACT table.

In release 9.0.014.04 and earlier, the subroutines use the value of the prr-ixn-timestamp variable, in the default_skill_data List object, which is passed to the ActivatePredictiveRouting_v3 subroutine. prr-ixn-timestamp defines the number of seconds since midnight when the interaction was queued.

This variable is then used for the calculation of the gpmWaitTime which denotes the actual wait time of the interaction in the queue before an agent is selected. This calculation uses the TimeStamp[] and TimeDifference[] functions, available in IRD/URS.

Turn Off Predictive Routing

You might choose to turn predictive routing off temporarily for A/B testing or troubleshooting. To do so, use one of the following methods:

- Set the **prr-mode** configuration option to `off`.
- Set AgentScore = `N` in the default_skill_data object parameter passed to the ActivatePredictiveRouting_v3 subroutine.

Deploying the Composer Strategy Subroutines

Genesys Predictive Routing (GPR) provide subroutines components that are integrated with your Genesys Routing solution. The subroutines are placed within an existing Composer workflow, where they add agent scoring and best-match functionality that enables you to fine-tune the routing of a specific interaction to the agent who can best handle it, based on the KPIs you want to optimize.

Genesys Predictive Routing provides pre-configured subroutines for use with either:

- Interaction Routing Designer (IRD) and Universal Routing Server (URS)
- Composer, Orchestration Server (ORS), and URS

These strategy subroutines are *static strategies*, that is, they run automatically once they have been set up. They respond to configured conditions without needing ongoing adjustments.

> **Important**
>
> - If you would like to evaluate Genesys Predictive Routing for use with schedule-based routing (using Genesys Workforce Management), service-level routing, or business-objective routing, contact Genesys Professional Services for a review of your routing environment.
> - For descriptions of how the subroutines handle agent-surplus and interaction-surplus scenarios, see Interaction Flows.

Out-of-the-Box Composer/ORS/URS Workflow Subroutines

The Composer strategy subroutines IP provides a complete Composer project, including all the workflows necessary to perform predictive routing, and including an example workflow. The Composer strategy subroutines make use of the underlying functionality provided by the URS subroutines described above.

When you extract the **ScoreBasedRoutingComposer_v4.zip** file, which contains the Composer subroutines, you get a top-level folder called **ScoreBasedRoutingComposer_v4**, with various subfolders, including the following:

The **Workflows** subfolder contains the following workflow files:

- Activate GPR.workflow

- GetActionFilters.workflow
- GetCustomerRecord.workflow
- GetScoringAuthToken.workflow
- GPRIxnCompleted.workflow
- Main.workflow
- SetScoreMaps.workflow

The **src** subfolder contains a subsubfolder called **subroutines**, which contains the following SCXML files:

- ClearTargets.scxml
- InteractionAccept.scxml
- QueueCancel.scxml
- QueueQuery.scxml
- SuspendForEvent.scxml

The **src-gen** subfolder contains the following SCXML files:

- Activate GPR.scxml
- GetActionFilters.scxml
- GetCustomerRecord.scxml
- GetScoringAuthToken.scxml
- GPRIxnCompleted.scxml
- IPD_GPRDemo_GPRMain.scxml
- Main.scxml
- SetScoreMaps.scxml

The ActivateGPR subroutine (called ActivatePredictiveMatching in earlier releases), after you insert it into your Composer workflow, triggers the ActivatePredictiveRouting_v<*version_number*> URS subroutine.

The GPRIxnCompleted (called PMIxnCleanup in earlier releases) and (in releases prior to 9.0.007.00) PMIxnCleanup subroutines can be inserted into the Composer workflow in the appropriate places. They perform the same functions of attaching matching user data and cleaning up the URS global map as the comparable GPRIxnCompleted and PrrIxnCleanup subroutines.

Other Files

The following files included in the URS Strategy Subroutines component IP are required for the Composer Strategy Subroutines component to function correctly.

- **SetIdealAndReadyConditionforOCS**: An RBN file included in the URS Strategy Subroutines component IP, which is required for the Composer Strategy Subroutines component to function correctly.

- **objects.kvlt**: A text file containing objects required by the strategy subroutines. You must import thais file as well as the RBN file for the strategy subroutines to work correctly.

Installing the Composer/ORS/URS Strategy Subroutines

The following is a high-level overview of the steps required to deploy the Composer/ORS/URS Strategy Subroutines:

1. Configuring URS to Support Predictive Routing.
2. Import the subroutines. For a list of the subroutines, with descriptions of their functionality, see Composer/ORS/URS Strategy Subroutines.
3. Define the entry points in your Composer workflow for the appropriate subroutines. See the complete integration instructions for Composer for specific recommendations and configuration information.
4. Set appropriate values for the strategy subroutine configuration options, located in the Predictive_Route_CfgData Transaction List object.
5. Configure the parameters for the subroutines used in your environment.
6. Test that the subroutines are correcting directing interactions to agents.

The following section provides detailed instructions for setting up your subroutines:

- Composer/Orchestration Server (ORS)/Universal Routing Server (URS)

Configuring URS to Support Predictive Routing

There are two main steps to configure URS to work with Predictive Routing:

1. Create the **static_strategy** configuration option in the **[default]** section of the URS `Application` object and set its value to `empty`. You can set this option either on the level of individual routing points or on the tenant/URS level. Depending on where you set it, the option works slightly differently. This option takes effect immediately and does not require that you restart URS.
2. Configure the http log for URS to check scoring requests and responses. To do this, set the **verbose** option in the **[web]** section to 3.
3. Set the **run_verbose** option in the **[default]** section to `0`.
4. Set the **vqtime** option in the **[default]** section to `13:2048`.

Strategy Subroutine Integration for ORS/Composer

Predictive Routing comes with a pre-configured Composer strategy subroutine. The Composer subroutine ActivatePredictiveMatching is a wrapper around the URS subroutine

ActivatePredictiveRouting and calls the latter via URS web API call. You can use it as-is, edit it, or create your own based on the model of the pre-configured subroutine.

Design Considerations

The subroutine that comes with Predictive Routing has been created to work efficiently. It:

1. Minimizes the use of IRD subroutines invoked by the Composer workflow.
2. Reuses existing IRD subroutines as much as possible to avoid maintenance and upgrade of multiple versions.
3. Uses the same configuration options as the strategy subroutines for URS, specifically those configured in the Predictive_Route_DataCfg `Transaction List` object.

Functional Blocks and their Behaviors

The ActivatePredictiveMatching subroutine contains a number of functional blocks:

- The Entry block handles the setAgtScores timeout, which is set once before the agent score memory maps are set.
- The Score request content **format_as_map** can be set to set to `false`, which returns the agent scores as a JSON array with entries ordered by decreasing score, or to `true`, which returns scores as a dictionary, where the keys are employee IDs for the targeted agents and values are their scores for the interaction.
- The WebRequestScore block is configured to use a single retry in the event of an error. After one retry, if the retry fails, an error is logged and recorded in the attached data and it returns a value of `false`.
- The AgentScores ECMA script block parses the returned JSON array and prepares a JSON object with entries ordered by score. In case of JSON object parsing error the script assigns variables varResult.success = "false" and varResult.message = "Parsing of agent scores failed".
- The AgentScoreMaps subroutine invocation block includes the **parCustomerId** parameter, which maintains consistency with the IRD version.
- The PMSetIdealAndReady SCXML State block invokes the IRD SetIdealAndReadyConditionForORS strategy through an HTTP request to URS. Details of the invocation can be found below.

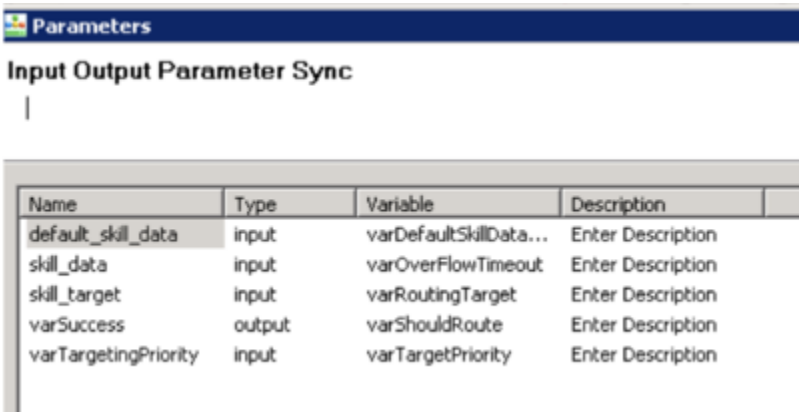
After a target is returned from the Target block in the ActivatePredictiveMatching subroutine, the PMIxnCompleted and PMIxnCleanup subroutines are invoked. They work essentially as the PrrIxnCompleted and PrrIxnCleanup IRD subroutines do, except that the PMIxnCleanup subroutine takes advantage of new memory map cleanup functionality implemented in URS 8.1.400.37, and therefore does not explicitly remove agent score data from the memory map.

Importing and Configuring the Composer Workflows

To integrate the GPR Composer subroutines into your routing environment, you must import the workflows.

1. In Composer, select **Window > Preferences > Composer > SCXML Templates**.
2. In the resulting **Templates** dialog box, click the **Import...** button.
3. Navigate to the folder containing the SCXML file, select it, and click **Open**.

The main workflow should invoke the ActivateGPR subroutine with the parameters shown in the following image:



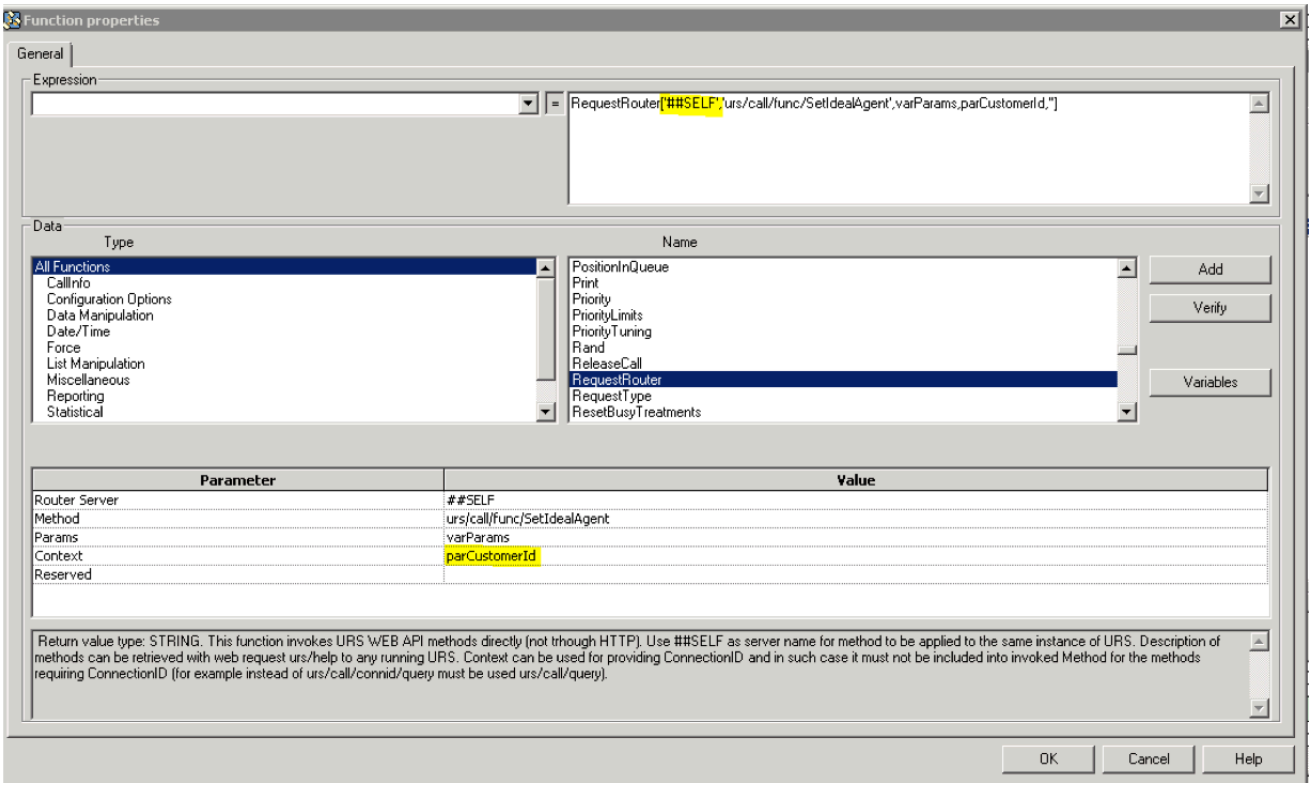
Important

The input variables **default_skill_data**, **skill_data**, and **skill_target** are equivalent to the IRD implementation (above). The **varTargetPriority** value is the targeting priority to be used. The output is a Boolean value indicating success if `true`, `false` if targeting fails.

The implementation and invocation of the ORS SetIdealAndReadyConditionForORS subroutine are a key part of the Composer integration. The implementation within this IRD subroutine provides the URS callbacks for the SetIdealAgent and SetReadyCondition functions. This enables the use of virtual queues for reporting and agent reservation within the main workflow implementation.

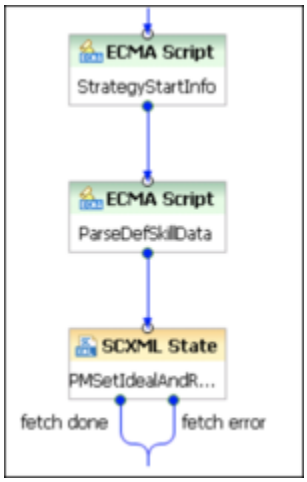
The workflow performs scoring authentication in the GetScoringAuthToken subroutine workflow, which obtains the url, api key, and so on from the Predictive_Route_DataCfg Transaction List object, just as in the IRD implementation. The agent scores are obtained from AICS, and URS memory maps are populated with agent ID, connid, and agent score. Again, this is equivalent to the IRD implementation. Next the workflow invokes the SetIdealAndReadyConditionForORS subroutine. After completion, normal workflow targeting proceeds.

The SetIdealAndReadyConditionForORS IRD subroutine uses the IRD function, RequestRouter, shown below, to invoke the SetIdealAgent and SetReadyCondition functions through an internal URS HTTP interface, directed at ##SELF, the URS on which the call is being handled. The Context is the `connid` to be used.



The example shows the invocation of the SetIdealAgent URS function. The equivalent is performed for the SetReadyCondition function as well.

Invoking the SetIdealAndReadyConditionForORS from a Composer workflow details are shown below.



The two ECMA Script blocks prepare the required information for the invocation, which occurs in the SCXML State block.

Setting the Configuration Options

Set values for the following configuration options in the Predictive_Route_DataCfg Transaction List

Object.

All About How the Composer Workflow Blocks Work

Details of each block follow.

StrategyStartInfo

This block sets the following variables as below varConnID, varURSRequestURL, and ursRequestTimeout are User variables.

```
try {

        // get the connid of this interaction

        varConnID = _genesys.ixn.interactions[system.InteractionID].voice.connid;

        // set urs call start request

        var ursHost = _genesys.session.getConfigOption('null', "hostname",
         _genesys.session.lookupseq.StartFromRouter);
        var ursHttpPort = _genesys.session.getConfigOption('null', "http_port",
        _genesys.session.lookupseq.StartFromRouter);
                varURSRequestURL =  'http://' + ursHost + ':' + ursHttpPort +'/urs/call/'
+varConnID + '/exec';

        // fetch timeout in seconds for strategy - TODO - add to xlist

        ursRequestTimeout = 15;

} catch (error) {
        __Log('###DWS StrategyStartInfo error l = ' + uneval(error)  );
}
```

ParseDefSkillData

The User variable varDefaultSkillDataString is populated from the default_skill_data input variable to the ActivatePredictiveMatching.workflow subroutine, as show below.

```
try {

        var vKeys = Object.keys( default_skill_data );

        // want “prr-ixn-timestamp:64407|predictor:qaart-predictor|AgentScore:Y"

        var vKeyCount = vKeys.length;
        for ( var iKey = 0; iKey < vKeyCount; iKey++ )
        {
                varDefaultSkillDataString  = varDefaultSkillDataString + vKeys[iKey]  +':'
+   default_skill_data[vKeys[iKey]];
                if ( iKey < vKeyCount - 1)  varDefaultSkillDataString =
varDefaultSkillDataString + '|';
        }

} catch (error) {
        __Log('###PRRDemo ActivatePredictiveRouting PatseDefSkillData  error  = ' +
uneval(error)  );
```

```
}
```

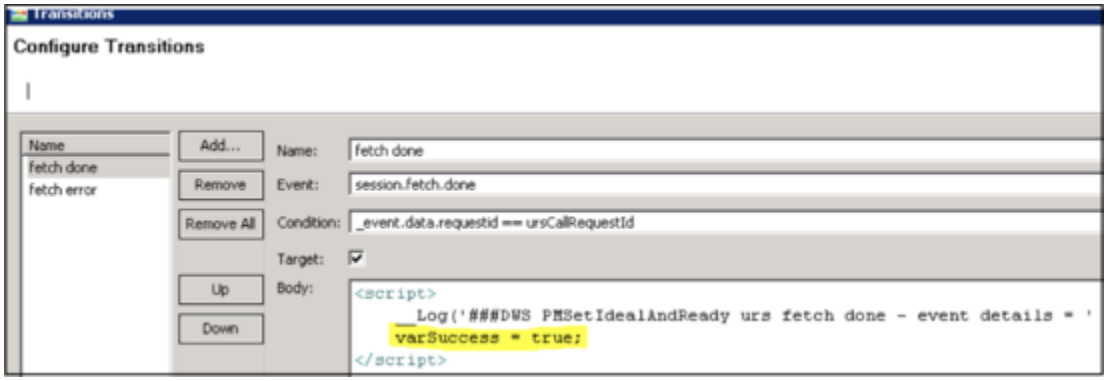
PMSetIdealAndReady

The onentry element performs the invocation with the following parameters:

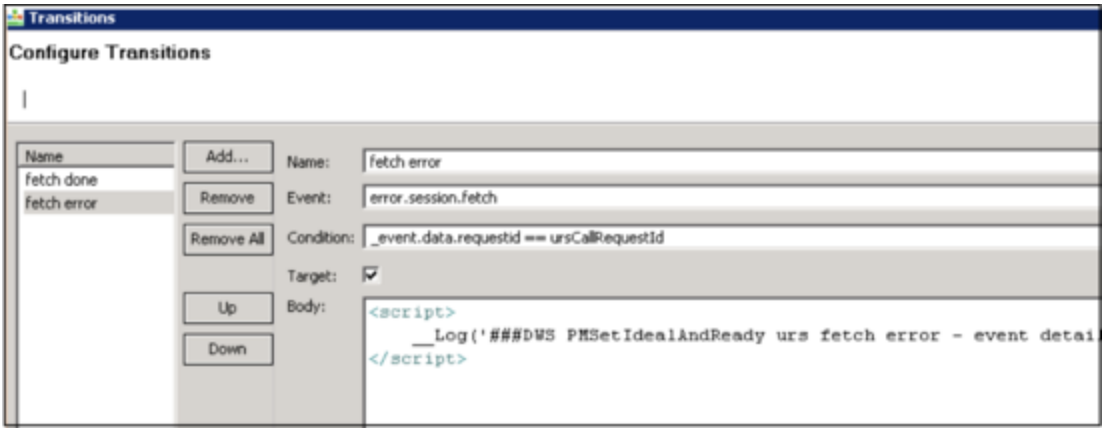
- skill_target and skill_data parameters: Equivalent to the IRD implementation. Input variables to the ActivatePredictiveMatching.workflow subroutine.
- varPredictorCfg is also the equivalent to the IRD implementation, retrieved from the Predictive_Route_DataCfg `Transaction List` object for the defined predictor.

```
<onentry>
        <session:fetch requestid="ursCallRequestId" srcexpr="varURSRequestURL" method="'get'"
timeout="ursRequestTimeout" >
                <param name="tenant" expr="_genesys.session.tenant"/>
                <param name="strategy" expr="'SetIdealAndReadyConditionForORS'"/>
                  <param name="TARGET" expr="skill_target"/>
                <param name="TARGET_TIME_TO_WAIT" expr="skill_data"/>
                <param name="CustomerId" expr="parCustomerId"/>
                <param name="DefaultSkillData" expr="varDefaultSkillDataString"/>
                <param name="PredictorCfg" expr="varPredictorCfg"/>
        </session:fetch>
</onentry>
```

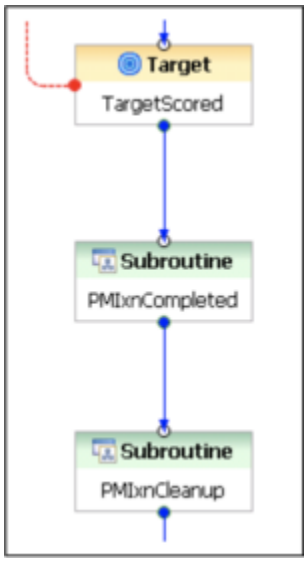
The transitions from the state are as shown below. The session.fetch.done event body sets the varSuccess User variable to `true`.



The error.session.fetch event body leaves the varSuccess User variable at the default value of `false`.



Once this block has completed, the subroutine exits to the main workflow where normal targeting occurs if the invocation was successful. If not, the sample main workflow still performs normal targeting, but skips the PMIxnCompleted and PMIxnCleanup as shown below.

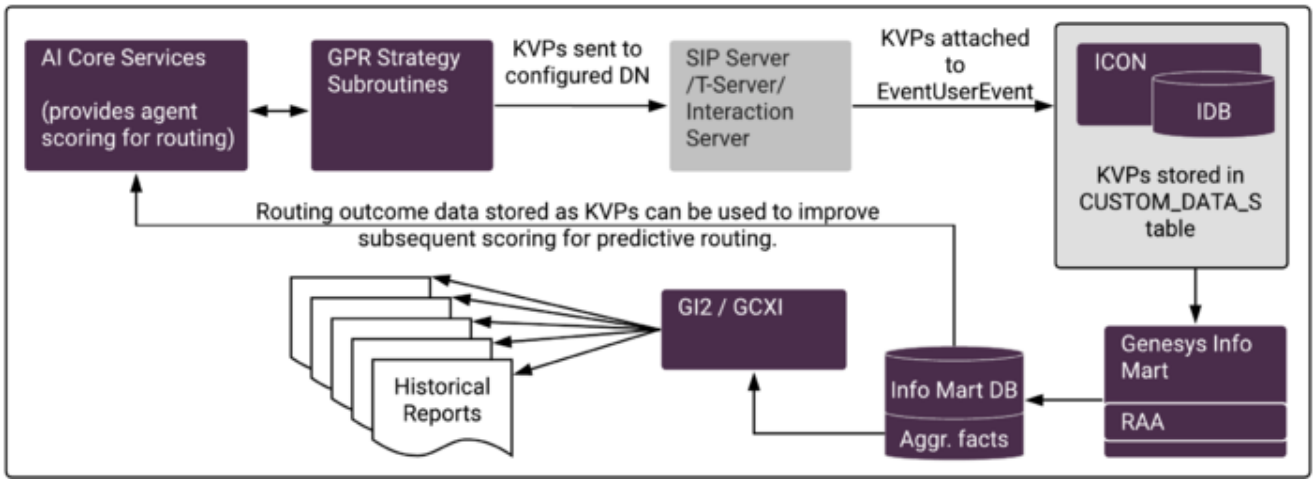


The PMIxnCompleted input is simply the resource selected returned from the Target block.

The PMIxnCleanup input parameters are the varInteractionId, which is the connid of the call being processed, and varResourceSelected, which is the same as for PMIxnCompleted.

Integrate with Genesys Reporting

Genesys Predictive Routing (GPR) can supply a variety of information about routing outcomes for use by the Genesys reporting applications. GPR sends data for historical reporting in key-value pairs (KVPs). This KVP data, which is stored in the Info Mart database, can also to be fed back into GPR to refine predictors. In addition, Stat Server sends this KVP data to Pulse for real-time reporting.



1. GPR data in the form of KVPs is attached to EventUserEvent TEvents.
   The UserEvent contains **AttributeThisDN** with a value of `Route Point`, which identifies where the strategy is executed and **AttributeUserData,** which holds a list of KVPs containing data about the interaction.

   This Route Point should also be specified in the **vq-for-reporting** option.
2. Interaction Concentrator stores the KVP data in the Interaction Database (IDB), in the CUSTOM_DATA_S table.
3. Genesys Info Mart gathers this raw data from IDB and prepares it for use in Genesys CX Insights (GCXI) historical reporting on GPR activity and performance.
   - Genesys Info Mart has specific requirements for the KVPs that must be attached (see GPR KVPs for Genesys Reporting, below).
4. Reporting and Analytics Aggregates (RAA) further transforms the data in preparation for use by the presentation layer.
5. Using GCXI, you can create the following reports based on the GPR data. These reports are described in the *Genesys Customer Experience Insights User's Guide*:
   - Predictive Routing - AHT & Queue Dashboard
   - Predictive Routing - Model Efficiency Dashboard
   - Predictive Routing Agent Occupancy Dashboard
   - Predictive Routing A/B Testing Report
   - Predictive Routing Detail Report

- Predictive Routing Operational Report
- Predictive Routing Queue Statistics Report

6. Using **Pulse**, you can access the *Agent Group KPIs by Predictive Model* and *Queue KPIs by Predictive Model* templates for real-time reporting. These templates are available from the Genesys Dashboard Community Center.

In addition to the powerful Genesys Reporting solution, GPR offers in-interface analysis reports:

- Feature Analysis
- Agent Variance
- Lift Estimation

The following topics provide in-depth information about how Universal Routing Server (URS) makes routing decisions when you are using Predictive Routing and how GPR scores agents in various scenarios. Use the material covered in these topics to inform your understanding about what data the KVPs described in this topic store and how to create the most useful reports from the available data:

- Routing Scenarios Using GPR
- How Does GPR Score Agents?

Configure Historical Reporting

Genesys Info Mart release 8.5.009 and later provides support for GPR reporting out-of-box, with no additional configuration required on the Genesys Info Mart side. However, to send GPR data to Genesys Info Mart, as well as to see GPR data in GCXI reports, you need to modify the configuration of GPR, Interaction Concentrator, and Reporting & Analytics Aggregates (RAA, the aggregation engine hosted by Genesys Info Mart).

> **Tip**
>
> For general information about how Genesys Info Mart uses attached user data, see the Genesys Info Mart and Attached User Data section of the *Genesys Info Mart Deployment Guide.*

1. Ensure that ICON and IDB have been deployed as Genesys Info Mart requires, and that ICON is connected to the T-Server(s) or SIP Server(s) handling the GPR interactions. For details, see Preparing Interaction Concentrator in the *Genesys Info Mart Deployment Guide.*
2. Configure a DN for GPR reporting data.
   - Open Genesys Administrator Extension (GAX) and specify a DN to use with GPR. This DN can be a VQ DN, a Trunk Group DN, or any other recognized type, as long as you configure ICON to monitor it. The name of the DN is used inside URS strategy subroutines or Composer SCXML scripts (depending on your environment), so it should be meaningful and recognizable.
3. Configure URS or Orchestration Server:

- In the connections of your Universal Routing Server application or Orchestration Server application (as appropriate in your environment), add the T-Server/SIP Server used to define the reporting Switch and DN in the GPR configuration. For example, GPR_Switch.

4. In IRD or Composer, set up your routing solution to attach the required KVPs in UserEvents. For an example to guide you, refer to the GPRIxnCompleted (formerly PRRIxnCompleted) subroutine provided with GPR.
5. Configure Interaction Concentrator to store the GPR KVPs:
   1. Set the store-event-data option to `all`, the recommended setting in GPR deployments, or `conf`. This option controls which KVP data from AttributeUserData of EventUserEvent ICON stores in the G_CUSTOM_DATA_S table.
   2. If you set **store-event-data** to `conf`, use the EventData option to specify which KVPs to store.
      To simplify configuration in deployments where GPR data is extracted for reporting, Genesys recommends setting the **[custom-states].store-event-data** configuration option to `all`, which ensures that ICON stores all the UserEvent-based KVPs that Genesys Info Mart requires. However, be aware that setting **store-event-data**=`all` has performance and security implications:
      - Performance — Processing and storing a large number of UserEvent-based KVPs increases database resource requirements and can impact performance.
      - Security — Sensitive data (for example, credit card information) might be sent in UserEvents that are not used for reporting. Unlike the situation for call-based attached data, where the G_SECURE_USERDATA_HISTORY table is available to provide secure IDB storage, there is no secure IDB table parallel to G_CUSTOM_DATA_S that provides separate, secure storage for sensitive data.
6. Ensure that you have added the T-Server/SIP Server corresponding to the DN you created earlier for GPR to the **Connections** tab of the Interaction Concentrator `Application` object.
7. Configure GPR to attach KVP data by configuring the following options on the **Predictive_Route_DataCfg** Transaction List object:
   - send-user-event - Enables attaching the Predictive Routing-specific key-value pairs.
   - vq-for-reporting - Indicates the virtual queue or DN where URS sends the GPR user event data. The user event data, in the form of key-value pairs (KVPs), is attached to EventUserEvent in the AttributeUserData attribute.

     For the list of KVPs to be attached, see GPR KVPs for Genesys Reporting, below. The following KVPs are mandatory for data to be available for Genesys Reporting:
   - gpmResult
   - CALLID
   - START_TS
   - ADDED_TS
8. Ensure that your deployment has been configured as required for Genesys Info Mart to support reporting on contact center activity in general. For a summary of the configuration requirements, see Enabling Reporting on Voice Activity in the *Genesys Info Mart Deployment Guide*.
9. Enable aggregation of GPR data. (Required for GCXI reporting or other applications that use RAA aggregation.)
   - In the **[agg-feature]** section on the Genesys Info Mart application object, specify the enable-gpr and enable-gpr-fcr options.

10. Verify the reporting data chain.
    After a few interactions have been routed with GPR in an operational mode, verify that the required KVPs are being sent, stored, and used as expected:

    - Check the T-Server/SIP Server logs to verify that UserEvents are being sent with the required KVPs.
    - Check the ICON logs and the G_CUSTOM_DATA_S table in IDB to verify that ICON is recording the required KVPs.
    - Check the GPM_* tables in the Info Mart database to verify that Genesys Info Mart is correctly transforming the data.

For more information about configuring user data storage in Interaction Concentrator to work with Genesys Info Mart, see Important custom-states ICON Configuration Options and Configuration Considerations in the *Genesys Info Mart Deployment Guide.*

GPR KVPs for Genesys Reporting

The following table describes the KVPs that Genesys Info Mart uses to enable GPR reporting.

Tip

- Use the Search box to quickly locate a specific KVP.
- Although the Predictive Routing short name is GPR, the GPM_* prefix shown in the table below is correct. It reflects an earlier name for the product.

Important

A number of new KVPs were introduced in URS Strategy Subroutines 9.0.015.00, resulting in additional columns in the Genesys Info Mart tables that store GPR data and extending the range of information provided in some existing columns. Genesys Info Mart 8.5.014.09 and higher is required to use the new KVPs.

Interaction rows created before the new GPR subroutine IP is installed are handled as follows:

- New columns in the GPM_FACT table have NULL values for all existing rows.
- The new gpm_dim1_key field in the GPM_FACT table has the value -2 for all existing rows.

The descriptions for the new KVPs indicate the "Introduced In" and "Modified In" versions, where relevant.

KVP	Description	KVP Type	Info Mart Database Target
ADDED_TS	UTC timestamp, indicating the date and time when the record was added as inherited from the T-Server TEvent. **Default value:** no default value **Valid values:** any valid UTC timestamp **Note:** This KVP is mandatory for Genesys Info Mart reporting.	INT	GPM_FACT.ADDED_TS
CALLID	Value of AttributeCallUUID for the interaction. **Default value:** a valid CALLID **Note:** This KVP is mandatory for Genesys Info Mart reporting.	CHAR(32)	GPM_FACT.MEDIA_SERVER_IXN_GUID
CustomerID **Introduced:** 9.0.016.00	The GPRIxnCleanup subroutine takes this KVP from user data attached to the interaction, and passes it to the Genesys Historical Reporting solution in the EventUserEvent event. GPR does not generate this KVP.	Postgres: varchar(255); Oracle: VARCHAR2(255 CHAR); Microsoft SQL: varchar(255)/nvarchar(255)	IRF_USER_DATA_GEN_1.CUSTOMER_ID
gpmAdjustedAgentScore **Introduced:** 9.0.015.00	The final agent score used to route the associated interaction to the selected agent. This score is calculated from the gpmAgentScore combined with any agent occupancy factor. **Default value:** 0 **Valid values:** any non-negative float value	FLOAT	GPM_FACT.ADJUSTED_SCORE
gpmAgentDBID	Optional. The DBID of the agent to whom the interaction was routed. **Default value:** no	INT	RESOURCE_.RESOURCE_CFG_DBID (referenced through GPM_FACT.RESOURCE_KEY)
KVP	**Description**	**KVP Type**	**Info Mart Database Target**

KVP	Description	KVP Type	Info Mart Database Target
	default value		
gpmAgentRank	The rank of the agents in the target group, based on agent scores sorted in descending order. **Default value:** 0 **Valid values:** 0, any positive integer	SHORT	GPM_FACT.AGENT_RANK
gpmAgentScore	The score of the agent to whom the interaction was routed. **Default value:** 0 **Valid values:** any non-negative float value	FLOAT	GPM_FACT.AGENT_SCORE
gpmCustomerFound	Indicates whether features from the customer record specified in the routing strategy were successfully retrieved from the Customer Profile schema uploaded to the AI Core Services and used to calculate agent scores. **Default value:** unknown **Valid values:** 0 (= No), 1 (= Yes), unknown	Enum	GPM_RESULT.CUSTOMER_FOUND (referenced through GPM_FACT.GPM_RESULT_KEY)
gpmDefaultAgentScore **Introduced:** 9.0.015.00	This default agent score for the associated interaction. The value is the outcome, for this interaction, of the setting specified in the **default-agent-score** configuration option. **Default value:** 0 **Valid values:** any non-negative float value	FLOAT	GPM_FACT.DEFAULT_SCORE
gpmDefaultScoredAgents **Introduced:** 9.0.015.00	The number of agents assigned the default score for the associated interaction. **Default value:** 0 **Valid values:** 0, any positive integer	INT	GPM_FACT.DEFAULT_SCORES_COUNT
KVP	**Description**	**KVP Type**	**Info Mart Database Target**

KVP	Description	KVP Type	Info Mart Database Target
gpmDefaultScoreUsed **Introduced:** 9.0.015.00	• 0 - The agent score for the associated interaction is taken from the scoring response returned by GPR. • 1 - The agent score for the associated interaction is calculated based on the value set for the **default-agent-score** configuration option. **Default value:** 0 **Valid values:** 0, 1	INT	GPM_FACT.DEFAULT_SCORE_USED
gpmFinalScoreThreshold **Introduced:** 9.0.015.00	The final threshold value used to route the associated interaction to the selected agent. The routing strategy calculates the value from the configured score threshold combined with values resulting from any agent holdout options. **Default value:** 0 **Valid values:** any integer	INT	GPM_FACT.FINAL_SCORE_THRESHOLD
gpmGlobalScore	The mean score calculated for an interaction using the Global Model. **Default value:** 0 **Valid values:** any non-negative float value	FLOAT	GPM_FACT.GLOBAL_SCORE
gpmGlobalScoreCount **Introduced:** 9.0.015.00	Describes the number of agent scores returned for an interaction using a GLOBAL model. **Default value:** 0 **Valid values:** 0, any positive integer	INT	GPM_FACT.GLOBAL_SCORES_COUNT
gpmInitialScoreThreshold **Introduced:** 9.0.015.00	The initial threshold value used for the interaction, taken from	INT	GPM_FACT.INITIAL_SCORE_THRESHOLD
KVP	**Description**	**KVP Type**	**Info Mart Database Target**

KVP	Description	KVP Type	Info Mart Database Target
	the value set in the **score-base-threshold** configuration option. **Default value:** 0 **Valid values:** any integer		
gpmMaxScore	The score of the best-matching agent in the target group. **Default value:** 0 **Valid values:** any non-negative float value	FLOAT	GPM_FACT.MAX_SCORE
gpmMedianScore	The median score for the target group of agents to which the agent who received the interaction belongs. **Default value:** 0 **Valid values:** any non-negative float value	FLOAT	GPM_FACT.MEDIAN_SCORE
gpmMessage	The message that displays when the Predictive Routing result reported in the gpmResult KVP is an error. **Default value:** no default value	CHAR(255)	GPM_FACT.MESSAGE
gpmMinScore	The score of the worst-matching agent in the target group. **Default value:** 0 **Valid values:** any non-negative float value	FLOAT	GPM_FACT.MIN_SCORE
gpmMode **Modified:** 9.0.015.00 - The value `off` was added.	The mode in which Predictive Routing is operating, as specified by the prr-mode configuration option. For information about turning predictive routing off, see Turn Off Predictive Routing. **Default value:** unknown **Valid values:** prod, off, dry-run, ab-test-time-sliced, unknown	Enum	GPM_RESULT.GPM_MODE (referenced through GPM_FACT.GPM_RESULT_KEY)
gpmModel	The name of the Model	CHAR(255)	GPM_MODEL.MODEL
KVP	**Description**	**KVP Type**	**Info Mart Database Target**

KVP	Description	KVP Type	Info Mart Database Target
	used to calculate agent scores for the interaction. **Default value:** unknown **Valid values:** the name of any Model in your environment		(referenced through GPM_FACT.GPM_MODEL_KEY)
gpmModelId	The UUID of the Model used to calculate agent scores for the interaction. **Default value:** unknown **Valid values:** the ID for any Model in your environment	CHAR(24)	GPM_MODEL.MODEL_ID (referenced through GPM_FACT.GPM_MODEL_KEY)
gpmPredictor	The name of the Predictor in the AI Core Services (AICS). If an error is encountered, the section name specified in the **Predictive_Route_DataCfg** Transaction List object is used as the Predictor name. **Default value:** unknown **Valid values:** the name of any Predictor in your environment	CHAR(255)	GPM_PREDICTOR.PREDICTOR (referenced through GPM_FACT.GPM_PREDICTOR_KEY)
gpmPredictorId	The UUID of the Predictor used for scoring. **Default value:** unknown **Valid values:** the ID for any Predictor in your environment	CHAR(24)	GPM_PREDICTOR.PREDICTOR_ID (referenced through GPM_FACT.GPM_PREDICTOR_KEY)
gpmPredictorType **Introduced:** 9.0.015.00	Reserved for future use. **Default value:** unknown **Valid values:** Sales, Service	CHAR[32]	GPM_DIM1.PREDICTOR_TYPE
gpmPriorityIncrement **Introduced:** 9.0.016.00	If the value is 0, the priority of the interaction did not increase above the configured base_priority	N/A	N/A
KVP	**Description**	**KVP Type**	**Info Mart Database Target**

KVP	Description	KVP Type	Info Mart Database Target
	value. If the value is 1, the priority of the interaction did increase above the configured base_priority and, as a result, the selected agent was not verified for the expected threshold score. **Note:** This KVP is not currently stored as a separate column in the Genesys Info Mart database. It can be accessed from the score_log file using the GPR API. **Default value:** 0 **Valid values:** 0,1		
gpmResult **Modified:** 9.0.015.00 - The values 12, 13, 14, and 15 were added.	The result of Predictive Routing processing. If there is an error, the gpmMessage KVP contains the error message. • 1 - Ok • 2 - Authentication to scoring engine failed • 3 - Scoring request failed • 4 - Agent list is empty • 5 - URS overload, interaction skipped • 6 - Predictor not found • 7 - Failed to build scoring request • 8 - SetIdealAgent or SetReadyCondition execution error • 9 - Interaction log not found in global map	Enum	GPM_RESULT.GPM_RESULT (referenced through GPM_FACT.GPM_RESULT_KEY)
KVP	**Description**	**KVP Type**	**Info Mart Database Target**

KVP	Description	KVP Type	Info Mart Database Target
	• 10 - Unknown error • 11 - Channel is not supported • 12 - Reserved for future use • 13 - Call Abandoned • 14 - Call Routing Failed • 15 - Predictive Routing is turned off or not used for this interaction **Default value:** no default value **Valid values:** 1–15 **Note:** This KVP is mandatory for Genesys Info Mart reporting.		
gpmRouteAttemptId	The sequence number of the attempt to route an interaction using Predictive Routing. The value of this KVP is incremented each time the ActivatePredictiveRouting subroutine is called by the strategy, starting from 1. **Default value:** 0 **Valid values:** integers starting from 1	INT	GPM_FACT.ROUTE_ATTEMPT_ID
gpmRoutingMethod **Introduced:** 9.0.015.00	Reserved for future use. **Default value:** unknown	CHAR[32]	GPM_DIM1.ROUTING_CRITERIA
gpmScoreAboveMedian	Indicates whether the score for the selected agent was better than the median score for the target group. **Default value:** unknown **Valid values:** 0 (no), 1 (yes), unknown	Enum	GPM_FACT.SCORE_ABOVE_MEDIAN
KVP	**Description**	**KVP Type**	**Info Mart Database Target**

KVP	Description	KVP Type	Info Mart Database Target
gpmStatus	Indicates the scenario under which the interaction was processed. For more information about the scenarios, see Routing Scenarios Using Predictive Routing. **Default value:** unknown **Valid values:** agent-surplus, call-surplus, unknown	Enum	GPM_RESULT.GPM_STATUS (referenced through GPM_FACT.GPM_RESULT_KEY)
gpmSuitableAgentsCount **Introduced:** 9.0.015.00	The number of agents who had scores greater than or equal to the initial threshold value when the scoring response was received. **Default value:** 0 **Valid values:** 0, any positive integer	INT	GPM_FACT.SUITABLE_AGENTS_COUNT
gpmTargetSize	The size of the scored target group (in other words, the length of the list of agents received from the scoring engine). **Default value:** 0 **Valid values:** 0, any positive integer	SHORT	GPM_FACT.TARGET_SIZE
gpmUse	The meaning depends on the mode in which Predictive Routing is operating (see the description of the gpmMode KVP). This field is set to one of the following values: • 1 - When the mode is `ab-test-time-sliced`, indicates that the interaction was selected for Predictive Routing. When the mode is `prod`, indicates the normal case, when Predictive Routing	Enum	GPM_RESULT.GPM_USE (referenced through GPM_FACT.GPM_RESULT_KEY)
KVP	**Description**	**KVP Type**	**Info Mart Database Target**

KVP	Description	KVP Type	Info Mart Database Target
	occurred without error. • `0` - When the mode is `ab-test-time-sliced`, indicates the interaction was processed with skill-based routing. When the mode is `dry-run`, indicates that the interaction completed without error. • `unknown` - For any mode, indicates that an error occurred in one of the Predictive Routing subroutines, and the solution defaulted to skill-based routing. **Default value:** unknown **Valid values:** 1, 0, unknown		
gpmVQDBID **Introduced:** 9.0.016.00	The DBID of the virtual queue or DN configured in the vq-for-reporting configuration option (configured on the Predictive_Route_DataCfgTransaction List object). • Requires Genesys Info Mart release 8.5.014.19 or higher. • This KVP is sent only to Genesys Info Mart. It does not appear in the score_log file. **Default value:** No default value **Valid values:** Any valid DBID	INT	RESOURCE_.RESOURCE_CFG_DBID (referenced through GPM_FACT.VQ_RESOURCE_KEY)
KVP	**Description**	**KVP Type**	**Info Mart Database Target**

KVP	Description	KVP Type	Info Mart Database Target
gpmVQGUID **Introduced:** 9.0.016.00	Value of the Virtual Queue ID (RPVQID) stored in the interaction user data. This is a special GUID value that uniquely identifies the entrance of the interaction into certain virtual queues. The RPVQID is created by URS when the interaction enters into the virtual queue and is present in all VirtualQueue events that URS distributes. • Requires Genesys Info Mart release 8.5.014.19 or higher. • This KVP is sent only to Genesys Info Mart. It does not appear in the score_log file. **Default value:** No default value **Valid values:** Any valid Virtual Queue GUID	CHAR[32]	GPM_FACT.VQ_GUID
gpmWaitTime	The amount of time, in seconds, the interaction spent in the queue used for Predictive Routing decision-making, starting from when the strategy started to process the interaction until it was routed to the agent. Note that the point when processing starts might depend on how you have configured your strategy. **Default value:** 0 **Valid values:** 0, any positive integer	INT	GPM_FACT.WAIT_TIME
ServiceType	The GPRIxnCleanup subroutine takes this	Oracle: VARCHAR2(255 CHAR); Postgres:	INTERACTION_DESCRIPTOR.SERVICE_TY
KVP	**Description**	**KVP Type**	**Info Mart Database Target**

KVP	Description	KVP Type	Info Mart Database Target
Introduced: 9.0.016.00	KVP from user data attached to the interaction, and passes it to the Genesys Historical Reporting solution in the EventUserEvent event. GPR does not generate this KVP.	varchar(255); Microsoft SQL: nvarchar(170)	
START_TS	UTC timestamp, indicating the time when the interaction arrived at the contact center. Note that this value is different from gpm-ixn-timestamp (previously called prr-ixn-timestamp), which, in release 9.0.014.04 and earlier, indicates the time when the strategy started processing the interaction. gpm-ixn-timestamp is configured in the default_skill_data object, from which it is passed to the ActivatePredictiveRouting_v3 subroutine. In URS Strategy Subroutines 9.0.015.00 and higher, gpm-ixn-timestamp is not used, and START_TS must be passed in the default_skill_data parameter. gpmWaitTime (the actual wait time of the interaction in the queue before an agent is selected) is calculated based on the difference between the UTC time when agent is selected minus the START_TS value. **Default value:** no default value **Valid values:** a valid UTC timestamp **Note:** This KVP is mandatory for Genesys Info Mart reporting.	INT	GPM_FACT.START_DATE_TIME_KEY
KVP	**Description**	**KVP Type**	**Info Mart Database Target**

Operations: Updating, Maintenance, Logging, Troubleshooting

This section contains topics that provide information to help you keep your Predictive Routing deployment running smoothly, monitor it's functions, and troubleshoot any issues that might arise.

It includes the following topics:

Agent State Connector

The following sections offer ASC-specific operations and user information.

- Understanding Agent Login States
- Guidelines for Configuration Options Values

You might also find the ASC-specific sections of the following pages helpful:

- Starting and Stopping
- System Monitoring and Logging

Understanding Agent Login States

When you review the fields in the Agent Profile or Dataset configuration windows, you might find a field containing numerical values indicating an agent's login state. (This field is not present if your dataset does not include it.)

The numerical values correspond to the following agent login states:

- 4 - WaitForNextCall (Ready)
- 5 - OffHook
- 6 - CallDialing
- 7 - CallRinging
- 8 - NotReadyForNextCall
- 9 - AfterCallWork
- 13 - CallOnHold
- 16 - ASM_Engaged
- 17 - ASM_Outbound
- 18 - CallUnknown
- 19 - CallConsult
- 20 - CallInternal
- 21 - CallOutbound
- 22 - CallInbound
- 23 - LoggedOut

Guidelines for Configuration Options Values

> **Tip**
> All option changes for ASC take effect after the application restart.

agents-batch-size

The **agents-batch-size** option defines the number of agent configuration profiles submitted to the Predictive Routing web application during ASC startup. A higher number improves startup timing, however setting this option too high leads to timeouts for the API requests. Setting the value to `500` proved to be optimal in a test environment.

reset-jop-on-startup

> **Important**
> This option is removed in Agent State Connector (ASC) release 9.0.006.08 and higher. To delete agents, delete your current agent profile schema in the Predictive Routing application, and then upload an updated schema.

Setting the **reset-jop-on-startup** option to `true` tells the ASC application to send a request to clean up previously-stored agent profiles in the web application database on startup. If this request fails, the old agent profiles can be manually removed from Mongo DB.

stat-srv-ws-conn

> **Important**
> This option has been removed in Agent State Connector (ASC) release 9.0.006.08 and higher. ASC now supports warm standby connections by default.

Setting the **stat-srv-ws-conn** option to `true` specifies that ASC should activate a warm-standby connection to a Stat Server primary/backup pair. With a warm-standby connection to a Stat Server pair, ASC resubscribes for agent statistics to the new primary Stat Server after a Stat Server switchover.

skip-groups

If the **skip-groups** option is set to `true`, ASC ignores reading of agent groups and ignores all events from Configuration Server that are connected with updating agent groups. Genesys recommends that you use this option to improve ASC initialization time if agent group names are not used to define the

target groups of agents for interaction routing.

include-person-annex-sections

Specifies which sections of the Person configuration object Annex tab are parsed by ASC when an agent profile is uploaded to the Journey Optimization Platform (JOP). Other Annex sections are skipped. If both the **include-person-annex-sections** and **ignore-person-annex-sections** options have values specified, ASC ignores the value set for **ignore-person-annex-sections** and works only with the **include-person-annex-sections** setting.

ignore-person-annex-sections

Specifies which sections of the Person configuration object Annex tab are skipped by ASC when agent profile is uploaded to the Journey Optimization Platform. By default, the sections related to Interaction Workspace are skipped. If both the **include-person-annex-sections** and **ignore-person-annex-sections** options have values specified, ASC ignores the value set for **ignore-person-annex-sections** and works only with the **include-person-annex-sections** setting.

ignore-employee-ids

ASC ignores agents with employee ids that are specified in the **ignore-employee-ids** option. Use commas as delimiters between employee ids. For example, this option can be used to skip events for agents with a large amount of configuration data stored in the Person configuration object if interactions to these agents are not routed using Predictive Routing.

cfg-reading-threads-size

Used for reading agents and groups from Configuration Server using a multithreading approach. By default, the value is `100`. The maximum value is `2000`.

jop-update-thread-wait-timeout

Specifies the thread waiting timeout used for the JOP subscription process to avoid a busy loop.

cfg-retry-request-attempts

Specifies the number of retry attempts for reading agents and groups from Configuration Server, if any connection issues with Configuration Server occur. By default, the value is 1.

ss-subscription-timeout

The timeout period observed after each agent subscription to Stat Server. Used to throttle requests for statistics subscriptions to Stat Server.

AI Core Services Monitoring and Logging

AICS uses a number of logs to track the status of the various containers: Tango, Gunicorn worker containers, MongoDB, Minio (in release 9.0.013.01 and higher), and NGINX (in releases prior to 9.0.013.01). This topic provides the following information on monitoring the AI Core Services containers:

- Configure AICS Log Settings
- Access the Logs for AICS
- Using Logs to Troubleshoot Common Errors
- Checking the Logs for AICS Containers in HA Environments

The 200 response code indicates that a service is running normally. If you receive a different response code that indicates an issue, check additional logs.

Configure AICS Log Settings

To configure logging for AICS, set the following:

1. Configure the LOG_LEVEL environment variable.
2. If necessary, change the maximum log size (set to 100m by default).

Access the Logs for AICS

> **Important**
> To access the logs conveniently, you must add your username (PR_USER, by default) to the following Linux group: `sudo usermod -aG systemd-journal pm`. Otherwise, you must use the **sudo** command to see the logs for the various containers. The following steps assume that you are already connected to the shell of the hosts where AICS is deployed. You can issue these commands from any machine that is part of the AICS deployment cluster.

Tango Logs

To access Tango logs, use the following command from any machine in the cluster:

```
$ journalctl CONTAINER_NAME=tango -o cat
2019-07-12 12:09:26,755 [30] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:12:09:26 +0000] "GET /status HTTP/1.1" 200 22 "-" "curl/7.29.0"
2019-07-12 12:09:36,986 [30] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:12:09:36 +0000] "GET /status HTTP/1.1" 200 22 "-" "curl/7.29.0"
2019-07-12 12:09:47,217 [30] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:12:09:47 +0000] "GET /status HTTP/1.1" 200 22 "-" "curl/7.29.0"
2019-07-12 12:09:57,446 [30] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:12:09:57 +0000] "GET /status HTTP/1.1" 200 22 "-" "curl/7.29.0"
2019-07-12 12:10:07,678 [30] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:12:10:07 +0000] "GET /status HTTP/1.1" 200 22 "-" "curl/7.29.0"
2019-07-12 12:10:17,909 [30] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:12:10:17 +0000] "GET /status HTTP/1.1" 200 22 "-" "curl/7.29.0"
2019-07-12 12:10:28,140 [30] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:12:10:28 +0000] "GET /status HTTP/1.1" 200 22 "-" "curl/7.29.0"
```

MongoDB Logs

To access MongoDB logs, use the following command from any machine in the cluster:

```
$ journalctl CONTAINER_NAME=mongo -o cat
2019-07-12T12:07:58.662+0000 I NETWORK  [conn4] received client metadata from
172.18.0.4:56298 conn4: { driver: { name: "PyMongo", version: "3.7.2" }, os: { type: "Linux",
name: "Linux", architecture: "x86_64", v
2019-07-12T12:07:58.684+0000 I NETWORK  [listener] connection accepted from 172.18.0.4:56300
#5 (3 connections now open)
2019-07-12T12:07:58.695+0000 I NETWORK  [conn5] received client metadata from
172.18.0.4:56300 conn5: { driver: { name: "PyMongo", version: "3.7.2" }, os: { type: "Linux",
name: "Linux", architecture: "x86_64", v
2019-07-12T12:07:58.697+0000 I NETWORK  [listener] connection accepted from 172.18.0.4:56302
#6 (4 connections now open)
2019-07-12T12:07:58.707+0000 I NETWORK  [conn6] received client metadata from
172.18.0.4:56302 conn6: { driver: { name: "PyMongo", version: "3.7.2" }, os: { type: "Linux",
name: "Linux", architecture: "x86_64", v
2019-07-12T12:10:01.959+0000 I NETWORK  [listener] connection accepted from 172.18.0.4:56388
#7 (5 connections now open)
2019-07-12T12:10:01.969+0000 I NETWORK  [conn7] received client metadata from
172.18.0.4:56388 conn7: { driver: { name: "PyMongo", version: "3.7.2" }, os: { type: "Linux",
name: "Linux", architecture: "x86_64", v
2019-07-12T12:10:01.970+0000 I NETWORK  [listener] connection accepted from 172.18.0.4:56390
#8 (6 connections now open)
2019-07-12T12:10:01.991+0000 I NETWORK  [conn8] received client metadata from
172.18.0.4:56390 conn8: { driver: { name: "PyMongo", version: "3.7.2" }, os: { type: "Linux",
name: "Linux", architecture: "x86_64", v
2019-07-12T12:10:02.488+0000 I NETWORK  [conn7] end connection 172.18.0.4:56388 (5
connections now open)
2019-07-12T12:10:02.488+0000 I NETWORK  [conn8] end connection 172.18.0.4:56390 (4
connections now open)
```

Model Training Workers Logs

- By default, a cluster contains two running model training worker containers. As a result, the container name changes to reflect which container logs are to be reviewed.

To access Model Training Worker logs, use the following command from any machine in the cluster:

```
$ journalctl CONTAINER_NAME=workers_model_training_1 -o cat
2019-07-12 12:14:39 [INFO] [] Using gunicorn timeout value = [600]
2019-07-12 12:14:39 [INFO] [] Using gunicorn keepalive value = [2]
```

```
2019-07-12 12:14:39 [INFO] [] Using default value for gunicorn workers = [2]
2019-07-12 12:14:39 [INFO] [] TLS disabled. Files /data/ssl/tango.crt and /data/ssl/tango.key
not found in /data/ssl
2019-07-12 12:14:39 [INFO] [] Logs level is set to INFO level. Access logs redirection to
syslog enabled for gunicorn
2019-07-12 12:14:39 [INFO] [] Starting executor - topic model_training, timeout 2, sleeping
timeout 30
2019-07-12 12:14:40,197 [15] INFO  <solariat> fields.py:723  Successfully configured signed
pickle.
 * Serving Flask app "worker_status" (lazy loading)
 * Environment: production
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
127.0.0.1 - - [12/Jul/2019 12:14:49] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:14:59] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:10] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:20] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:30] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:40] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:50] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:00] "GET /status HTTP/1.1" 200 -

$ journalctl CONTAINER_NAME=workers_model_training_2 -o cat
2019-07-12 12:14:39 [INFO] [] Using gunicorn timeout value = [600]
2019-07-12 12:14:39 [INFO] [] Using gunicorn keepalive value = [2]
2019-07-12 12:14:39 [INFO] [] Using default value for gunicorn workers = [2]
2019-07-12 12:14:39 [INFO] [] TLS disabled. Files /data/ssl/tango.crt and /data/ssl/tango.key
not found in /data/ssl
2019-07-12 12:14:39 [INFO] [] Logs level is set to INFO level. Access logs redirection to
syslog enabled for gunicorn
2019-07-12 12:14:39 [INFO] [] Starting executor - topic model_training, timeout 2, sleeping
timeout 30
2019-07-12 12:14:40,476 [15] INFO  <solariat> fields.py:723  Successfully configured signed
pickle.
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
 * Serving Flask app "worker_status" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
127.0.0.1 - - [12/Jul/2019 12:14:49] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:00] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:10] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:20] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:30] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:40] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:50] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:00] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:10] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:20] "GET /status HTTP/1.1" 200 -
```

Analysis Workers Logs

To access Analysis Workers logs, use the following command from any machine in the cluster:

```
$ journalctl CONTAINER_NAME=workers_analysis_1 -o cat
2019-07-12 12:14:40 [INFO] [] Using gunicorn timeout value = [600]
2019-07-12 12:14:40 [INFO] [] Using gunicorn keepalive value = [2]
2019-07-12 12:14:40 [INFO] [] Using default value for gunicorn workers = [2]
2019-07-12 12:14:40 [INFO] [] TLS disabled. Files /data/ssl/tango.crt and /data/ssl/tango.key
```

```
not found in /data/ssl
2019-07-12 12:14:40 [INFO] [] Logs level is set to INFO level. Access logs redirection to
syslog enabled for gunicorn
2019-07-12 12:14:40 [INFO] [] Starting executor - topic analysis, timeout 2, sleeping timeout
30
2019-07-12 12:14:41,140 [15] INFO  <solariat> fields.py:723  Successfully configured signed
pickle.
 * Serving Flask app "worker_status" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [12/Jul/2019 12:14:50] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:00] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:10] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:20] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:30] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:41] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:51] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:01] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:11] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:21] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:31] "GET /status HTTP/1.1" 200 -
```

Purging Workers Logs

To access Purging Worker logs, use the following command from any machine in the cluster:

```
$ journalctl CONTAINER_NAME=workers_purging_1 -o cat
2019-07-12 12:14:41 [INFO] [] Using gunicorn timeout value = [600]
2019-07-12 12:14:41 [INFO] [] Using gunicorn keepalive value = [2]
2019-07-12 12:14:41 [INFO] [] Using default value for gunicorn workers = [2]
2019-07-12 12:14:41 [INFO] [] TLS disabled. Files /data/ssl/tango.crt and /data/ssl/tango.key
not found in /data/ssl
2019-07-12 12:14:41 [INFO] [] Logs level is set to INFO level. Access logs redirection to
syslog enabled for gunicorn
2019-07-12 12:14:41 [INFO] [] Starting executor - topic purging, timeout 2, sleeping timeout
30
2019-07-12 12:14:41,871 [15] INFO  <solariat> fields.py:723  Successfully configured signed
pickle.
 * Serving Flask app "worker_status" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [12/Jul/2019 12:14:51] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:01] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:11] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:21] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:31] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:41] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:51] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:01] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:11] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:21] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:32] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:42] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:52] "GET /status HTTP/1.1" 200 -
```

Dataset Upload Workers Logs

To access Dataset Upload Worker logs, use the following command from any machine in the cluster:

```
$ journalctl CONTAINER_NAME=workers_dataset_upload_1 -o cat
2019-07-12 12:14:42 [INFO] [] Using gunicorn timeout value = [600]
2019-07-12 12:14:42 [INFO] [] Using gunicorn keepalive value = [2]
2019-07-12 12:14:42 [INFO] [] Using default value for gunicorn workers = [2]
2019-07-12 12:14:42 [INFO] [] TLS disabled. Files /data/ssl/tango.crt and /data/ssl/tango.key
not found in /data/ssl
2019-07-12 12:14:42 [INFO] [] Logs level is set to INFO level. Access logs redirection to
syslog enabled for gunicorn
2019-07-12 12:14:42 [INFO] [] Starting executor - topic dataset_upload, timeout 2, sleeping
timeout 30
2019-07-12 12:14:42,563 [15] INFO  <solariat> fields.py:723  Successfully configured signed
pickle.
 * Serving Flask app "worker_status" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [12/Jul/2019 12:14:52] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:02] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:12] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:22] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:32] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:42] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:15:52] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:02] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:12] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:22] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:32] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:42] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:16:52] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:17:02] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:17:12] "GET /status HTTP/1.1" 200 -
127.0.0.1 - - [12/Jul/2019 12:17:23] "GET /status HTTP/1.1" 200 -
```

Example Commands to Locate Logs

To get last 100 lines of the Tango log, run:

```
$ journalctl CONTAINER_NAME=tango -n 100 -o cat
```

To get last 60 minutes of the Tango log, run:

```
$ journalctl CONTAINER_NAME=tango --since="1 hour ago" -o cat
```

To get the last ten hours of the MongoDB log, run:

```
$ journalctl CONTAINER_NAME=mongo --since="10 hour ago" -o cat
```

To tail Tango logs, run:

```
$ journalctl CONTAINER_NAME=tango -f -o cat
```

Common Errors with AICS

This section provides information about how to diagnose and troubleshoot problems when running the AICS application deployed in Docker Containers. This information applies to both HA and single node installations.

Volume

The default permissions on shared volumes are not configurable. If you are working with applications that require permissions different from the shared volume defaults at container runtime, you need to either use non-host-mounted volumes or find a way to make the applications work with the default file permissions.

The only volume that is required to run the AICS application mounted locally on the hosts is **/datadir**. This could be a mount from the network or locally available on the filesystem of the hosts.

If there is an existing mount already on the system with the similar name or there is an existing data in the folder, rename the mount or move the files to a different location within the system. During the installation of the AICS stack, it is assumed that the path exists and there is no existing data on the volume.

Permissions Errors

The current user might not be added to the Docker group or does not have enough permissions to perform Docker related operations.

```
$ docker ps
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/
docker.sock: Get http://%2Fvar%2Frun%2Fdocker.sock/v1.30/containers/json: dial unix /var/run/
docker.sock: connect: permission denied
```

During installation of the AICS application, ad the configured user (by default, PR_USER) to the `docker` and `log` groups. These permissions are necessary for administration tasks related to Docker and to identify any application-related issues.

If your user can access system logs, you should be able to discover common application errors related to functionality of the AICS stack.

Stopped Containers

There are scenarios in which the containers can fail and move from the running state to the stopped state. To check the status of all the running containers on the hosts, use the following command:

```
$ docker ps
CONTAINER ID        IMAGE                                   COMMAND
CREATED             STATUS                      PORTS               NAMES
a9ed052f99d6        jop_tango:2019_07_11_16_00              "./docker-entrypoi..."   27
minutes ago      Up 27 minutes (healthy)                         workers_dataset_upload_1
3656dae9e4cf        jop_tango:2019_07_11_16_00              "./docker-entrypoi..."   27
minutes ago      Up 27 minutes (healthy)                         workers_purging_1
de9640523ad4        jop_tango:2019_07_11_16_00              "./docker-entrypoi..."   28
minutes ago      Up 28 minutes (healthy)                         workers_analysis_1
9f461a9dbe0f        jop_tango:2019_07_11_16_00              "./docker-entrypoi..."   28
```

```
minutes ago      Up 28 minutes (healthy)                          workers_model_training_2
a380bc06279b        jop_tango:2019_07_11_16_00                 "./docker-entrypoi..."   28
minutes ago      Up 28 minutes (healthy)                          workers_model_training_1
871e6f2f1f7f        jop_tango:2019_07_11_16_00                 "./docker-entrypoi..."   28
minutes ago      Up 28 minutes (healthy)   0.0.0.0:443->3031/tcp    tango
72cd51047081        minio/minio:RELEASE.2018-07-23T18-34-49Z   "sh -c 'mkdir -p /..."   29
minutes ago      Up 29 minutes (healthy)   0.0.0.0:9000->9000/tcp   scripts_minio_1
97122976b30d        mongo:3.6                                  "docker-entrypoint..."   29
minutes ago      Up 29 minutes             27017/tcp                mongo
```

This indicates all the containers are running correctly, with no issues. However, to identify any stopped containers, use the following command:

```
$ docker ps -a
CONTAINER ID        IMAGE                                          COMMAND
CREATED             STATUS                          PORTS                     NAMES
a9ed052f99d6        jop_tango:2019_07_11_16_00                 "./docker-entrypoi..."   28
minutes ago      Up 28 minutes (healthy)
workers_dataset_upload_1
3656dae9e4cf        jop_tango:2019_07_11_16_00                 "./docker-entrypoi..."   28
minutes ago      Up 28 minutes (healthy)                                 workers_purging_1
de9640523ad4        jop_tango:2019_07_11_16_00                 "./docker-entrypoi..."   28
minutes ago      Up 28 minutes (healthy)                                 workers_analysis_1
9f461a9dbe0f        jop_tango:2019_07_11_16_00                 "./docker-entrypoi..."   28
minutes ago      Up 28 minutes (healthy)
workers_model_training_2
a380bc06279b        jop_tango:2019_07_11_16_00                 "./docker-entrypoi..."   28
minutes ago      Up 28 minutes (healthy)
workers_model_training_1
871e6f2f1f7f        jop_tango:2019_07_11_16_00                 "./docker-entrypoi..."   29
minutes ago      Exited (137) 8 seconds ago                              tango
72cd51047081        minio/minio:RELEASE.2018-07-23T18-34-49Z   "sh -c 'mkdir -p /..."   29
minutes ago      Up 29 minutes (healthy)      0.0.0.0:9000->9000/tcp   scripts_minio_1
97122976b30d        mongo:3.6                                  "docker-entrypoint..."   29
minutes ago      Up 29 minutes                27017/tcp                mongo
```

In the example above, the Tango container has failed and the state has transitioned from `running` to `exited`. To identify the cause of the container failure, use the following command:

```
$ docker inspect --format '{{ json .State.ExitCode }}' tango
137
```

This provides the exit code for the container. If the code is not familiar, check online resources, such as Exit Codes With Special Meanings in the Advanced Bash-Scripting Guide to identify the cause of the failure.

To determine whether the failure is caused by a memory issue or some other cause, use the following command:

```
$ docker inspect --format '{{ json .State.OOMKilled }}' tango
true
```

This indicates whether the container was able to allocate or utilize the amount of memory required to run the application. If there was insufficient memory, free up some memory from the system or kill unnecessary memory-intensive processes.

CPU Issues

CPU issues can be difficult to identify and debug because CPU is a compressible resource, unlike

memory. When memory requests exceed the limit, the kernel kills the process. When CPU exceeds the limit, the kernel simply allocates that process less CPU time, making it run slower. The healthcheck configured in the containers automatically detects unresponsive containers and then recreates them, making it more likely in this situation that the container recovers in good time.

The installation script ensures that minimum CPU and memory requirements are fulfilled during the setups of AICS, but these can change over time. If the resources are increased or you plan to change them, there is no need to run the installation script again. Instead, stop and then restart the application, following the procudure provided in Start and Stop AICS.

Restarting an Exited Container

The container can be brought up normally by issuing the following command. If it does not return to a normal running state, you will need to continue troubleshooting.

```
$ docker start tango
tango
```

Exit Codes With Special Meanings in the Advanced Bash-Scripting Guide lists some exit codes you might encounter.

The following links provide more information to help troubleshoot Docker containers:

- docker service ps
- docker inspect

Configure Maximum Log Size

GPR 9.0.013.01 and higher has a default maximum log file size of 100m. If you are running an earlier version of AICS or if your environment requires a different setting, use the instructions in this section to configure the log file size.

In a single-server environment, execute the commands in this section in your AICS server. In a high availability (HA) environment, review the output of the `docker service ls` command executed on node-1:

- Ensure that there are three tango instances.
- Note down how many worker containers you are running.

Perform the following steps to change the log file size setting:

1. Open the following files on your single server or node-1, depending on your environment:
   - **tango-swarm.yml**
   - **worker-swarm.yml**
2. Check that both files contain the following section at the same level as the `deploy:` section:

   ```
   logging:
   ```

```
        options:
         max-size: 100m
```

3. After you make changes, execute the `bash restart.sh` command on your single server/node-1. This executes a rolling restart of all containers.
4. Check the health of the system by executing the following command on your single server/node-1: `docker service ls`.
   Verify that the number of instances of tango and the workers containers is the same as when you started.
5. To clean up old log files that are not needed anymore, use the following Docker `prune` commands:

```
docker container prune -f
docker volume prune -f
docker network prune -f
```

Checking the Logs for HA AICS Containers

To access AICS logs when the services are running in a HA architecture, execute the following commands on any node in the cluster:

For Tango Logs

```
$ docker service logs tango_tango
tango_tango.0.quljwdycmeq0@ip-172-31-43-210.eu-west-1.compute.internal    | 2019-07-12
11:33:04,722 [32] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:11:33:04 +0000] "GET /health/runtime HTTP/1.1" 200 12 "-" "curl/7.29.0"
tango_tango.0.quljwdycmeq0@ip-172-31-43-210.eu-west-1.compute.internal    | 2019-07-12
11:33:14,977 [30] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:11:33:14 +0000] "GET /health/runtime HTTP/1.1" 200 12 "-" "curl/7.29.0"
tango_tango.0.quljwdycmeq0@ip-172-31-43-210.eu-west-1.compute.internal    | 2019-07-12
11:33:25,204 [32] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:11:33:25 +0000] "GET /health/runtime HTTP/1.1" 200 12 "-" "curl/7.29.0"
tango_tango.0.quljwdycmeq0@ip-172-31-43-210.eu-west-1.compute.internal    | 2019-07-12
11:33:35,432 [32] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:11:33:35 +0000] "GET /health/runtime HTTP/1.1" 200 12 "-" "curl/7.29.0"
tango_tango.0.quljwdycmeq0@ip-172-31-43-210.eu-west-1.compute.internal    | 2019-07-12
11:33:45,659 [32] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:11:33:45 +0000] "GET /health/runtime HTTP/1.1" 200 12 "-" "curl/7.29.0"
tango_tango.0.quljwdycmeq0@ip-172-31-43-210.eu-west-1.compute.internal    | 2019-07-12
11:33:55,886 [32] INFO  <gunicorn-access> glogging.py:353  127.0.0.1 - - [12/Jul/
2019:11:33:55 +0000] "GET /health/runtime HTTP/1.1" 200 12 "-" "curl/7.29.0"
```

For MongoDB Logs

```
$ docker service logs mongo_mongo1
I NETWORK  [conn59] end connection 127.0.0.1:47132 (37 connections now open)
mongo_mongo1.1.hju604bk6bzy@ip-172-31-43-210.eu-west-1.compute.internal    |
2019-07-12T11:35:01.733+0000 I NETWORK  [listener] connection accepted from 10.0.0.5:33372
#60 (38 connections now open)
mongo_mongo1.1.hju604bk6bzy@ip-172-31-43-210.eu-west-1.compute.internal    |
2019-07-12T11:35:01.744+0000 I NETWORK  [conn60] received client metadata from 10.0.0.5:33372
conn60: { driver: { name: "PyMongo", version: "3.7.2" }, os: { type: "Linux", name: "Linux",
```

```
architecture: "x86_64", version: "3.10.0-862.14.4.el7.x86_64" }, platform: "CPython
3.6.8.final.0" }
mongo_mongo1.1.hju604bk6bzy@ip-172-31-43-210.eu-west-1.compute.internal    |
2019-07-12T11:35:01.745+0000 I NETWORK  [listener] connection accepted from 10.0.0.5:33376
#61 (39 connections now open)
mongo_mongo1.1.hju604bk6bzy@ip-172-31-43-210.eu-west-1.compute.internal    |
2019-07-12T11:35:01.755+0000 I NETWORK  [conn61] received client metadata from 10.0.0.5:33376
conn61: { driver: { name: "PyMongo", version: "3.7.2" }, os: { type: "Linux", name: "Linux",
architecture: "x86_64", version: "3.10.0-862.14.4.el7.x86_64" }, platform: "CPython
3.6.8.final.0" }
mongo_mongo1.1.hju604bk6bzy@ip-172-31-43-210.eu-west-1.compute.internal    |
2019-07-12T11:35:02.263+0000 I NETWORK  [conn61] end connection 10.0.0.5:33376 (38
connections now open)
mongo_mongo1.1.hju604bk6bzy@ip-172-31-43-210.eu-west-1.compute.internal    |
2019-07-12T11:35:02.263+0000 I NETWORK  [conn60] end connection 10.0.0.5:33372 (37
connections now open)

$ docker service logs mongo_mongo2
 NETWORK  [LogicalSessionCacheRefresh] Starting new replica set monitor for rs0/
mongo_mongo1:27017,mongo_mongo2:27017,mongo_mongo3:27017
mongo_mongo2.1.kk93g753swsc@ip-172-31-41-16.eu-west-1.compute.internal    |
2019-07-12T11:40:01.634+0000 I NETWORK  [listener] connection accepted from 127.0.0.1:40506
#27 (16 connections now open)
mongo_mongo2.1.kk93g753swsc@ip-172-31-41-16.eu-west-1.compute.internal    |
2019-07-12T11:40:01.643+0000 I NETWORK  [conn27] received client metadata from
127.0.0.1:40506 conn27: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB
Internal Client", version: "3.6.13" }, os: { type: "Linux", name: "Ubuntu", architecture:
"x86_64", version: "16.04" } }
mongo_mongo2.1.kk93g753swsc@ip-172-31-41-16.eu-west-1.compute.internal    |
2019-07-12T11:40:01.651+0000 I NETWORK  [conn27] end connection 127.0.0.1:40506 (15
connections now open)
mongo_mongo2.1.kk93g753swsc@ip-172-31-41-16.eu-west-1.compute.internal    |
2019-07-12T11:40:02.000+0000 I NETWORK  [listener] connection accepted from 10.0.0.5:44322
#28 (16 connections now open)
mongo_mongo2.1.kk93g753swsc@ip-172-31-41-16.eu-west-1.compute.internal    |
2019-07-12T11:40:02.016+0000 I NETWORK  [conn28] received client metadata from 10.0.0.5:44322
conn28: { driver: { name: "PyMongo", version: "3.7.2" }, os: { type: "Linux", name: "Linux",
architecture: "x86_64", version: "3.10.0-862.14.4.el7.x86_64" }, platform: "CPython
3.6.8.final.0" }
mongo_mongo2.1.kk93g753swsc@ip-172-31-41-16.eu-west-1.compute.internal    |
2019-07-12T11:40:02.534+0000 I NETWORK  [conn28] end connection 10.0.0.5:44322 (15
connections now open)

$ docker service logs mongo_mongo3
I NETWORK  [LogicalSessionCacheRefresh] Starting new replica set monitor for rs0/
mongo_mongo1:27017,mongo_mongo2:27017,mongo_mongo3:27017
mongo_mongo3.1.3meg8sc7dvl9@ip-172-31-34-117.eu-west-1.compute.internal    |
2019-07-12T11:40:01.925+0000 I NETWORK  [listener] connection accepted from 127.0.0.1:50938
#28 (16 connections now open)
mongo_mongo3.1.3meg8sc7dvl9@ip-172-31-34-117.eu-west-1.compute.internal    |
2019-07-12T11:40:01.935+0000 I NETWORK  [conn28] received client metadata from
127.0.0.1:50938 conn28: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB
Internal Client", version: "3.6.13" }, os: { type: "Linux", name: "Ubuntu", architecture:
"x86_64", version: "16.04" } }
mongo_mongo3.1.3meg8sc7dvl9@ip-172-31-34-117.eu-west-1.compute.internal    |
2019-07-12T11:40:01.944+0000 I NETWORK  [conn28] end connection 127.0.0.1:50938 (15
connections now open)
mongo_mongo3.1.3meg8sc7dvl9@ip-172-31-34-117.eu-west-1.compute.internal    |
2019-07-12T11:40:01.999+0000 I NETWORK  [listener] connection accepted from 10.0.0.5:47094
#29 (16 connections now open)
mongo_mongo3.1.3meg8sc7dvl9@ip-172-31-34-117.eu-west-1.compute.internal    |
2019-07-12T11:40:02.013+0000 I NETWORK  [conn29] received client metadata from 10.0.0.5:47094
conn29: { driver: { name: "PyMongo", version: "3.7.2" }, os: { type: "Linux", name: "Linux",
```

```
architecture: "x86_64", version: "3.10.0-862.14.4.el7.x86_64" }, platform: "CPython
3.6.8.final.0" }
mongo_mongo3.1.3meg8sc7dvl9@ip-172-31-34-117.eu-west-1.compute.internal    |
2019-07-12T11:40:02.533+0000 I NETWORK  [conn29] end connection 10.0.0.5:47094 (15
connections now open)
```

And so on, for however many MongoDB nodes you have configured.

For Workers Logs

```
$ docker service logs workers_analysis
workers_analysis.2.zas8ia65ficd@ip-172-31-43-210.eu-west-1.compute.internal    | 127.0.0.1 -
- [12/Jul/2019 11:41:09] "GET /status HTTP/1.1" 200 -
workers_analysis.2.zas8ia65ficd@ip-172-31-43-210.eu-west-1.compute.internal    | 127.0.0.1 -
- [12/Jul/2019 11:41:19] "GET /status HTTP/1.1" 200 -
workers_analysis.2.zas8ia65ficd@ip-172-31-43-210.eu-west-1.compute.internal    | 127.0.0.1 -
- [12/Jul/2019 11:41:29] "GET /status HTTP/1.1" 200 -
workers_analysis.2.zas8ia65ficd@ip-172-31-43-210.eu-west-1.compute.internal    | 127.0.0.1 -
- [12/Jul/2019 11:41:39] "GET /status HTTP/1.1" 200 -
workers_analysis.2.zas8ia65ficd@ip-172-31-43-210.eu-west-1.compute.internal    | 127.0.0.1 -
- [12/Jul/2019 11:41:49] "GET /status HTTP/1.1" 200 -
workers_analysis.2.zas8ia65ficd@ip-172-31-43-210.eu-west-1.compute.internal    | 127.0.0.1 -
- [12/Jul/2019 11:42:00] "GET /status HTTP/1.1" 200 -
workers_analysis.2.zas8ia65ficd@ip-172-31-43-210.eu-west-1.compute.internal    | 127.0.0.1 -
- [12/Jul/2019 11:42:10] "GET /status HTTP/1.1" 200 -
workers_analysis.2.zas8ia65ficd@ip-172-31-43-210.eu-west-1.compute.internal    | 127.0.0.1 -
- [12/Jul/2019 11:42:20] "GET /status HTTP/1.1" 200 -

$ docker service logs workers_model_training
workers_model_training.1.rei8sjqnrpe4@ip-172-31-34-117.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:41:59] "GET /status HTTP/1.1" 200 -
workers_model_training.1.rei8sjqnrpe4@ip-172-31-34-117.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:42:09] "GET /status HTTP/1.1" 200 -
workers_model_training.1.rei8sjqnrpe4@ip-172-31-34-117.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:42:19] "GET /status HTTP/1.1" 200 -
workers_model_training.1.rei8sjqnrpe4@ip-172-31-34-117.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:42:29] "GET /status HTTP/1.1" 200 -
workers_model_training.1.rei8sjqnrpe4@ip-172-31-34-117.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:42:39] "GET /status HTTP/1.1" 200 -
workers_model_training.1.rei8sjqnrpe4@ip-172-31-34-117.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:42:49] "GET /status HTTP/1.1" 200 -
workers_model_training.1.rei8sjqnrpe4@ip-172-31-34-117.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:42:59] "GET /status HTTP/1.1" 200 -
workers_model_training.1.rei8sjqnrpe4@ip-172-31-34-117.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:43:09] "GET /status HTTP/1.1" 200 -

$ docker service logs workers_purging
workers_purging.1.q2ak4vs2p7ef@ip-172-31-41-16.eu-west-1.compute.internal    | 127.0.0.1 - -
[12/Jul/2019 11:42:26] "GET /status HTTP/1.1" 200 -
workers_purging.1.q2ak4vs2p7ef@ip-172-31-41-16.eu-west-1.compute.internal    | 127.0.0.1 - -
[12/Jul/2019 11:42:36] "GET /status HTTP/1.1" 200 -
workers_purging.1.q2ak4vs2p7ef@ip-172-31-41-16.eu-west-1.compute.internal    | 127.0.0.1 - -
[12/Jul/2019 11:42:46] "GET /status HTTP/1.1" 200 -
workers_purging.1.q2ak4vs2p7ef@ip-172-31-41-16.eu-west-1.compute.internal    | 127.0.0.1 - -
[12/Jul/2019 11:42:56] "GET /status HTTP/1.1" 200 -
workers_purging.1.q2ak4vs2p7ef@ip-172-31-41-16.eu-west-1.compute.internal    | 127.0.0.1 - -
[12/Jul/2019 11:43:07] "GET /status HTTP/1.1" 200 -
workers_purging.1.q2ak4vs2p7ef@ip-172-31-41-16.eu-west-1.compute.internal    | 127.0.0.1 - -
[12/Jul/2019 11:43:17] "GET /status HTTP/1.1" 200 -
workers_purging.1.q2ak4vs2p7ef@ip-172-31-41-16.eu-west-1.compute.internal    | 127.0.0.1 - -
[12/Jul/2019 11:43:27] "GET /status HTTP/1.1" 200 -
```

```
workers_purging.1.q2ak4vs2p7ef@ip-172-31-41-16.eu-west-1.compute.internal    | 127.0.0.1 - -
[12/Jul/2019 11:43:37] "GET /status HTTP/1.1" 200 -

$ docker service logs workers_dataset_upload
workers_dataset_upload.1.ulp2r911ubuc@ip-172-31-43-210.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:42:47] "GET /status HTTP/1.1" 200 -
workers_dataset_upload.1.ulp2r911ubuc@ip-172-31-43-210.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:42:57] "GET /status HTTP/1.1" 200 -
workers_dataset_upload.1.ulp2r911ubuc@ip-172-31-43-210.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:43:07] "GET /status HTTP/1.1" 200 -
workers_dataset_upload.1.ulp2r911ubuc@ip-172-31-43-210.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:43:18] "GET /status HTTP/1.1" 200 -
workers_dataset_upload.1.ulp2r911ubuc@ip-172-31-43-210.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:43:28] "GET /status HTTP/1.1" 200 -
workers_dataset_upload.1.ulp2r911ubuc@ip-172-31-43-210.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:43:38] "GET /status HTTP/1.1" 200 -
workers_dataset_upload.1.ulp2r911ubuc@ip-172-31-43-210.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:43:48] "GET /status HTTP/1.1" 200 -
workers_dataset_upload.1.ulp2r911ubuc@ip-172-31-43-210.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:43:58] "GET /status HTTP/1.1" 200 -
workers_dataset_upload.1.ulp2r911ubuc@ip-172-31-43-210.eu-west-1.compute.internal    |
127.0.0.1 - - [12/Jul/2019 11:44:08] "GET /status HTTP/1.1" 200 -
```

To return only the last *N* lines of a log file, use the same commands as above, appending the command `--tail` *N*, as in the following example:

```
$ docker service logs workers_analysis --tail 100
```

To continuously stream output of a log, use the same commands as above, appending the command `-f`, as in the following example:

```
$ docker service logs workers_analysis -f
```

Database Maintenance

This topic provides recommendations for keeping your MongoDB database up-to-date and working correctly. It includes sections covering the following:

- How to Specify a Dedicated Mount Point
- Backing Up and Restoring Your Data

Store MongoDB on a Dedicated Mount Point

MongoDB data can grow quite large depending on the size of your environment and how you configure Predictive Routing. By default, the **docker-compose.yml** file (which you obtain from Docker.com) specifies `/datadir/` as the MongoDB storage location. If you expect to write a lot of data on the MongoDB server, Genesys recommends that you create a separate mount point for that purpose.

- **Why?** If your OS uses the default configuration, the `/datadir/` directory shares storage with the root (`/`). If your Predictive Routing data files or logs grow too large, they will fill up your hard drive to a point where you might even lose access to the server.

The instructions below assume you are using Logical Volume Manager (LVM) to handle your disks. This example shows how to create a 20G logical volume for your data:

1. Create the new directory, using the following commands:

   ```
   sudo lvcreate -L 20G -n mongoVolume LVMVolGroup
   mkfs.ext4 /dev/LVMVolGroup/mongoVolume
   mkdir /new-datadir
   mount /dev/LVMVolGroup/projects /new-datadir
   ```

   To make the new mount point permanent, update the **/.../fstab** file.

2. Update those servers running MongoDB (those with the `mongo` label). To do this, open the **docker-compose.yml** file and replace the following text:

   ```
   volumes:
         - /datadir:/data/db
   ```

   with

   ```
   volumes:
         - /new-datadir:/data/db
   ```

3. (Optional) If you initialized the application by running the **start.sh** script, move the data to your new directory using the following commands:

   ```
   cd ./scripts/
   bash stop.sh
   mv /datadir/* /new-datadir/
   ```

4. Start the application:

```
bash start.sh
```

Back Up and Restore MongoDB

This section supplies the commands needed to back up and restore MongoDB in a single-site/single-server AICS deployment. For backup and restore instructions for HA environments, see Backing Up Your Data in the Deploying: High Availability topic.

Using SSL with MongoDB

The procedure below is for MongoDB with SSL enabled. *Genesys recommends that you use SSL.*

- To use SSL, add the `--ssl` parameter to your commands.
  In test environments, you can optionally add `--sslAllowInvalidCertificates` following the `--ssl` parameter.

  In test environments ONLY, if you need to maintain an environment without SSL connections, omit the `--ssl` and `--sslAllowInvalidCertificates` parameters.

Backup MongoDB

Use the following procedure to back up MongoDB in a single-server (single node) environment:

1. Log into the container:

```
docker exec -it mongo bash
```

2. Generate the dump of the **/backup** file:

```
mongodump --ssl --out /data/db/backup --host localhost:27017
```

> **Tip**
> You can view the **/data/db/backup** directory for the container from the base system in the **/datadir/backup** directory if you are using the default defined directory for MongoDB in the **docker-compose.yml** file.

Restore MongoDB

Use the following procedure to restore MongoDB on single node installation:

1. Go to the installation directory:

```
cd "IP_JOP_PRR_<version_number>_ENU_linux/scripts/"
```

2. Stop all the application containers:

```
bash stop.sh
```

3. Restart the MongoDB container:

   ```
   ../docker-compose -f "docker-compose.yml" up -d mongo
   ```

4. Start a bash session on the MongoDB container:

   ```
   docker exec -it mongo bash
   ```

5. Log into the container:

   ```
   mongo solariat_bottle --ssl --eval "db.dropDatabase()"
   ```

6. Restore the dump from the **/backup** file:

   ```
   mongorestore --ssl --drop /data/db/backup --host localhost:27017
    exit
   ```

7. To restart all the application containers, run:

   ```
   bash install.sh
   ```

   and then:

   ```
   bash start.sh
   ```

Troubleshooting

This topic contains a partial and growing collection of tips and best practices to help you identify and resolve frequently-encountered issues with your Predictive Routing deployment.

Model Training Issues

If model training is taking longer than it should, check the following points:

- Is MongoDB correctly using indexes?
  If you see `COLLSCAN` in the log, it indicates that MongoDB is having trouble with the index set-up. To resolve this issue, re-run one of the queries that resulted in the `COLLSCAN` message, appending `.explain()` at the end. The result returned will help locate the problem.

Troubleshooting for a URS-based Predictive Routing Environment

If you are using the URS Strategy Subroutines, the following points can help troubleshoot issues with your configuration:

- Does the interaction in question have user data keys with the prefix `prr` attached.
- Does the key prrResult have the value `ok`.
- If prrResult=`error`, check the prrMessage key for the error message.
- Check if the prrAgentScore key contains a valid score. An empty score might be a result of one of the following issues:
  - Normal operation. For example, the agent logged in and received the interaction after it was already scored.
  - The score for the agent is not present in the URS global map. See below for tips on global map troubleshooting.
  - The list of logged-in agents is out of synchronization between the URS and Journey Optimization Platform (JOP).
  - An interaction transfer. Currently, Predictive Routing is supported only on the first leg of the interaction.
- Check if REST API calls to the scoring engine return any errors.
  - Did the strategy authenticate with JOP REST API and receive a token? If not, check whether the request is sent to the correct account in JOP and the correct user name, password, and API key were provided.
  - Was the scoring request formed correctly? Did the call to the subroutine GetActionFilters returned a valid skill expression or AgentGroup name?
  - Did the scoring REST API request return the scores for all agents in the target group that URS knows

to be logged in? If not, check whether Agent State Connector lost connection to Stat Server.

- Check if the interaction data was prematurely removed from the URS global map.
  - Did the interaction stay in queue longer than the value set in the **global-map-timeout** option? In this case, the data could have been cleaned automatically.
  - Was the subroutine PrrIxnCleanup called at the right time for the interaction?
- Is URS overloaded?
  - Predictive Routing IRD strategy subroutines utilize the URS TimeBehind[] function to detect when URS is overloaded and adjust their behavior accordingly. TimeBehind[] returns a value that indicates the delay, in milliseconds, between the moment an event is received from T-Server and when it is processed by URS for the current interaction.

    If the delay is more than 1000 milliseconds, URS is experiencing an overload and is unable to process interactions in a timely manner. In this case, the Predictive Routing IRD subroutines skip agent scoring and matching any new interactions passing through the ActivatePredictiveRouting subroutine.

    If you configured the strategy to hold out for higher-scoring agents, the hold-out is interrupted and interactions are distributed to agents as they become ready, regardless of agent score, until the overload condition ends.
  - When Predictive Routing is deployed in an Orchestration Server strategy and such an overload condition occurs, the URS SetIdealReadyConditionForORS subroutine exits through the default port when it is called from the ORS strategy.