CX Contact Deployment Guide

CX Contact Current

12/30/2021

Table of Contents

CX Contact Deployment Guide

Welcome to the CX Contact 9.0 Deployment Guide!

Important
CX Contact is being released to pre-approved customers as part of the Early Adopter Program. This means that both the product and the documentation are still under development. As a result, documentation sections might require revision as the product develops. We advise that you use this documentation with care. Before you make changes that could affect the success of your deployment, verify them with your Genesys representatives.

CX Contact contains a set of components that enable you to create, run, and manage outbound voice, SMS, and email campaigns. Some of its key principles and capabilities include the following:

- Has state-of-the-art user interface (UI) and middleware components.
- Is set of microservices that run in Docker containers, each scalable in N+1 horizontal mode.
- Uses Genesys servers on the back end, such as Configuration Server, Outbound Contact Server (OCS), and Stat Server.
- Has Genesys Web Services (GWS) as a prerequisite.

You can deploy CX Contact on premises using one of two methods:

- Docker Compose—Suitable for lab or demo environments only, where no product traffic exists. A Docker Compose deployment is easier than a Kubernetes deployment because all CX Contact and GWS components are deployed using a single docker-compose file on a single VM. There are also fewer prerequisites with a Docker Compose deployment because components such as External Load Balancer, Enterprise Redis, and Network File System are excluded from a Docker Compose deployment.
- Kubernetes—Suitable for production environments but is considerably more complicated because it deploys CX Contact across multiple VMs and presumes availability of all third-party prerequisites, such as External Load Balancer, Enterprise Redis, and Elasticsearch cluster.

Other Considerations

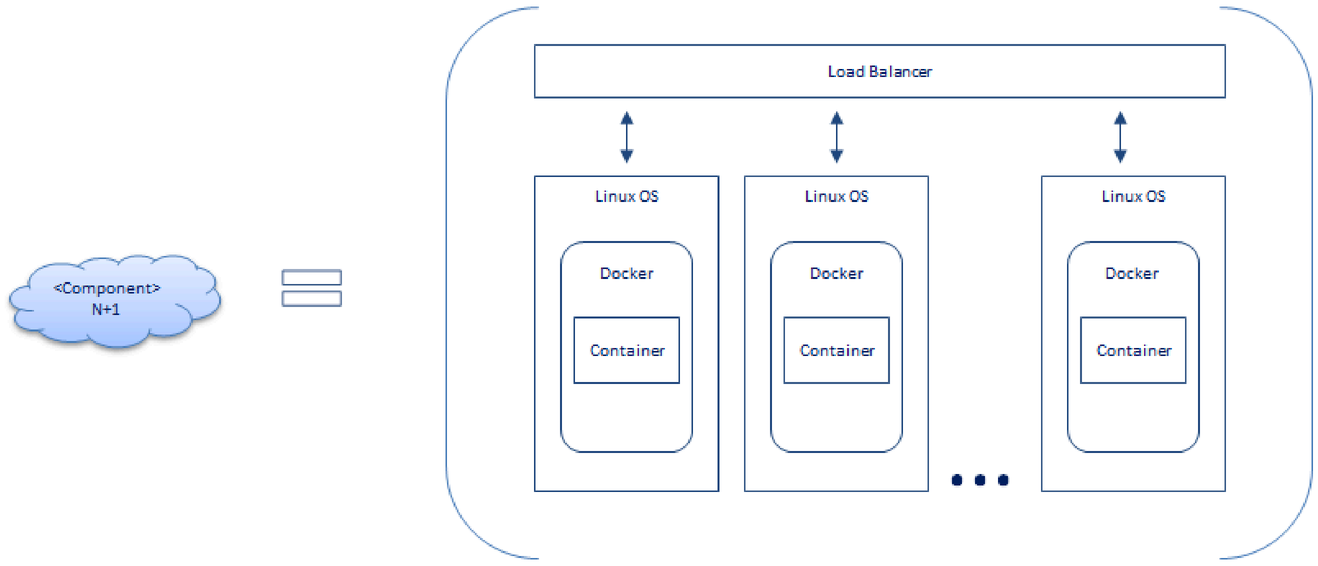
Before deciding on the deployment method you'll use, consider the following additional information about CX Contact:

- Currently, CX Contact supports single region deployments.
- Only a Helm v3 deployment method is supported.

- High Availability (HA) is provided through N+1 architecture.
- Information about Disaster Recovery (DR) is provided through your Genesys representative. Contact the Architecture team for guidance about recommended DR designs.
- CX Contact Compliance Data. Contact your Genesys representative to ensure coverage is provided for your desired calling region.
- Check with your Genesys representative for supported container orchestration technologies.
- Genesys does not deploy and operate databases in on-premises deployments. It is the responsibility of the end user. In a production deployment, data store components (PostgreSQL, Redis, Elasticsearch) must be deployed outside of the Kubernetes cluster and managed by the end user's DBA team. The end user's DBA team is also responsible for ensuring that the data store components are configured with the appropriate scalability, resiliency, and data protection (backups, and so on).

Introduction

All CX Contact components are represented as individual microservices, each executed in Docker containers under N+1 horizontal scaling model principles and running behind internal Elastic Load Balancer.



CX Contact Core Components

The following is an overview of the CX Contact core components:

CX Contact Component	Description
List Builder	• Responsible for importing and exporting contact lists and suppression lists. • Works in conjunction with Outbound Database, which stores the contact lists and suppression lists. • Works in conjunction with REDIS, which stores suppression entries. • Uses compliance data to process records on import.
List Manager	• Responsible for operations related to lists.

CX Contact Component	Description
	• Creates contact lists and suppression lists in Configuration Manager. • Reads Compliance data from a compliance data provider. • Copies files from FTP to NFS for List Builder consumption.
Compliance Manager	• Responsible for dynamic compliance rules validation. • Reads suppression entries from Redis and responds to OCS pre-validation requests.
Campaign Manager	• Responsible for operations related to campaigns. • Executes pre-loading of campaigns. Processing is done in Outbound Database.
Job Scheduler	• Responsible for creating and invoking jobs at the right time, providing for automation of tasks.
Dial Manager	• Responsible for managing SMS and email interactions with Genesys Message Aggregation.
API Aggregator	• This is the entry point of APIs to CX Contact. Ensures APIs stay invariant when internal implementation changes.
User Interface (UI)	• A set of static HTML5 pages served by Nginx.

Using Docker Compose to Deploy CX Contact

This topic and its related subtopics describe everything you need to know about using Docker Compose to deploy CX Contact.

Before you Begin

- Prepare a single VM or set of VMs for the CX Contact deployment.
- Install Docker Engine CE on the VM(s) running RHEL 7.0.
- Pull CX Contact and GWS Docker images from an FTP directory and import them into an internal Docker registry. Your Genesys representative will provide you with access information to the FTP directory.
- Install Docker Compose according to the instructions on the Docker documentation site.
- Obtain Docker-compose files (available for CX Contact and GWS).

Once you've completed these mandatory procedures, return to this manual to learn how to complete an on-premise deployment of CX Contact. Start by reviewing the Prerequisites.

Prerequisites

The table below outlines all prerequisites for a CX Contact deployment using Docker Compose.

Component	Description	Mandatory or Optional
CDP NG Access Credentials	As of CX Contact 9.0.025, Compliance Data Provider Next Generation (CDP NG) is used as a CDP by default. Obtain the necessary access credentials (ID and Secret) before attempting to connect to CDP NG. Request these credentials from Genesys Customer Care.	Mandatory
VM	A single VM running RHEL 7.0 64-bit, 8 CPU cores; 16 GB RAM minimum, 32 GB RAM recommended; at least 100 GB HDD When RHEL/CentOS 7.8 is used, the Kernel must be upgraded to 3.10.0-1160.15.2.el7.x86_64 or later.	Mandatory
Docker	Docker 17.03.2-ce or newer stable	Mandatory
Chrome	The latest version of Chrome must be used as the UI browser.	Mandatory
Container orchestration	Docker Compose and Portainer	Mandatory
Network/DNS	All VMs running CX Contact components should belong to the same local network segment and be interconnected so that all components can communicate over the network. DNS must be present in the network and allow for names resolution. CX Contact components always use FQDNs (not IP addresses) to establish communication to each other.	Mandatory
PostgreSQL	PostgreSQL 9.5+ CX contact supports non-standard Postgre SQL ports for the Data Access Point (DAP) to assist in Disaster Recovery. List Manager, List Builder, and Campaign Manager can all communicate with Postgre SQL via non-standard ports.	Mandatory
SFTP Server	Use when automation capabilities are required	Optional
Genesys Web Services (GWS)	v.9.0. GWS9 is an integral part of the CX Contact Docker Compose	Mandatory

Component	Description	Mandatory or Optional
	deployment. For this reason, you do not need to deploy GWS9 separately. You must push these images to the local Docker registry.	
Genesys core components	v.8.5 or v.8.1 CX Contact components operate with Genesys core services on the back end. It's expected that all voice-processing components (Voice VM and shared services such as GVP) are deployed and running. CX Contact requires a Multi Tenant Configuration Server.	Mandatory

Deploying with Docker Compose

To deploy CX Contact by using Docker Compose, complete the following deployment procedures. The first three procedures are common to both deployment methods. Click the link for the appropriate topic:

Summary of deployment procedures
1. Ensure the Prerequisites are met
2. Create the Outbound Database
3. Create the Outbound Database Access Point
4. Start Outbound Contact Server (OCS)
5. Deploy with Docker Compose

Deploy with Docker Compose

Summary of procedures: Deploy with Docker Compose
1. Obtain the Docker Compose scripts   • Use common CX Contact commands   • Obtain images in disconnected environments
2. Set up the environment: • Use automatic setup (Genesys strongly recommends that you use this method to set up the environment.)   • Use manual setup
3. Log in to CX Contact

Important

Contact Genesys Customer Care regarding downloading CX Contact and GWS Docker images. These images should be pushed to the local registry. Refer to https://docs.docker.com/registry/deploying/ for details. Make note of the GWS Components versions. You may need to enter these versions when performing the initial setup.

Docker Compose scripts

To receive all of the latest files required for the Docker Compose deployment, you must first copy the **Docker Compose** scripts.

To obtain the Docker Compose scripts:

1. Execute the following Bash commands in the order that they appear here:
   - `$ export DEPLOY_CXCONTACT_IMAGE=<deploy_cx_contact_image>`
   - `$ echo "docker run --rm -it -u $(id -u ${USER}):$(id -g ${USER}) -e init=true -v "$(pwd)":/env:rw $DEPLOY_CXCONTACT_IMAGE" > cxc-app.sh`
   - `$ bash cxc-app.sh`

   When asked if you would like to replace with updates found for the **cxc-app.sh** script select **2** to replace.

   - The old file is moved to the `/backup/MMDDYY-hhmm` folder.
     Note: Each update is saved in a new directory. The name of the new directory is the date on which the update was performed and completed.

2. Once you obtain the Docker Compose script make it executable by running the following command:

   ```
   $ chmod +x cxc-app.sh
   ```

   Then, execute the following **Help** command to obtain all of the available commands:

   ```
   $ ./cxc-app.sh help
   ```

   Help Output

   ```
   Welcome to CXContact deployment service. Following commands are available:

    init <deploy_cx_contact image>            - Initial setup
    start                                     - Start CXContact docker-compose environment
    stop                                      - Stop CXContact docker-compose environment
    restart                                   - Restart CXContact docker-compose environment
    status                                    - Get status of all running containers
    provision <deploy_cx_contact image>       - Provisioning CXContact
    cxc-only [on/off]                         - Switch between cxc-only deployment and
    single node (with GWS services) deployment

    save <optional parameters>                - Save docker images in archive
    Available optional parameters for save:
    --only <tag> for e.g cxcontact.            - Will save only specific images
    -o,--output <name>                         - Output names for archive with images and
    import script
    -t,--tag <tag>                             - Will apply new tag to images, for internal
    registries

    Note! <deploy_cx_contact image> parameter is optional. Default - will be used latest
    local image
    If no image found - you will be asked to prompt image name to pull.
   ```

3. Execute the following command to obtain the CX Contact Docker Compose **.yaml** files:

   ```
   $ ./cxc-app.sh init
   ```

At this point, the following occurs:

- You will be prompted for a CX Contact Deployment Service image (if it is not present locally).

Note: The Docker Compose script uses the latest `deploy_cx_contact` image. To use a specific image run `./cxc-app.sh init <image_name>`.

- The Docker Compose script pulls the Deployment Service image and verifies which deployment should be initialized (that is, Docker or Kubernetes).
- The **.yaml** files are copied to the Docker or Kubernetes folder in the same directory in which the Docker Compose script is located.
- You ae asked to configure CX Contact. The default values are suggested for each environment variable. You can replace the default values with values applicable for your environment.

Note: If you are using the local Docker registry, specify it as the value of CXC_DOCKER_REPOSITORY and GWS_DOCKER_REPOSITORY variables. Change default values of GWS components versions to the actual versions of GWS Components images pushed to the local registry.

4. Start the CX Contact Docker Compose Environment by running the following command:

```
$ ./cxc-app.sh start
```

5. By default, as of CX Contact 9.0.025, the Docker Compose **.env** file is configured for CDP NG connectivity. The following default values show how CDP NG is used.

```
# COMPLIANCE DATA SETTINGS
# List Builder embedded CDP_NG Compliance Data
EMBEDDED_COMPLIANCE_DATA_BASEPATH="/list_builder/data/ng_init_data"
# OPTIONAL: List Builder embedded LEGACY CDP Compliance Data
#EMBEDDED_COMPLIANCE_DATA_BASEPATH="/list_builder/init_data"
# Compliance data rule sets:
AREACODE_RULE_SET="AU,CA,GB,NZ,US"
GEO_RULE_SET="AU,CA,GB,NZ,US"
POSTAL_RULE_SET="CA,GB,US"
DNC_RULE_SET="GB,US"
# In order to switch to Legacy CPD, change CDP_NG_URL and CDP_NG_GCLOUD_AUTH to empty
values
CDP_NG_URL="https://api.usw2.pure.cloud/api/v2/outbound/compliancedata"
CDP_NG_GCLOUD_AUTH="https://login.usw2.pure.cloud/oauth/token"
```

- CDP_NG_GCLOUD_ID and CDP_NG_GCLOUD_SECRET are required parameters and do not have a default value.
- These parameters must be requested by creating a CLOUDCON ticket before attempting to upgrade to CX Contact 9.0.025+ or before deploying CX Contact 9.0.025+ for the first time.
- The new CLOUDCON ticket must include the customer name and the name of the person who will receive the keys in the Jira ticket. CDP_NG_GCLOUD_ID=<Must be provided> CDP_NG_GCLOUD_SECRET=<Must be provided>
- To return to CDP Legacy, use the following configuration parameters:

```
EMBEDDED_COMPLIANCE_DATA_BASEPATH="/list_builder/init_data"
CDP_NG_URL=
CDP_NG_GCLOUD_AUTH=
CDP_NG_GCLOUD_ID=
CDP_NG_GCLOUD_SECRET=
```

Common CX Contact commands

CX Contact Procedure	Command
Start CX Contact	$ ./cxc-app.sh start
Stop CX Contact	$ ./cxc-app.sh stop
Restart CX Contact	$ ./cxc-app.sh restart
View current CX Contact status and uptime.	$ ./cxc-app.sh status
Switch between single host deployment and multi host deployment (that is, when GWS and CX Contact are deployed on different hosts).	$ ./cxc-app.sh cxc-only [on/off]
Provision CX Contact	$ ./cxc-app.sh provison **Note:** Provisioning can be executed multiple times.
Revalidate the configuration when provisioning fails.	$ ./cxc-app.sh init
Pull images in a disconnected environment.	$ ./cxc-app.sh init $ ./cxc-app.sh save <optional parameters> **Note:** The images are archived and an be transferred to a Docker host that does not have an internet connection.

Obtain Images in Disconnected Environments

Deployments that are not connected to the Internet must obtain images from the Genesys Engage Docker Repository and archive them. The archived images are then transferred to a Docker host that is not connected to the Internet.

To obtain images from the Genesys Engage Docker Repository run the following command on a computer with access to the Internet and save the images in an archive location.

```
$ ./cxc-app.sh init
$ ./cxc-app.sh save <optional>
```

The archived images must then be transferred to a destination host and used for deployment.

Important

To store the images in your own Docker registry (only required for Kubernetes deployments or your own Docker registry), you can add the **--tag** parameter with your docker-registry. The archived images must be transferred to a destination host and used for deployment.

Set up the environment

Set up the environment either automatically (recommended) or manually. Procedures for each option are provided below.

Set up the environment automatically (Recommended)

If you have already deployed CX Contact using Docker Compose, start at step 1 below. If not, see Docker Compose scripts for details.

1. Obtain the CX Contact Docker Compose **.yaml** files by executing the following command:

   ```
   $ ./cxc-app.sh init
   ```

2. While executing **$ ./cxc-app init.sh**, select the Docker deployment and select **y** when asked if you would like to configure CX Contact.
3. In the directory containing **cxc-app.sh**, execute the following:

   ```
   ./cxc-app.sh provision
   ```

Provisioning will take approximately 1 to 2 minutes.

Set up the environment manually

Important

Genesys recommends this option for advanced users only.

To set up the environment manually, you'll make API requests to GWS. Requests should point to the host name or IP address of the external load balancer for GWS or CX Contact. In Docker Compose deployments, this is the VM where Docker runs.

Complete the procedures in the summary below.

Summary of procedures: Set up environment manually
1. Verify successful start of gws-postgres
2. Check gws-core-environment
3. Check gws-core-auth
4. Create the environment
5. Create the contact center
6. Create the authentication client
7. Create the external_api_client
8. Get the access token for cx_contact

Summary of procedures: Set up environment manually
9. Verify authentication (Optional)
10. Verify clients
11. Create the tenant in api-aggregator
12. Configure session profile
13. Add location in CloudCluser Application

Verify successful start of gws-postgres

To verify the successful start of **gws-postgres**:

```
docker-compose logs gws-postgres
```

This is the expected response:

```
gws-postgres_1                 | LOG:  database system is ready to accept connections
gws-postgres_1                 | LOG:  autovacuum launcher started
```

Check gws-core-environment

To check **gws-core-environment**:

```
curl http://localhost:8091/environment/v3/version
```

Check gws-core-auth

To check **gws-core-auth**:

```
curl  'http://localhost:8095/auth/v3/oauth/
authorize?response_type=code&client_id=cx_contact&redirect_uri=http://localhost/api-
aggregator/v2/login-callback'
```

Check that there is a reply. It will be an error, but it must be received.

Create the environment in GWS

Send a POST request to GWS.

Pass a body parameter in JSON format, called **data**, with the following properties:

Property	Value
username	The super administrator account name for Configuration Server.
appName	Cloud
password	Use the password for the super administrator

Property	Value
	account name.
connectionProtocol	addp
localTimeout	7
remoteTimeout	11
traceMode	CFGTMBoth
tlsEnabled	false
primaryPort	Configuration Server's TCP Listener port.
readOnly	false
primaryAddress	Configuration Server's host name or IP address
locations	/USW1
tenant	Environment

Code sample

```
curl  --user ops:ops  -H "Content-Type: application/json" -X POST http://localhost:8091/
environment/v3/environments  -d  '
{
  "data": {
    "username": "default",
    "appName": "Cloud",
    "password": "password",
    "connectionProtocol": "addp",
    "localTimeout": 7,
    "remoteTimeout": 11,
    "traceMode": "CFGTMBoth",
    "tlsEnabled": false,
    "configServers": [
      {
        "primaryPort": 8888,
        "readOnly": false,
        "primaryAddress": "10.51.30.154",
        "locations": "/USW1"
      }
    ],
    "tenant": "Environment"
  }
}
'
```

Expected response

```
{"status":{"code":0},"path":"/environments/bf032640-9073-435d-9447-718b7cc7dc43"}
```

Important

Take note of the environment ID parameter in the response (**bf032640-9073-435d-9447-718b7cc7dc43**) – it is required for subsequent requests.

Create the contact center

Send a POST request to GWS.

Pass a body parameter in JSON format, called **data**, with the following properties:

Property	Value
environmentid	bf032640-9073-435d-9447-718b7cc7dc43
auth	configServer

Code sample

```
curl  --user ops:ops  -H "Content-Type: application/json" -X POST http://localhost:8091/
environment/v3/contact-centers  -d  '
{
  "data": {
    "environmentId": "bf032640-9073-435d-9447-718b7cc7dc43",
    "domains": ["domain.com"],
    "auth": "configServer"
  }
}
'
```

Expected response

This produces the following result:

```
{"status":{"code":0},"path":"/contact-centers/3952ccd2-a34a-46c1-b51e-8917628554c9"}
```

Create the authentication client

To create the authentication client, send a POST request to GWS.

Pass a body parameter in JSON format, called **data**, with the following properties:

Property	Value
clientType	*
internalClient	true
authorizedGrantTypes	refresh_token, implicit, password, client_credentials, authorization_code
redirectURIs	http://10.11.64.16
authorities	ROLE_INTERNAL_CLIENT
description	cx_contact
accessTokenExpirationTimeout	43200
refreshTokenExpirationTimeout	2592000
name	cx_contact
client_id	cx_contact
client_secret	<client secret token>

Code sample

```
curl --user ops:ops -X POST   http://localhost:8095/auth/v3/ops/clients/   -H 'Cache-Control:
no-cache'   -H 'Content-Type: application/json'    -d '
{"data": {
        "clientType": "CONFIDENTIAL",
        "scope": [
            "*"
        ],
        "internalClient": true,
        "authorizedGrantTypes": [
            "refresh_token",
            "implicit",
            "password",
            "client_credentials",
            "authorization_code"
        ],
      "redirectURIs": [
        "http://10.11.64.16"
      ],
        "authorities": [
            "ROLE_INTERNAL_CLIENT"
        ],
        "description": "cx_contact",
        "accessTokenExpirationTimeout": 43200,
        "refreshTokenExpirationTimeout": 2592000,
        "name": "cx_contact",
        "client_id": "cx_contact",
        "client_secret" : "<client secret token>"
    }
}'
```

Expected response

The expected response is **200 OK**.

Create external_api_client

To create the **external_api_client** for communication with GWS services, send a POST request.

Pass a body parameter in JSON format, called data, with the following properties:

Property	Value
clientType	CONFIDENTIAL
scope	*
internalClient	true
authoraizationGrantTypes	refresh_token, implicit, client_credentials, password, authorization_code
authorities	ROLE_INTERNAL_CLIENT
description	external_api_client
accessTokenExpirationTimeout	43200
refreshTokenExpirationTimeout	2592000

Property	Value
name	external_api_client"
client_id	external_api_client"
client_secret	client secret token

Code sample

```
curl   --user ops:ops  -H "Content-Type: application/json"  -X POST http://localhost:8095/
auth/v3/ops/clients  -d '
{
      "data":
      {
         "clientType": "CONFIDENTIAL",
          "scope": ["*"],
          "internalClient": true,
          "authorizedGrantTypes":  [
                  "refresh_token", "implicit", "client_credentials",
                   "password", "authorization_code"
          ],
          "authorities": ["ROLE_INTERNAL_CLIENT"],
          "description": "external_api_client",
          "accessTokenExpirationTimeout": 43200,
          "refreshTokenExpirationTimeout": 2592000,
          "name": "external_api_client",
          "client_id": "external_api_client",
          "client_secret": "client secret token"
       }
}
'
```

Expected response

The expected response is **200 OK**.

Get access token for cx_contact

To get the access token for **cx_contact**, send a POST request to GWS:

```
curl --user cx_contact:<client secret token>   -H "Content-Type: application/json"  -X POST
'http://localhost:8095/auth/v3/oauth/
token?grant_type=client_credentials>ope=*&client_id=cx_contact&client_secret=<client secret
token>'
```

This is the response:

```
{"access_token":"<bearer token>","token_type":"bearer","expires_in":43199,"scope":"*"}
```

Verify authentication (optional step)

To verify that authentication was successful, send a POST request to GWS:

```
curl --user external_api_client:secret  -H "Content-Type: application/json"  -X POST
'http://localhost:8095/auth/v3/oauth/
```

```
token?grant_type=client_credentials>ope=*&client_id=external_api_client&client_secret=<client
secret token>'
```

Verify clients

To verify that clients were created successfull, send a POST request to GWS

```
curl --user ops:ops http://localhost:8095/auth/v3/ops/clients   |   python -m json.tool
```

Create tenant in api-aggregator

To create the tenant in **api-aggregator**, send a POST request.

Pass a body parameter in JSON format, called data, with the following properties:

Parameter	Value
envrionmentid	bf032640-9073-435d-9447-718b7cc7dc43
shortTenantName	The short tenant name (for example 22-06).
customerName	The short tenant name (for example 22-06).
contactCenterId	The unique ID generated when a request is sent to GWS to create an Environment/Contact Center (for example 3952ccd2-a34a-46c1-b51e-8917628554c9).

Sample code

```
curl -X POST  -H "Authorization: Bearer  <bearer token>"  -H "Content-Type: application/
json"  -H 'Cache-Control: no-cache'  http://localhost:8102/api-aggregator/v2/tenants   -d '
{
   "data":
   {
        "domains": ["domain.com"],
        "environmentId": "bf032640-9073-435d-9447-718b7cc7dc43",
        "shortTenantName": "22-06",
        "customerName": "tenant_22-06",
        "contactCenterId": "3952ccd2-a34a-46c1-b51e-8917628554c9"
    }
}
'
```

Expected response

The expected response is **200 OK**.

Configure session profile

During CX Contact provisioning a set of objects is created in the Configuration Server.

Configure the Annex of the **DefaultSessionProfile** object (type=script), by replacing all **-1** with the following correct DBIDs:

- Voice Transfer Destination DN (origDNDBID)
- Trunk Group DBID (trunkGroupDNDBID)
- Statistics Server DBID (statServerDBID)
- Remove -1 from "serverDBIDs"

For example:

```
"data": {
  "interactionQueueDBID": 0,
  "origDNDBID": -1,
  "trunkGroupDNDBID": -1,
  "operationMode": 1,
  "statServerDBID": -1,
  "serverDBIDs": [
    -1
  ],
  "IVRProfileDBID": 0,
  "numOfChannels": 10
},
"isDefault": true
```

Add location in the CloudCluster Application

1. In the CloudCluster application, open the properties of the Connection to the OCS application.
2. In **Advanced -> Application Parameters**, enter the location using its short region name—for example: `locations=/USW1`

Log in to CX Contact

Log in to the CX Contact user interface with the URL `http://<your-docker-hostname>/ui/cxcontact/`

Important
You must include the backslash (/) after **cxcontact** (cxcontact/)

Post Deployment Procedures

After you've deployed CX Contact, you'll want to monitor the status of containers, access logs, upgrade CX Contact components, and so on. This topic describes the post deployment procedures you'll use if you deployed CX Contact using Docker Compose.

Upgrading CX Contact Components

This topic outlines the steps required to upgrade CX Contact components.

1. Ensure the CX Contact Docker images included in the upgrade are tagged with the proper version number and are available in the Docker registry.
2. Edit the **Container Versions** section of the **.env** file to specify one or more versions of the CX Contact components that you're upgrading to. For example, if you're upgrading the List Builder component of CX Contact to version 9.0.000.07.1616, specify the following: **CCSListBuilderTag=9.0.000.07.1616**
3. Execute the following command:

   ```
   ./cxc-app-deploy.sh
   ```

   ### Tip
   When upgrading, ensure that old containers are stopped and new containers are started. This means that the CX Contact solution is not available during the upgrade.

Using Portainer

Genesys recommends you use Portainer to view the status of all containers and to access logs.

1. To start Portainer on the same VM where all containers are started, execute the following command:

   ```
   docker volume create portainer_data
   ```

   ```
   docker run --restart=always -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock
   -v portainer_data:/data portainer/portainer --no-auth
   ```

2. Go to `http://<your-vm-ip>:9000`. The user interface displays the state of all containers. Open each container and click **logs** to see the `stdout / stderr` logs.

Ports

Warning

CX Contact components must use the Docker host network mode. The components must not use Docker bridge networks or swarms, which severely impact performance of the production systems.

CX Contact components use the following ports:

Service	Ports used
User Interface (UI)	8101 - Nignx listener port (static content served)
API Aggregator	• 8102 - API main • 9102 - API management
List Builder	• 3004 - API main • 3101 - API management
Compliance Manager	• 3007 - API main • 3107 - API management
List Manager	• 3005 - API main • 3105 - API management

Using Kubernetes to Deploy CX Contact

This topic and its related subtopics describe everything you need to know to deploy CX Contact using Kubernetes.

Before you Begin

- Prepare a single VM or set of VMs for the CX Contact deployment.
- Install Docker Engine CE on the VM(s) running RHEL 7.0.
- Pull CX Contact and GWS Docker images from an FTP directory and import them into an internal Docker registry. Your Genesys representative will provide you with access information to the FTP directory.
- Install Kubernetes according to the installation instructions on the Kubernetes documentation site. You can also refer to the Genesys Docker Deployment Guide for information about Kubernetes and High Availability.
- Install Helm according to the instructions outlined on the Helm documentation site.

Once you've completed these mandatory procedures, return to this manual to learn how to complete an on-premise deployment of CX Contact. Start by reviewing the Prerequisites.

Prerequisites

The table below outlines all prerequisites for a CX Contact deployment using Kubernetes.

Important

Genesys does not deploy and operate databases in on-premise deployments. It is the responsibility of the end user. In a production deployment, data store components (PostgreSQL, Redis, Elasticsearch) must be deployed outside of the Kubernetes cluster and managed by the end user's DBA team. The end user's DBA team is also responsible for ensuring that these data store components are configured with the appropriate scalability, resiliency, and data protection (backups, and so on).

Component	Description	Mandatory or Optional
CDP NG Access Credentials	As of CX Contact 9.0.025, Compliance Data Provider Next Generation (CDP NG) is used as a CDP by default. Obtain the necessary access credentials (ID and Secret) before attempting to connect to CDP NG. Request these credentials from Genesys Customer Care.	Mandatory
VMs	• Set of VMs running RHEL 7.0 64-bit • Each machine should run Red Hat Enterprise Linux 7.0 64-bit as a guest OS and have at least 8 CPU cores and 16 GB RAM minimum (32 GB RAM recommended), 100 GB HDD minimum. • When RHEL/CentOS 7.8 is used, the Kernel must be upgraded to 3.10.0-1160.15.2.el7.x86_64 or later.	Mandatory
Docker	Docker 17.03.2-ce, with CX Contact Docker images stored in the Docker registry.	Mandatory
Chrome	The latest version of Chrome must be used as the UI browser.	Mandatory
Container orchestration	Any certified K8s platform	Mandatory

Component	Description	Mandatory or Optional
Network/DNS	All VMs running CX Contact components should belong to the same local network segment and be interconnected so that all components can communicate over the network. DNS must be present in the network and allow for names resolution. CX Contact components always use FQDNs (not IP addresses) to establish communication to each other.	Mandatory
Load Balancers	F5 or functionally comparable hardware or software load balancer. The load balancer must be configured to ensure that internal CX Contact components cannot be accessed via load balancer. Only API Aggregator should be accessible.	Mandatory
Shared file system (NFS)	NFS	Mandatory
PostgreSQL	PostgreSQL 9.5+ CX contact supports non-standard Postgre SQL ports for the Data Access Point (DAP) to assist in Disaster Recovery. List Manager, List Builder, and Campaign Manager can all communicate with Postgre SQL via non-standard ports.	Mandatory
Redis	Redis 5.x cluster, Enterprise Redis with persistence is recommended	Mandatory
Elasticsearch	ES Cluster 6.3x	Mandatory
SFTP Server	Use when automation capabilities are required	Optional
Genesys Web Services (GWS)	v.9.0 **Note:** You will need to push these images to the local Docker registry.	Mandatory (Deployed using Docker Compose)
Genesys core components	v.8.5 or v.8.1 CX Contact components operate with Genesys core services on the back end. It's expected that all voice-processing components (Voice VM and shared services such as GVP) are deployed and running. CX Contact requires a multi tenant Configuration Server.	Mandatory

Recommendations

The recommendations in this topic apply only to Kubernetes.

Ingress

CX Contact UI requires Session Stickiness. Use **ingress-nginx** as the **ingress controller** (https://github.com/kubernetes/ingress-nginx).

Important

CX Contact helm chart contains default annotations for session stickiness only for **ingress-nginx**. If you are using a different ingress controller, refer to its documentation for session stickiness configuration.

Ingress SSL

Starting from Chrome 80, the **SameSite cookie** must have the **Secure** flag (https://blog.chromium.org/2020/02/samesite-cookie-changes-in-february.html). Therefore, it is highly recommended that you configure a valid SSL certificate on ingress.

Logging

Log rotation is required so that logs do not consume all of the available storage on the node.

Kubernetes is currently not responsible for rotating logs. Log rotation can be handled by the **docker json-file log driver** by setting the **max-file** and **max-size** options.

For effective troubleshooting, the engineering team should provide **stdout logs** of the pods (using the command **kubectl logs**). As a result, log retention will not be very aggressive (https://docs.docker.com/config/containers/logging/json-file/#examples). For example:

```
{
  “log-driver”: “json-file”,
  “log-opts”: {
    “max-size”: “100m”,
   “max-file”: “3”
  }
}
```

For on-site debugging purposes, CX Contact logs can be collected and stored in Elasticsearch. (For example, EFK stack https://medium.com/avmconsulting-blog/how-to-deploy-an-efk-stack-to-kubernetes-ebc1b539d063).

Monitoring

CX Contact provides metrics that can be consumed by **Prometheus** and **Grafana**. It is recommended to have the **Prometheus Operator** (https://github.com/prometheus-operator/prometheus-operator) installed in the cluster. CX Contact **helm chart** supports the creation of **CustomResourceDefinitions** that can be consumed by the **Prometheus Operator**.

Shared Filesystem

The **Kubernetes** cluster must support **ReadWriteMany Persistent Volumes**. To support **ReadWriteMany Persistent Volumes**, use the **NFS server** configured outside the cluster or via container (https://github.com/kubernetes/examples/tree/master/staging/volumes/nfs). Containers run as a **Genesys** user (uid:gid 500:500). Therefore, shared volume must have permissions that allow write access to uid:gid 500:500.

Deploying with Kubernetes

To deploy CX Contact by using Kubernetes, complete the following deployment procedures. The first three procedures are common to both deployment methods. Click the link to go to that topic:

Summary of deployment procedures
1. Ensure the Prerequisites are met
2. Review the Recommendations
3. Create the Outbound Database
4. Create the Outbound Database Access Point
5. Start Outbound Contact Server (OCS)
6. Deploy with Kubernetes

Deploy with Kubernetes

Summary of Procedures: Deploy with Kubernetes
1. Deploy CX Contact using Kubernetes and Helm charts. (CX Contact deployment with Kubernetes using shell scripts is obsolete.) • Complete the Prerequisites (for using Helm Charts) • Install CX Contact using Helm Charts • Upgrade CX Contact using Helm Charts • Configure the Helm Charts
2. Enable TLS Termination at Ingress Controller
3. Set Connectivity to the Compliance Data Provider
4. Log in to CX Contact

Deploy CX Contact using Helm Charts

Prerequisites

To begin, ensure your system contains the following prerequisite software:

- Helm 2.8+ client (without Tiller) or Helm 3
- GWS Services installed:
  - gws-core-auth
  - gws-core-environment

    - gws-platform-configuration
    - gws-platform-ocs
    - gws-platform-voice
    - gws-platform-statistics
    - gws-platform-setting
- Local Docker Repository (the location of the stored CX Contact Docker images and Helm Charts).

Install CX Contact using Helm Charts

1. Select one of the following options to obtain the CX Contact Helm chart:
    - **If you have access to the local Docker Repository:** Access the Helm charts repository and run the following two commands:
      ```
      helm repo add <repo_name> <helm_charts_repo>

      helm fetch <repo_name>/cxcontact
      ```
      As a result, the `cxcontact-<version>.tgz` archive file is added to the current working directory.
    - **If you do not have access to the local Docker Repository:** Obtain the `cxcontact-<version>.tgz` archive file and save the file in your current working directory.
2. Obtain the `yaml` default values from the following location and file:
   ```
   helm inspect values cxcontact-<version>.tgz > overrides.yaml
   ```
3. Edit **overrides.yaml** and change the default parameter values to values that match your environment. See Configure the Helm Charts table for the parameters, their description and default values.
4. Using one of the following commands, install CX Contact:
    - Helm 2: `helm template cxc cxcontact-<version>.tgz -f overrides.yaml | kubectl -n <namespace> apply -f -`
    - Helm 3: `helm -n <namespace> install cxc cxcontact-<version>.tgz -f overrides.yaml`

Upgrade CX Contact using Helm Charts

1. Select one of the following options to obtain the CX Contact Helm chart:
    - Access the Helm charts repo and run the following two commands:
      ```
      helm repo update

      helm fetch <repo_name>/cxcontact
      ```
      As a result, the **cxcontact-<new_version>.tgz** archive file is added to the current working directory.
    - From the FTP Server, obtain the **.tgz** archive file.
2. Obtain the files used for the previous deployment:
    - When working with Helmp 2, obtain the **overrides.yaml** file used for the initial deployment.

- When working with Helm 3, access `helm -n <namespace> get values cxc -o yaml > overrides.yaml` to obtain the parameters used for the initial deployment.

3. Upgrade the Helm deployment:
   - When working with Helm 2, perform the following command:
     ```
     helm template cxc cxcontact-<new_version>.tgz -f overrides.yaml | kubectl -n
     <namespace> apply -f -
     ```
   - When working with Helm 3, perform the following command:
     ```
     helm -n <namespace> upgrade cxc cxcontact-<new_version>.tgz -f overrides.yaml
     ```

Configure the Helm Charts

Parameter	Description	Default Value
image.registry	The Docker registry base-path, where CX Contact images are stored.	pureengage-docker-staging.jfrog.io/cxcontact
image.imagePullSecrets	Kubernetes imagePullSecrets	
image.pullPolicy	Kubernetes imagePullPolicy	IfNotPresent
configserver.user_name	The Configuration Server user name. This user name should be created during provisioning and stored in Users Secret.	cloudcon
configserver.user_password	The Configuration Server user password in plain text. This password should be stored in Users Secret.	
configserver.DAP_name	Database access point application. The DAP_name should be used to connect from CX Contact.	OCSDAP_usw1
configserver.OCS_name	The Outbound Contact Server application name.	OCS_usw1
configserver.tenant_dbid	The Configuration Server Tenant DBID.	1
configserver.gws_server_app_name	The server application name that is used by GWS Services.	CloudCluster
cxcontact.replicas	The number of pod replicas that should be deployed. The recommended amount is N+1.	2
cxcontact.environment	Changes the log level of errors displayed in the UI. The environment can be either "development" or "prod".	prod
cxcontact.region	The CX Contact region. Region can be used for the deployment of multiple CX Contact installations with the same GWS Services and Redis.	g0-usw0

Parameter	Description	Default Value
cxcontact.existingPGPSecretName	The name of the existing Kubernetes Secret with PGP. existingPGPSecretName should contain the following data: • cxc_pgp_private_key • cxc_pgp_public_key • passphrase • user_id	
cxcontact.existingUsersSecretName	The name of the existing Kubernetes Secret with user credentials. existingUsersSecretName should contain the following data: • gws_client_id • gws_client_secret • configserver_user • configserver_user_pass • dial_manager_dial_api_key (optional)	
cxcontact.rbac.enabled	Configures Role Based Access Control for CX Contact.	false
cxcontact.pgp.enabled	text	Configures PGP encryption.
cxcontact.pgp.passphrase	The passphrase for the private key.	
cxcontact.pgp.user_id	The user_id for the private key.	customercare@genesys.com
cxcontact.pgp.create_k8s_secret	When set to `true`, CX Contact creates a new Secret in kubernets with pgp keys. When set to `false`, CX Contact uses the Secret from existingPGPSecretName.	false
cxcontact.pgp.private_key	The contents of the PGP private key.	
cxcontact.pgp.public_key	The contents of the PGP public key.	
cxcontact.log.level	Configures the log level for all CX Contact pods. Permitted values: • trace • debug • info	info

Parameter	Description	Default Value
	• error • fatal	
cxcontact.log.log_to_file	Configures writing logs to log files located in **/mnt/log/cxc-***.	false
cxcontact.override.amark-app.replicas	Overrides the number of pod replicas for a specific micro-service.	2
cxcontact.override.amark-app.env	Extra environment variables that will be appended for the container env: definition. Env can be specified as: VAR_NAME: VAR_VAL	{}
cxcontact.override.amark-app.resources	Overrides the resources for a specific micro-service.	{}
override.amark-app.readinessProbe	Enables/Disables readinessProbe	true
cxcontact.override.amark-app.livenessProbe	livenessProbe	true
cxcontact.override.job-scheduler.replicas	Overrides the number of pod replicas for a specific micro-service.	2
cxcontact.override.job-scheduler.env	Extra environment variables that will be appended for the container env: definition. Env can be specified as: VAR_NAME: VAR_VAL	{}
cxcontact.override.job-scheduler.resources	Overrides the resources for a specific micro-service.	{}
cxcontact.override.job-scheduler.readinessProbe	Enables/Disables readinessProbe	true
cxcontact.override.job-scheduler.livenessProbe	livenessProbe	true
cxcontact.override.campaign-manager.replicas	Overrides the number of pod replicas for a specific micro-service.	2
cxcontact.override.campaign-manager.env	Extra environment variables that will be appended for the container env: definition. Env can be specified as: VAR_NAME: VAR_VAL	{}
cxcontact.override.campaign-manager.resources	Overrides the resources for a specific micro-service.	{}
cxcontact.override.campaign-manager.readinessProbe	Enables/Disables readinessProbe	true
cxcontact.override.campaign-	livenessProbe	true

Parameter	Description	Default Value
manager.livenessProbe		
cxcontact.override.list-manager.replicas	Overrides the number of pod replicas for a specific micro-service.	2
cxcontact.override.list-manager.env	Extra environment variables that will be appended for the container env: definition. Env can be specified as: VAR_NAME: VAR_VAL	{}
cxcontact.override.list-manager.resources	Overrides the resources for a specific micro-service.	{}
cxcontact.override.list-manager.readinessProbe	Enables/Disables readinessProbe	true
cxcontact.override.list-manager.livenessProbe	livenessProbe	true
cxcontact.override.complaince.replicas	Overrides the number of pod replicas for a specific micro-service.	2
cxcontact.override.complaince.env	Extra environment variables that will be appended for the container env: definition. Env can be specified as: VAR_NAME: VAR_VAL	{}
cxcontact. override.complaince.resources	Overrides the resources for a specific micro-service.	{}
cxcontact.override.complaince.readinessProbe	Enables/Disables readinessProbe	true
cxcontact.override.complaince.livenessProbe	livenessProbe	true
cxcontact.override.amark-ui.replicas	Overrides the number of pod replicas for a specific micro-service.	2
cxcontact.override.amark-ui.env	Extra environment variables that will be appended for the container env: definition. Env can be specified as: VAR_NAME: VAR_VAL	{}
cxcontact.override.amark-ui.resources	Overrides the resources for a specific micro-service.	{}
cxcontact.override.amark-ui.readinessProbe	Enables/Disables readinessProbe	true
cxcontact.override.amark-ui.livenessProbe	livenessProbe	true
cxcontact.override.list builder.replicas	Overrides the number of pod replicas for a specific micro-service.	2
cxcontact.override.list builder.env	Extra environment variables that will be appended for the container env: definition. Env can	{}

Parameter	Description	Default Value
	be specified as: VAR_NAME: VAR_VAL	
cxcontact.override.list builder.resources	Overrides the resources for a specific micro-service.	{}
cxcontact.override.list builder.readinessProbe	Enables/Disables readinessProbe	true
cxcontact.override.list builder.livenessProbe	livenessProbe	true
cxcontact.override.dial-manager.enabled	Enables/Disables Dial Manager service deployment.	false
cxcontact.override.dial-manager.nexus.host	Configures the Nexus service host.	
cxcontact.override.dial-manager.nexus.port	Configures the Nexus service port.	
cxcontact.override.dial-manager.api_key	The API key used to access Nexus. The api_key should be in plain text and will be stored in Users Secret.	
cxcontact.compliance_data.cdp_url	When configured cdp_url overrides the compliance data provider URL.	false
cxcontact.compliance_data.proxy	Configures the proxy connection to CDP. Disabled if false.	false
cxcontact.compliance_data.list_builder_test_files_mode	Configures List Builder Compliance Data Mode for debug purposes only.	false
cxcontact.compliance_data.list_builder_test_dnc_mode	Configures List Builder Compliance Data Mode for debug purposes only.	false
cxcontact.initContainers	Enables the configuration of extra initContainers for CX Contact pods.	[ ]
cxcontact.deployDefaultInitContainer	Allows you to disable the default initContainer if you mount Storage with uid:guid – 500:500.	true
k8s_optional.podSecurityContext	Enables you to set the securityContext for the pod.	{}
k8s_optional.securityContext	Enables you to set the securityContext for the container.	{}
k8s_optional.nodeSelector	Enables you to configure nodeSeclector to target specific nodes.	{}
k8s_optional.tolerations	Enables you to configure tolerations.	[ ]
k8s_optional.affinity	Enables you to configure	[ ]

Parameter	Description	Default Value
	affinity.	

k8s_optional.strategy	Enables you to configure strategy.	type: RollingUpdate
		`rollingUpdate:`
		`  maxSurge: 1`
		`  maxUnavailable: 25%`

redis.enabled	Enables/Disables the Reddis connection.	true
redis.cluster	Enables you to configure Redis.	true
redis.nodes	The Redis node URL.	redis://redis-cluster:6379
elasticsearch.enable	Enables/Disables the Elasticsearch Cluster connection.	true
elasticsearch.host	Elasticsearch host	http://elasticsearch
elasticsearch.port	Elasticsearch port	9200
gws.client_id	The client_id is created by the CX Contact provisioning service and is stored in the Users Secret.	cx_contact
gws.client_secret	The client_secret is created by the CX Contact provisioning service and is stored in the Users Secret.	
gws.frontend_host	Represents the GWS front end http/https URL. frontend_host is used for browser user authentication.	http://active.gke.local
gws.frontend_port	The GWS front end port.	80
loadbalander.host	GWS backend Load balacer host (optional).	
loadbalander.port	GWS backend Load balacer host (optional).	
loadbalander.core.auth.host	GWS Core Auth host	http://gws-core-auth-srv
loadbalander.core.auth.port	GWS Core Auth port	80
loadbalander.core.environment.host	GWS Core Environment host	http://gws-core-environment-srv
loadbalander.core.environment.port	GWS Core Environment port	80
loadbalander.platform.ocs.host	GWS Platform OCS host	http://gws-platform-configuration -srv
loadbalander.platform.ocs.port	GWS Platform OCS port	80
loadbalander.platform.configuration.host	GWS Platform Configuration host	http://gws-platform-configuration-srv
loadbalander.platform.configuration.port	GWS Platform Configuration port	80
loadbalander.platform.statistics.host	GWS Platform Statistics host	http://gws-platform-statistics -srv
loadbalander.platform.statistics.port	GWS Platform Statistics port	80

Parameter	Description	Default Value
loadbalander.platform.setting.host	GWS Platform Setting host	http://gws-platform-setting-srv
loadbalander.platform.setting.port	GWS Platform Setting port	80
loadbalander.platform.voice.host	GWS Platform Voice host	http://gws-platform-voice-srv
loadbalander.platform.voice.port	GWS Platform Voice port	80
ingress.enabled	Enables/Disables the deployment of the built-in ingress resource.	true
ingress.tls_enabled	HTTPS	false
ingress.cxc_frontend	The host used by ingress for all inbound traffic.	cxcontact.gke.local
ingress.annotations	The ingress resource annotations.	• nginx.ingress.kubernetes.io/affinity: cookie • nginx.ingress.kubernetes.io/session-cookie-samesite: "Strict" • nginx.ingress.kubernetes.io/session-cookie-name: "cxc-session-cookie" • nginx.ingress.kubernetes.io/proxy-body-size: "0"
ingress.tls	TLS configuration. When enabled TLS is True.	[ ]
internal_ingress.enabled	Enables/Disables the deployment of the built-in ingress resource for back-end services. When false, all endpoints are exposed on ingress with cxc_frontend.	false
internal_ingress.tls_enabled	HTTPS	false
internal_ingress.cxc_backend	The host used by ingress for all inbound traffic.	cxcontact-int.gke.local
internal_ingress.annotations	The ingress resource annotations.	• nginx.ingress.kubernetes.io/proxy-body-size: "0" • nginx.ingress.kubernetes.io/ssl-redirect: 'false'
internal_ingress.tls	TLS configuration. When enabled TLS is True.	[ ]
storage.pvc.enabled	Enables/Disables storage mounts.	true
storage.pvc.create	Enable pvc deployment.	true
storage.pvc.size	The size of the deployed pvc.	100Gi
storage.pvc.name	The name of the deployed pvc.	cxc-claim

Parameter	Description	Default Value
storage.pvc.storageClassName	The storageClass name that should be used when creating pvc. If storageClassName is empty it will not be used. storageClassName should be assigned accessModes: ReadWriteMany.	files-standard-zrs
storage.pv.create	Enables the creation of pv.	false
storage.pv.name	The pv name that should be created and used by pvc.	cxc-volume
storage.pv.spec	PV specification.	`capacity:` `storage: 100Gi` `accessModes:` `- ReadWriteMany` `persistentVolumeReclaimPolicy: Retain` `nfs:` `path: /data` `server: 10.128.0.42`
amark-app	docker image tag	Dependent on the CX Contact release.
job-scheduler	docker image tag	Dependent on the CX Contact release.
campaign-manager	docker image tag	Dependent on the CX Contact release.
list-manager	docker image tag	Dependent on the CX Contact release.
compliance	docker image tag	Dependent on the CX Contact release.
amark-ui	docker image tag	Dependent on the CX Contact release.
list-builder	docker image tag	Dependent on the CX Contact release.
dial-manager	docker image tag	Dependent on the CX Contact release.

Enable TLS Termination at Ingress Controller

1. Prepare the k8s secret with the SSL Certificate using the following code: `kubectl create secret cxc-tls ${CERT_NAME} --key ${KEY_FILE} --cert ${CERT_FILE}`
   Note: Skip this step if the kubernetes cluster has a cert-manager installed.

2. Update **overrides.yaml** that is used for the CX Contact installation as follows:

```
ingress:
  enabled: true
  tls_enabled: true
```

```
cxc_frontend: <fqdn>
# if kubernetes cluster has a cert-manager installed:
annotations:
        cert-manager.io/cluster-issuer: <name of cert-manager>
tls:
  - hosts:
     - <fqdn>
     secretName: cxc-tls
```

Note: The same configuration can be applied to **internal_ingress**. If the configuration is applied to **internal_ingress**, you must add the CX Contact FQDN and a certificate of the host where Configuration Server runs.

3. Prepare the k8s secret with the SSL Certificate as follows: `kubectl create secret cxc-int-tls ${CERT_NAME} --key ${KEY_FILE} --cert ${CERT_FILE}`

```
internal_ingress:
  enabled: true
  cxc_backend: <int_fqdn>
 # if kubernetes cluster has a cert-manager installed:
  annotations:
          cert-manager.io/cluster-issuer: <name of cert-manager>
  tls:
    - hosts:
       - <int_fqdn>
       secretName: cxc-int-tls
```

4. Apply the following new configuration:
   `helm -n <namespace> upgrade cxc cxc -f overrides.yaml`

5. Whitelist a new **<fqdn>** on the **auth service** using one of the following methods:

   - Manually via the REST API:

```
          curl -u <GWS_BASIC_AUTH_USER>:<GWS_BASIC_AUTH_PASSWORD> -L -X PUT
'<GWS_LB_HOST>/auth/v3/ops/clients/<GWS_CLIENT_ID>' \
                 -H 'Content-Type: application/json' \
                 -d '{
                     "data": {
                         "redirectURIs": [
                             "https://<fqdn>/cx-contact/v3/login-callback",
                             "http://<fqdn>/cx-contact/v3/login-callback"
                         ]
                     }
                 }'
```

   - Using the **cxcontact provisioning service** (cxc-app.sh), update **CXC_EXTERNAL_URL** in the **.env** file and execute: `./cxc-app.sh provision`

Set Connectivity to the Compliance Data Provider

As of CX Contact 9.0.025.xx, CDP NG is used by default. The following Helm Chart settings control the CDP NG connectivity:

```
cxcontact:
  compliance_data:
    cdp_ng:
      url: "https://api.usw2.pure.cloud/api/v2/outbound/compliancedata"
      gcloud_auth: "https://login.usw2.pure.cloud/oauth/token"
      gcloud_id:
```

```
      gcloud_secret:
    # LIST_BUILDER_DATA_EMBEDDED_BASEPATH
    embedded_basepath: "/list_builder/data/ng_init_data"
    rule_set:
      areacode: "AU,CA,GB,NZ,US"
      geo: "AU,CA,GB,NZ,US"
      postal: "CA,GB,US"
      dnc: "GB,US"
```

Important

The **gcloud_id** and **gcloud_secret** parameters are required and do not have default values.

The following parameters can be used to switch to legacy CDP:

```
cxcontact:
  compliance_data:
    cdp_ng:
      url: false
      gcloud_auth: false
      gcloud_id: false
      gcloud_secret: false
    # LIST_BUILDER_DATA_EMBEDDED_BASEPATH
    embedded_basepath: "/list_builder/data/init_data"
```

Log in to CX Contact

Log in to the CX Contact user interface with the URL `http://<your-docker-hostname>/ui/cxcontact/`

Important

You must include the backslash (/) after **cxcontact** (cxcontact/)

Common Deployment Procedures

This topic contains the deployment procedures that are common to both CX Contact deployment methods (Docker Composer and Kubernetes).

Creating the Outbound Database

1. To store calling and suppression lists, create the Outbound Database manually on the PostgreSQL server that will be used with CX Contact.
2. Log in to PostgreSQL as the administrator and execute the following set of SQL statements to create the Outbound Database and a user.

> **Important**
>
> - Take note of the database name, username, and password because you will need them when you create the Outbound Database Access Point.
> - CX Contact functionality depends on the following database settings. The calling and suppression lists will not be stored correctly if these SQL statements are not executed as documented in this section.

```
CREATE DATABASE cc_outbound;
 ALTER DATABASE cc_outbound SET bytea_output TO 'escape';
 ALTER DATABASE cc_outbound SET standard_conforming_strings TO 'off';
 CREATE USER cc_outbound WITH PASSWORD 'cc_outbound';
 GRANT ALL PRIVILEGES ON DATABASE cc_outbound TO cc_outbound;
 ALTER DATABASE cc_outbound OWNER TO cc_outbound;
```

3. Switch to the new database **cc_outbound** as the administrative user and issue the following command:

```
CREATE EXTENSION tablefunc;
```

Creating the Outbound Database Access Point

Log in to the configuration environment and create a **Database Access Point (DAP)** object that points to the newly-created database.

The DAP must reference the DB Server so that OCS can work with the database. We recommend you name the DAP object **OCSDAP**.

Use the database connection information, database name, username, and password from the previous step.

Starting Outbound Contact Server

Start the Outbound Contact Server (OCS) application with the Management Layer or GAX, as you usually do in your environment.

Optional Functionality

In some deployments optional functionality is required for applications, scripts, services, and so on. Genesys recommends the following optional functionality for some familiar deployment issues:

- PGP Encrytion
- Integrating CX Contact with Genesys Historical Reporting

PGP Encryption

In a Kubernetes deployment encryption is disabled by default.

Important
PGP Encryption is supported only in Kubernetes deployments.

Enable PGP encryption in Kubernetes deployments

1. Generate a pair of PGP keys to be used for encryption/decryption (private and public keys).
2. Store each generated key in the file on the host, so that these files are accessible by the deployment script.
3. Configure the following environment variables in the **cxc.env** file.

```
# CXC Contact encryption configuration.
 CXC_PGP_ENABLED: false
 # Host path(absolute) to the PGP Public Key
 CXC_PGP_PUBLIC_KEY_PATH: ""
 # Host path(absolute) to the PGP Private Key
 CXC_PGP_PRIVATE_KEY_PATH: ""
 # Passphrase for PGP Private Key
 CXC_PGP_PASSPHRASE: ""
 CXC_PGP_USER_ID: "customercare@genesys.com"
```

4. Verify that the **CXC_PGP_ENABLED** variable is set to **true**.
5. Verify that the **CXC_PGP_PUBLIC_KEY_PATH** variable is set to the absolute path to the file that stores the public key.
6. Verify that the **CXC_PGP_PRIVATE_KEY_PATH** variable is set to the absolute path to the file that stores the private key.
7. Verify that the **CXC_PGP_PASSPHRASE** variable (optional) is configured when the passphrase is present in CX Contact PGP keys.
8. Verify that the **CXC_PGP_USER_ID** variable is associated with the correct Private key user ID.
9. Save and close the file. The saved file is then used as input for the **cxc-app-deploy.sh** script.

Important
The host is used to create a Kubernetes secret (cxc-pgp-storage). During deployment, CXC_PGP_PUBLIC_KEY_PATH and CXC_PGP_PRIVATE_KEY_PATH data is stored in the Kubernetes secure storage. When the system is started, CX Contact components

collect key data from the Kubernetes secure storage. For more information about Kubernetes secrets, see Kubernetes Documentation.

Integrating CX Contact with Genesys Historical Reporting

This page describes the component and configuration requirements to enable historical reporting on unattempted records.

Overview: Historical Reporting on Unattempted Records

While Outbound Contact Server (OCS) reports on the outcome of all attempted records and records failed pre-dial validation checks, it does not report on records belonging to a contact suppression list for the campaign group. This comes from CX Contact. The process is as follows:

1. When the campaign group is activated, CX Contact writes all information related to unattempted records to an Elasticsearch index (it writes one Elasticsearch document for each suppressed record).
2. As part of the regular ETL cycle, Genesys Info Mart extracts the data from Elasticsearch and transforms it into Genesys Info Mart **LDR_*** tables, which you can join with OCS-sourced data on that campaign group's attempted records.

For more information about the Genesys Info Mart database tables, see the *Genesys Info Mart Physical Data Model* for your RDBMS. For more information about managing the Genesys Info Mart ETL jobs, see the *Genesys Info Mart Operations Guide*.

Defining Unattempted Records

In this context, an unattempted record refers to a record belonging to a contact suppression list. Records excluded from a campaign because of defined filtering criteria or compliance rules are not considered unattempted records in this context.

The following table summarizes the ways in which records are reported on:

Record Type	Reporting Source
Dialed/attempted records	OCS > ICON > Genesys Info Mart
Records belonging to a contact suppression list (unattempted records)	CX Contact > Elasticsearch > Genesys Info Mart
Records that failed pre-dial validation checks (unattempted records)	OCS > ICON > Genesys Info Mart

Enabling Historical Reporting on Unattempted Records

Prerequisites

The following table summarizes the minimum release requirements for the Genesys and third-party components that enable CX Contact historical reporting.

Component	Minimum release
CX Contact	9.0.000.09
Elasticsearch	6.3.1
Genesys Info Mart	8.5.012.15
ICON	8.1.514.11 (Recommended minimum for Genesys Info Mart; Required for OCS historical reporting)

Setting up Historical Reporting

To set up historical reporting of unattempted records:

1. Deploy Elasticsearch version 6.3.1. Once this is successfully deployed, CX Contact can write all required indexes to Elasticsearch. No explicit CX Contact configuration is required.

   Important

   There are index properties that contain personally identifiable information (PII) and therefore need to be considered for the EU General Data Protection Regulation (GDPR). Ensure you configure the Elasticsearch data-retention settings so that indexes are purged before 30 days.

2. Configure Genesys Info Mart to extract the CX Contact reporting data from Elasticsearch, as follows:
   1. On the **Options** tab of the Genesys Info Mart application object, create a new configuration section, called **elasticsearch-ldr0**.
   2. Add the client option. For example: **elasticsearch-ldr0/client=rest(host.domain.com)**
   3. Add the g:tenant-prefix option. For example: **elasticsearch-ldr0/g:tenant-prefix=-2115**

   Important

   Genesys expects that CX Contact reporting on unattempted records will be used to supplement existing Outbound Contact reporting sourced from OCS. Ensure that your deployment has been configured as required for Genesys Info Mart to support Outbound Contact reporting. For more information, see Enabling Reporting on Outbound Contact Activity in the *Genesys Info Mart Deployment Guide*.

Elasticsearch Index Properties

The following table describes the Elasticsearch index properties, in which CX Contact stores the data about unattempted records. Note the following:

- The **Index property** column represents the XPath term Genesys Info Mart uses to extract and map the data.
- The **Info Mart Database Target** column indicates the Info Mart database table and column to which the property is mapped.

Index property	Description	Info Mart Database Target
campaignGroupId	The DBID of the campaign group as assigned by Configuration Server.	LDR_CAMPAIGN.CAMPAIGN_GROUP_ID (referenced through LDR_FACT.LDR_CAMPAIGN_KEY)
campaignGroupName	The name of the campaign group.	LDR_CAMPAIGN.CAMPAIGN_GROUP_NAME (referenced through LDR_FACT.LDR_CAMPAIGN_KEY)
campaignTemplateName	The name of the campaign template on which the campaign group is based.	LDR_CAMPAIGN.CAMPAIGN_TEMPLATE_NAME (referenced through LDR_FACT.LDR_CAMPAIGN_KEY)
chainId	The chain identifier of the record from the contact list.	LDR_FACT.CHAIN_ID
chainN	The order of the contact list record within the chain.	LDR_FACT.CHAIN_NUMBER
clientId	The unique client identifier of the contact from the contact list.	LDR_FACT.CLIENT_ID
contact_info	The contact information (device) for the contact from the contact list.	LDR_FACT.CONTACT_INFO
contact_info_type	The type of the contact device. This field is set to one of the following values: **Valid values:** • No Contact Type • Home Phone • Direct Business Phone • Business With Extension • Mobile • Vacation Phone • Pager • Modem	LDR_RECORD.CONTACT_INFO_TYPE (referenced through LDR_FACT.LDR_RECORD_KEY)
Index property	**Description**	**Info Mart Database Target**

Index property	Description	Info Mart Database Target
	• Voice Mail • Pin Pager • E-Mail Address • Instant Messaging	
deviceAreaCode	The area code of the record from the contact list.	LDR_DEVICE.DEVICE_AREA_CODE (referenced through LDR_FACT.LDR_DEVICE_KEY)
deviceCountryCode	The country code of the record from the contact list.	LDR_DEVICE.DEVICE_COUNTRY_CODE (referenced through LDR_FACT.LDR_DEVICE_KEY)
deviceMask	The bit mask of the record from the contact list.	LDR_FACT.DEVICE_MASK
deviceStateCode	The state code (or country code) of the record from the contact list.	LDR_DEVICE.DEVICE_STATE_CODE (referenced through LDR_FACT.LDR_DEVICE_KEY)
deviceTimezone	The time zone indicated in the record from the contact list.	LDR_DEVICE.DEVICE_TIMEZONE (referenced through LDR_FACT.LDR_DEVICE_KEY)
disposition	The reason for filtering out the record from the campaign during the pre-loading phase, as reported by CX Contact.	LDR_RECORD.DISPOSITION (referenced through LDR_FACT.LDR_RECORD_KEY)
groupName	The name of the agent group or place group.	LDR_GROUP.GROUP_NAME (referenced through LDR_FACT.LDR_GROUP_KEY)
id	An identifier Genesys Info Mart generates based on the long UUID timestamp reported by CX Contact.	LDR_FACT.ID
listId	DBID of the contact list as assigned by Configuration Server.	LDR_LIST.LIST_ID (referenced through LDR_FACT.LDR_LIST_KEY)
listName	The name of the contact list.	LDR_LIST.LIST_NAME (referenced through LDR_FACT.LDR_LIST_KEY)
postalCode	The postal code of the record from the contact list.	LDR_POSTAL_CODE.POSTAL_CODE (referenced through LDR_FACT.LDR_POSTAL_CODE_KEY)
recordId	The identifier of the record from the contact list.	LDR_FACT.RECORD_ID
recordStatus	The status of the record from the contact list. This field is set to one of the following values: **Valid values:**	LDR_RECORD.RECORD_STATUS (referenced through LDR_FACT.LDR_RECORD_KEY)
Index property	**Description**	**Info Mart Database Target**

Index property	Description	Info Mart Database Target
	• No Record Status • Ready • Retrieved • Updated • Stale • Cancelled • Agent Error • Chain Updated • Missed Callback • Chain Ready	
recordType	The type of the record from the contact list. This field is set to one of the following values: **Valid values:** • No Record Status • Ready • Retrieved • Updated • Stale • Cancelled • Agent Error • Chain Updated • Missed Callback • Chain Ready	LDR_RECORD.RECORD_TYPE (referenced through LDR_FACT.LDR_RECORD_KEY)
timestamp_iso8601	The timestamp when the event regarding the suppressed contact list records was generated by CX Contact.	LDR_FACT.START_DATE_TIME_KEY
Index property	**Description**	**Info Mart Database Target**

Elasticsearch Index Fields

The following seven sections describe the seven types of Elasticsearch index fields. Each record represents the Elasticsearch data shown in the corresponding CX Contact Analytics Reporting panel.

	Job Record
	Call List Loading Record
	Preloading Record
	Campaign Group Event Record
	Call Result Record
	Contact History Record
	SMS/EMAIL Record
	User Actions Record

Job Record (cxc-job-*)

Field	Type
id	keyword
parentid	keyword
@timestamp	date
@endtime	date
ccid	keyword
type	keyword
name	keyword
state	keyword
result	keyword
created	date
started	date
finished	date
duration	integer
error	text
errorCode	integer
trace	keyword
component	keyword

Field	Type
version	keyword
hostname	keyword
address	keyword

Call List Loading Record (cxc-didr-*)

Field	Type
id	keyword
@timestamp	date
type	keyword
jobid	keyword
jobts	date
importfile	keyword
line	integer
mappingfile	keyword
ccid	keyword
listid	integer
listTableName	keyword
listName	keyword
customTZMap	boolean
chain_id	integer
chain_n	integer
contact_info	keyword
deviceDigits	text
defaultRegion	keyword
deviceIndex	short
accepted	byte
error	keyword
e164	keyword
countryCode	keyword
areaCode	keyword
exchange	keyword
restOfNumber	keyword
maskValue	long
tzuid	integer
state_code	keyword
country_code_iso	keyword

Field	Type
mask	object

Preloading Record (cxc-contact-*)

Field	Type
id	keyword
@timestamp	date
ccid	keyword
calluuid	keyword
contact_info	keyword
contact_info_type	keyword
contact_id	keyword
chain_id	integer
chain_n	integer
callTime	date
callResult	keyword
dialingMode	keyword
optimizationGoal	integer
optimizationMethod	keyword
listName	keyword
listid	integer
campaignName	keyword
campaignGroupName	keyword
sessionuuid	keyword
campaignTemplateName	keyword
groupName	keyword
agentLoginId	keyword
disposition	keyword
successful	boolean
userData	object

Campaign Group Event Record (cxc-cgevent-*)

Field	Type
id	keyword
@timestamp	date
ccid	keyword

Field	Type
sessionuuid	keyword
action	keyword
state	keyword
dialingMode	keyword
optimizationParameter	integer
optimizationType	keyword
campaignName	keyword
campaignGroupName	keyword
campaignGroupDBID	keyword
campaignTemplateName	keyword
groupName	keyword
actualBusyFactor	float
actualHitRatio	float
actualOverdialRate	Float
actualTimeToComplete	integer
lists	object

Call Result Record (cxc-crr-*)

Field	Type
id	keyword
@timestamp	date
@endtime	date
ccid	keyword
calluuid	keyword
contact_info	keyword
contact_info_type	keyword
blockingRuleName	keyword
duration	integer
durationCall	integer
durationACW	integer
durationCPD	integer
durationQueue	integer
timeDialing	date
timeClientRinging	date
timeBadCallReleased	date
timeClientPickedUp	date

Field	Type
timeCPDFinished	date
timeQueued	date
timeAgentRinging	date
timeAgentEstablished	date
timeAMDiverted	date
timeAbandoned	date
timeAgentCallReleased	date
callTime	date
callResult	keyword
dialingMode	keyword
optimizationGoal	integer
optimizationMethod	keyword
listName	keyword
campaignName	keyword
campaignGroupName	keyword
sessionuuid	keyword
campaignTemplateName	keyword
groupName	keyword
timezoneName	keyword
timezoneNameCME	keyword
timezoneOffset	integer
agentLoginId	keyword
scheduledTime	date
recordType	keyword
recordStatus	keyword
voiceTransferDestination	keyword
countryCode	keyword
clientCountryCode	keyword
areaCode	keyword
deviceTimezone	keyword
disposition	keyword
postalCode	keyword
userData	object

Contact History Record (cxc-ldr-*)

Field	Type
id	keyword
@timestamp	date
ccid	keyword
campaignName	keyword
campaignId	integer
campaignGroupName	keyword
campaignGroupId	integer
campaignTemplateName	keyword
campaignTemplateId	integer
groupName	keyword
groupId	integer
blockingRuleName	keyword
blockingRuleId	integer
listName	keyword
listId	integer
recordId	integer
clientId	keyword
chainId	integer
chainN	integer
contact_info	keyword
contact_info_type	keyword
recordType	keyword
recordStatus	keyword
deviceCountryCode	keyword
deviceAreaCode	keyword
deviceStateCode	keyword
deviceTimezone	keyword
deviceMask	integer
postalCode	keyword
disposition	keyword
reason	keyword
customFields	object
timestamp_iso8601	date

SMS/EMAIL Record (cxc-nexdr-*)

Field	Type
id	keyword
@timestamp	date
ccid	keyword
mediaType	keyword
calluuid	keyword
contact_info	keyword
clientId	keyword
chainId	integer
chainN	integer
from	keyword
subject	keyword
listName	keyword
campaignName	keyword
groupName	keyword
campaignGroupName	keyword
campaignTemplateName	keyword
sessionuuid	keyword
messageID	keyword
batchID	keyword
status	keyword
deliveryReceipt	keyword
disposition	keyword
callResult	keyword
errorCode	integer
errorMessage	keyword
timeReceivedFromOCS	date
timeSubmittedToNexus	date
timeResponseReceived	date
timeOCSNotified	date
timeConsumerResponded	date
optout	boolean
userData	object

User Actions Record (cxc-audit-*)

Field	Type
id	keyword
requestID	keyword
@timestamp	date
userName	keyword
@endtime	date
duration	integer
action	Keyword
actionDetails	keyword
objectType	keyword
objectSubtype	keyword
objectName	keyword
objectID	integer
apicall	boolean
successful	boolean
errorMessage	text
details	text
endPoint	text
changeSet	object

Elasticsearch Maintenance Recommendations

To help you better manage your indexes and snapshots and to prevent too many indexes from creating an overflow of shards, it is recommended that you set up a scheduled execution of Elasticsearch Curator with the following two actions.

1. Delete indexes older than 60 days according to the index name and mask.
    - cxc-job-*
    - cxc-audit-*
    - cxc-crr-*
    - cxc-didr-*
    - cxc-ldr-*
    - cxc-nexdr-*
    - cxc-cgevent-*
    - cxc-contact-*

2. Make a snapshot of each index.
    - cxc-analytics-*

Important

cxc-analytics-* indexes do not have a timestamp in their name and must not be deleted. Deleting a cxc-analytics-* index will result in the loss of all CX Contact Analytics Dashboard customizations.