Genesys Universal Messaging Administrator Guide

Gildas Cherruel (gildas.cherruel@genesys.com)

v. 1.0.21, 2023/11/09

Contents

Chapter 1

Concepts

Genesys Universal Messaging is a collection of micro-services connected with each other via message queueing and managed by Kubernetes.

Each micro-service is responsible to connect to one service that will provide either a Social Media, a Customer Center Platform, or a Storage. Some micro-services will, on the other hand, provide a service of their own, like the API, the config website.

Here is a non-exhaustive list of these micro-services:

- API, provides the REST API to configure and use Universal Messaging;

- Config, provides a website to configure Universal Messaging;

- Commander, provides commands to the agent (see later);

- Apple Messages for Business Connector, interfaces with Apple Messages for Business

- Bedore Connector, interfaces with the Bedore Bot;

- BizM for KakaoTalk, interfaces with the KakaoTalk Social Media via BizM;

- Disqus Connector, interfaces with Disqus Comment Service;

- Google Business Messages Connector, interfaces with Google Business Messages;

- Google Chat Connector, interfaces with Google Chat;
- Infobank for KakaoTalk, interfaces with the KakaoTalk Social Media via Infobank;

- LINE Connector, interfaces with the LINE Social Media;

- Media4U Connector, interfaces with the Media4U SMS Services;

- Microsoft Teams Connector, interfaces with Microsoft Teams;
- Slack Connector, interfaces with the Slack Social Media;
- PlusMessage Connector, interfaces with DOCOMO +Message Social Media;

- Telegram Connector, interfaces with the Telegram Social Media;

- Viber Connector, interfaces with the Viber Social Media;

- WeChat Connector, interfaces with the WeChat Social Media;

- Zalo Connector, interfaces with the Zalo Social Media;

- Genesys Cloud CX, the Genesys Cloud CX platform;

- PureConnect, the Genesys PureConnect CX platform;

- AWS Provider, Amazon Web Services S3 Storage;

- Azure Provider, Microsoft Azure Blob Storage;

- Box Provider, Box.com;

- Dropbox Provider, Dropbox;

- Google Provider, Google Cloud Storage;

All these services are connected together via the RabbitMQ Message Queueing System. It is used to exchange messages from a Social Media and a CX platform.

The following diagram describes all micro-services in the context of their relation with the Social Media, CX platforms, and Storage providers.
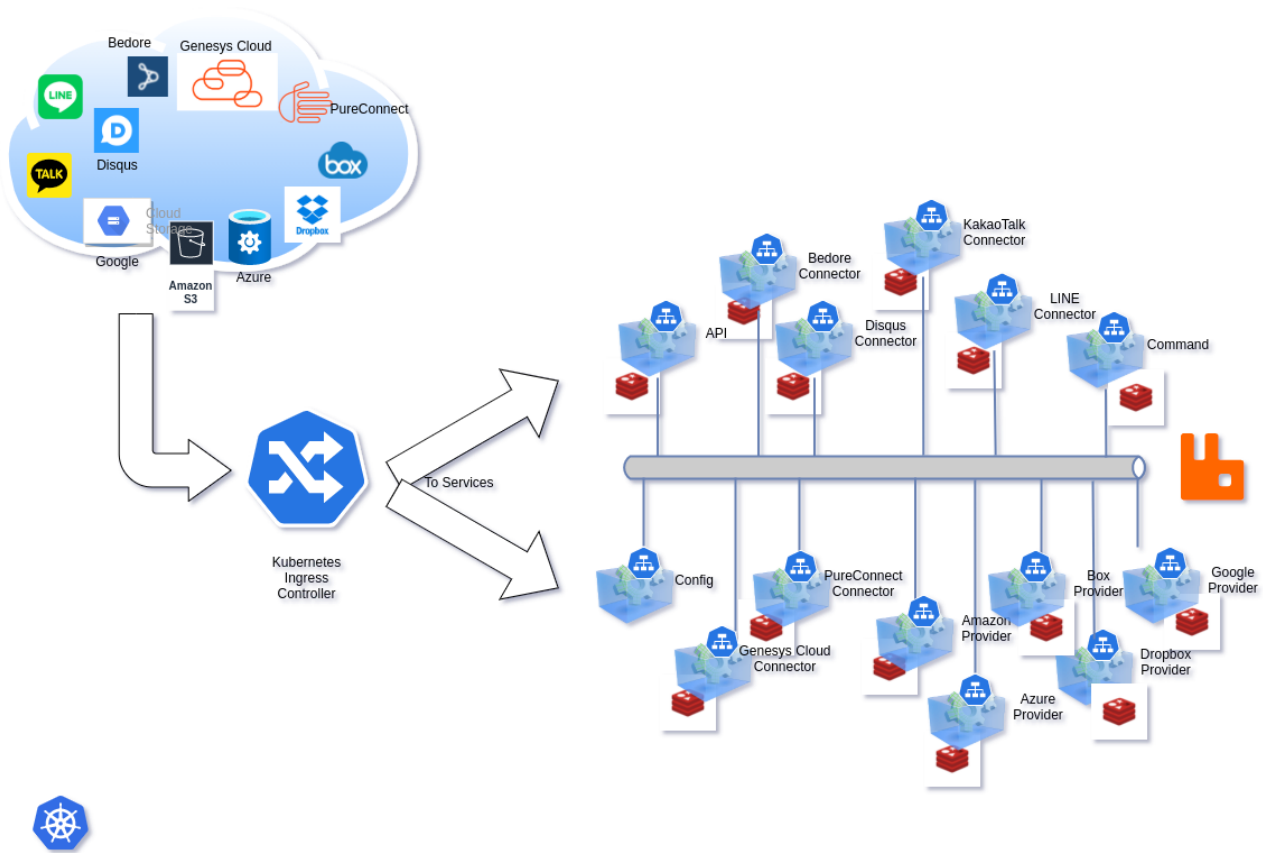
Each micro-service is a collection of Kubernetes Deployments, Services, like:

- a ClusterIP Kubernetes service,

- a Deployment,

- a REDIS (service + Stateful)

Only the Config micro-service does not carry a REDIS.

The micro-services access directly the Social Media, CX Platform, or Storage Provider they support. It is possible to make them access through a Proxy by using the config website.

A Kubernetes Ingress controller is needed to send the traffic to the respective micro-services. The Ingress Object looks like (not all properties are show):

```
apiVersion: networking.k8s.io/v1
kind: Ingress
spec:
```

```yaml
  rules:
  - http:
      paths:
      - path: /api
        backend:
          service:
            name: XX-api
            port: { number: 80 }
      - backend:
          service:
            name: XX-amb-connector
            port: { number: 80 }
        path: /amb
      - backend:
          service:
            name: XX-bedore-connector
            port: { number: 80 }
        path: /bedore
      - backend:
          service:
            name: XX-bizmsg-connector
            port: { number: 80 }
        path: /bizmsg
      - backend:
          service:
            name: XX-disqus-connector
            port: { number: 80 }
        path: /disqus
      - backend:
          service:
            name: XX-gbm-connector
            port: { number: 80 }
        path: /gbm
      - backend:
          service:
            name: XX-gchat-connector
            port: { number: 80 }
        path: /gchat
      - backend:
          service:
            name: XX-infobank-connector
            port: { number: 80 }
        path: /infobank
      - backend:
          service:
            name: XX-line-connector
            port: { number: 80 }
        path: /line
      - backend:
          service:
            name: XX-media4u-connector
            port: { number: 80 }
        path: /media4u
      - backend:
          service:
            name: XX-plusmessage-connector
            port: { number: 80 }
        path: /plusmessage
      - backend:
          service:
```

```yaml
          name: XX-slack-connector
          port: { number: 80 }
      path: /slack
  - backend:
      service:
        name: XX-teams-connector
        port: { number: 80 }
      path: /teams
  - backend:
      service:
        name: XX-telegram-connector
        port: { number: 80 }
      path: /telegram
  - backend:
      service:
        name: XX-viber-connector
        port: { number: 80 }
      path: /viber
  - backend:
      service:
        name: XX-wechat-connector
        port: { number: 80 }
      path: /wechat
  - backend:
      service:
        name: XX-zalo-connector
        port: { number: 80 }
      path: /zalo
  - backend:
      service:
        name: XX-gcloudcx-connector
        port: { number: 80 }
      path: /openmessaging
  - path: /
    backend:
      service:
        name: XX-config
        port: { number: 80 }
```

Notes:

- The TLS configuration is not included in this YAML.

- XX should be replaced by the Helm Release name.

Chapter 2

How to Install/deploy Universal Messaging on Kubernetes

Pre-requisites

As obvious as this may be, you must have a Kubernetes cluster (version 1.18+) to be able to deploy this application. It has to be production ready. We show here how to get one in the most common environments.

You also must have Helm Charts 3.1+ deployed on that cluster. Please see here.

If you need an ingress controller, you can install the NGINX controller. Most of the modern Cloud vendors (Amazon Web Services, Microsoft Azure, Google Cloud Platform) already give you an ingress controller.

You should also have received credentials from your GENESYS representative to download container images. If this is not the case, please contact us.

Installing Universal Messaging

The automated way

The easiest way to install Universal Messaging is the automated way. It takes care of creating the Kubernetes cluster and all that is needed to run the application.

You can find the install script there: https://artifacts.genesyslab.com/universal-messaging-install.sh

On the plus side, you just have to provide coffee… On the negative side, you have less control over the way the cluster is built. If you prefer creating the cluster yourself, please go directly to the next paragraph.

Here is the simplest way to deploy Universal Messaging, say on Microsoft Azure (the options here are mandatory):

```
./install.sh --flavor aks \
--release prod \
--registry-username your-genesys-username \
--registry-password your-genesys-password \
--registry-email your-email
```

You will see some warning about generating passwords for the application, Redis Database, and RabbitMQ. You can provide each of them in the options.

Supported flavors are:

- `aks` , Microsoft Azure
- `eks` , Amazon Web Services
- `gke` , Google Cloud
- `microk8s` , Micro K8S

Here are all the options you can use (mandatory options are marked with a *):

- `--api-root` fqdn-url
  The URL used to reach the Universal Messaging API.

- `--api-password` *password*
  The password for the API admin user, will be stored in a Kubernetes Secret,
- `--cluster` *name*
  The name of the Kubernetes Cluster to be created (when it applies),
- `--config` *path* (default: ./config.json)
  The filename used to store the generated Helm configuration,
- `--dry-run`
  Commands are not executed, use this to see what would be executed,
- `--flavor` *name* *
  The Kubernetes flavor to prepare.
  Values: aks, eks, gke, microk8s,
- `--helm-debug`
  Will run Helm in debug mode, providing more information,
- `--namespace` *name* (default: messaging)
  Universal Messaging will be deployed in this Kubernetes namespace,
- `--rabbitmq-password` *password*
  The password for the RabbitMQ admin user, will be stored in a Kubernetes Secret,
- `--rabbitmq-erlang-cookie` *cookie*
  The Erlang Cookie for RabbitMQ, will be stored in a Kubernetes Secret,
- `--redis-password` *password*
  The password for the REDIS database, will be stored in a Kubernetes Secret,
- `--registry-username` *name* *
  The username to use to download Container images from Genesys Container Registry,
- `--registry-password` *password* *
  The password to use to download Container images from Genesys Container Registry,
- `--registry-email` *email* *
  The email to use to download Container images from Genesys Container Registry,
- `--release` *name* *
  The name of the Helm Release to create,
- `--stage` *name*
  Will install another Helm chart, docker images than the default production ones. Try this only in your lab… Some stages are:
  `dev` , `beta` , `stage`
  Other can exist to address various scenarii.
  Stages can also alter the generated configuration,
- `--verbose` (default)
  Will run verbosely, displaying more information,
- `--workers` **number**
  The number of worker nodes in the Kubernetes cluster.
  Some Kubernetes flavors have restrictions!

Some flavors can have additional options:

- Amazon Web Services `EKS`
- Microsoft Azure `AKS`
  - −az-resource-group *name* *
    The Azure Resource Group to create,
  - −az-resource-location *name*
    The Azure Resource Location to use.
    This is **mandatory** if your account does not have a default location defined,
  - −az-sunscription *id_or_name*
    The Azure Subscription to use for billing.

Notes:

- The API password **MUST** be complex (must have a score of at least 3 on zxcvbn, test site: https://lowe.github.io/tryzxcvbn), if not the `xx-set-admin-password` job will fail and the Helm Chart deploying will also fail.
- When installing on `microk8s` , you must be logged in the host machine that will run it.
- There is only 1 node possible with `microk8s` as of today. You can add more, but by yourself.
- When installing on `Azure` , the script will prevent the number of worker nodes to be less than 3.

The manual way

Before installing and running anything, we must add the Genesys Helm repository:

```
helm repo add genesys oci://registry.genesys-services.com
```

You can look at the charts that will be deployed by pulling them in advance:

```
helm pull genesys/universal-messaging
```

For reference, you can also pull the chart at a specific version, if you need to analyze what it will do:

```
helm pull oci://registry.genesys-services.com/charts/universal-messaging-xx.tgz
```

where `xx` is the version number.

Then, we need to create a new Kubernetes namespace:

```
kubectl create namespace messaging
```

Since the container images for Universal Messaging are not public, we create the Kubernetes secret that will allow us to download them from the Genesys Docker registry:

```
kubectl create secret docker-registry \
  --namespace messaging regcred-genesys \
  --docker-server=registry.genesys-services.com \
  --docker-username=<your-name> \
  --docker-password=<your-password> \
  --docker-email=<your-email>
```

The Universal Messaging Application uses passwords to connect services to their RabbitMQ and Redis components. They are stored in Kubernetes secrets. While, you can let Helm create them randomly, it is advisable to set them yourself. You get better control and you don't take the risk of losing access to Universal Messaging when you upgrade to a more recent version.

The API service administrator password is **required** during the install. Helm will complain if it is not provided. That password is not stored in a secret but in the API service database. It also must be complex otherwise the Helm installation will fail.

```
tee myconfig.yaml &>/dev/null <<EOM
global:
  redis:
    password: r3d1sS3cr3t
rabbitmq:
  auth:
    password:     r2bb1tS3cr3t
    erlangCookie: MyV3ryB1gS3cr3tC00k1e
api:
  admin:
    password: s1ms3cr3t
  api_root: https://www.acme.org
EOM
```

Finally, we deploy Universal Messaging via Helm:

```
helm install \
  --namespace messaging \
  --values myconfig.yaml \
  genesys/universal-messaging
```

You can observe the deployment of all pods by running:

```
watch -n 1 kubectl get pods --namespace messaging
```

This chart will install Redis and RabbitMQ along with all services needed to run the Universal Messaging platform.

You should see all pods getting to the `running` state within a few minutes.

Notes:

- To configure the ingress, please read the section Configuring HTTPS

- If the Ingress Controller used by your platform requires an ingress class, you should add it in your config.yaml (here with `nginx` as an example):

```
ingress:
  className: "nginx"
```

- As keeping password in plain text is a dangerous thing to do, we advise you to use file encryption technologies such as Terraform Vault, Ansible Vault, Mozilla SOPS.
- You can also let Helm create the passwords by itself (except for the API's admin password that is required). In that case, do not forget to backup their values before upgrading the Helm chart (See Upgrading Universal Messaging).
- The value for `api_root` is optional. See Chapter Using the REST API for a more detailed explanation.

The Appveyor way

It is possible to use Appveyor automation to create the Kubernetes cluster and deploy Universal Messaging.

This method is used at Genesys internally.

When creating the git repository for your deployment (called a solution), choose the `GUM` Bitbucket project and the `Universal Messaging (GUM)` template.

If you are deploying a cluster for many customer, check the `optional` checkboxes for the `Project Number` and `Customer Name` fields. The `Solution Name` should follow the following pattern: `gum-cluster-<stage>-<aws region>`, for example: `gum-cluster-prod-us-west-2`. Optionally, you can add the shortname of a service to the name when the deployment will contain only the said service, for example: `gum-cluster-prod-us-west-2-amb`.

If you are deploying a cluster for a single customer, but this is not part of a SOW, check the `optional` checkbox for the `Project Number` field.

The following clusters have been already created:

- `gum-cluster-prod-ap-northeast-1`
- `gum-cluster-prod-eu-central-1`
- `gum-cluster-prod-us-west-2`
- `gum-cluster-prod-us-west-2-amb`

Open the `appveyor.yml` file and modify the variables to match your needs. The FQDN should use the shortname of the regions, for example: `gum-apne-1.genesyscsp.com`. prod clusters should use the `genesyscsp.com` domain, while non-prod clusters should use the `genesyscsp.net` domain.

Notes:

- You can use this method only if you are a Genesys staff.
- You can ignore the Upgrade section as it is simply a matter of forcing the Appveyor build to run again.
- This method also uses a customer Container Registry stored on AWS ECR.

Installing the beta/Development versions

By default, Helm will install the stable version of Universal Messaging and its services.

If you want to try a more recent version than the default (to test a new feature or a fix), you just need to modify which images are downloaded. Of course, Beta versions are less stable, and Development (dev) versions are very unstable.

We suggest to not do this in production, but on a test cluster, this can be useful.

Let's modify the yaml configuration we prepared previously (or better yet a copy of it):

```
tee -a myconfig.yaml >/dev/null <<EOM
  image: {tag: dev, pullPolicy: Always}
config: { image: {tag: dev, pullPolicy: Always} }
commander: { image: {tag: dev, pullPolicy: Always} }
amb-connector: { image: {tag: dev, pullPolicy: Always} }
bedore-connector: { image: {tag: dev, pullPolicy: Always} }
bizmsg-connector: { image: {tag: dev, pullPolicy: Always} }
disqus-connector: { image: {tag: dev, pullPolicy: Always} }
```

```
gbm-connector: { image: {tag: dev, pullPolicy: Always} }
gchat-connector: { image: {tag: dev, pullPolicy: Always} }
infobank-connector: { image: {tag: dev, pullPolicy: Always} }
line-connector: { image: {tag: dev, pullPolicy: Always} }
media4u-connector: { image: {tag: dev, pullPolicy: Always} }
plusmessage-connector: { image: {tag: dev, pullPolicy: Always} }
slack-connector: { image: {tag: dev, pullPolicy: Always} }
teams-connector: { image: {tag: dev, pullPolicy: Always} }
telegram-connector: { image: {tag: dev, pullPolicy: Always} }
viber-connector: { image: {tag: dev, pullPolicy: Always} }
wechat-connector: { image: {tag: dev, pullPolicy: Always} }
zalo-connector: { image: {tag: dev, pullPolicy: Always} }
gcloudcx-connector: { image: {tag: dev, pullPolicy: Always} }
pureconnect-connector: { image: {tag: dev, pullPolicy: Always} }
aws-provider: { image: {tag: dev, pullPolicy: Always} }
azure-provider: { image: {tag: dev, pullPolicy: Always} }
box-provider: { image: {tag: dev, pullPolicy: Always} }
dropbox-provider: { image: {tag: dev, pullPolicy: Always} }
google-provider: { image: {tag: dev, pullPolicy: Always} }
EOM
```

Upgrading Universal Messaging

To be able to upgrade the Universal Messaging deployment safely, you need to ensure the passwords, etc will not be overwritten by the upgrade process. This is particularly true in an Helm environment such as the one we use: `helm upgrade`

Before upgrading, if a YAML configuration file was not created, all passwords must be saved as they would get reset during the upgrade process:

```
cat <<EOM > config.yaml
global:
  redis:
    password: $(kubectl get secrets -n messaging \
               -l app.kubernetes.io/name=api-redis \
               -o jsonpath="{.items[0].data.redis-password}"|base64 --decode)
rabbitmq:
  auth:
    password: $(kubectl get secrets -n messaging \
               -l app=rabbitmq \
               -o jsonpath="{.items[0].data.rabbitmq-password}"|base64 --decode)
    erlangCookie: $(kubectl get secrets -n messaging \
                   -l app=rabbitmq \
                   -o jsonpath="{.items[0].data.rabbitmq-erlang-cookie}"|base64 --decode)
EOM
```

You might also need to add some fields that you used during the installation.

We should also backup the Redis Database:

```
gum-cli backup --host host --password xxx | \
  gzip >| backup-$(date +%Y%m%d%H%M%S).json.gz
```

Once this is done we can launch the upgrade:

```
helm upgrade rrr -f config.yaml genesys/universalmessaging
```

Where `rrr` is the Helm Release name and replace `config.yaml` with the YAML file where you keep the configuration.

Like before, you can observe the deployment of all pods by running:

```
watch -n 1 kubectl get pods --namespace messaging
```

Fine tuning

Deploying only the services you need

It is possible to not deploy some of the services, if you know you will never need them in your Universal Messaging instance.

To do so, mark their `enabled` property to `false` in your Helm YAML configuration, like this:

```
bedore-connector:
  enabled: false
```

```
dropbox-provider:
  enabled: false
```

Extra logging

When you are in the testing phase, you can also turn up the logging to get more information in the logs about what is happening.

To do so, you can simply modify the `logging` property in your Helm YAML configuration:

```
line-connector:
  logging:
    level: DEBUG
```

Logs are also flushed every so often or when an error occurs (written to their output stream, typically stdout). If you need to ensure the logs are written when they are generated, change their flush frequency:

```
line-connector:
  logging:
    flushFrequency: immediate
    level: DEBUG
```

You can also set default log setting at the global level that are superseded locally:

```
global:
  logging:
    level: DEBUG
```

```
line-connector:
  logging:
    level: TRACE
```

Here all services will log at `DEBUG`, instead of `INFO`, and `line-connector` will log `TRACE`.

Log levels are (in decreasing order):

- FATAL
- ERROR
- INFO
- DEBUG
- TRACE

The lower in this list the more verbose the logging becomes and the more impact it has on performance. The Default value is `INFO`.

More complex logging configuration is possible, like:

```
line-connector:
  logging:
    level: "DEBUG;TRACE:{:request}"
```

This set the default log level to `DEBUG` and the log level for the `request` scope to `TRACE`.

Make sure to read the documentation of the Logger at: https://github.com/gildas/go-logger#readme for a more detailed explanation about the logging configuration.

Horizontal Auto-Scaling

By default, Horizontal Auto-Scaling is configured for the services as follows:

```
line-connector:
  autoscaling:
    enabled: true
    minReplicas: 1
    maxReplicas: 10
    targetCPUUtilizationPercentage: 80
```

You can change these values in your own YAML configuration.

Use a Custom Container Registry

The Container images for Genesys services are provided by https://registry.genesys-services.com at specific versions.

It is possible to use a custom registry to download them faster or simply to alleviate any downtime of the original Container Registry.

Most of the Cloud vendors offer a per-cluster registry. Just download the container images using Docker and upload them to your registry:

```
docker login --username xxx registry.genesys-services.com
docker pull registry.genesys-services.com/gum/line_connector
docker tag registry.genesys-services.com/gum/line_connector asia.gcr.io/gum/line_connector
docker push asia.gcr.io/gum/line_connector
```

Then, change your Helm YAML configuration as follows:

```
line-connector:
  image:
    registry: asia.gcr.io
    repository: gum/line_connector
    tag: xxx
    pullPolicy: IfNotPresent
```

If the `tag` property is not given in your config.yaml, the Helm Chart will use the value from `.Chart.AppVersion`, which is the most common situation. Please note that using tags like `latest` is considered dangerous as it leads to inconsistencies since Kubernetes does not know when the container image has actually changed.

If you use `microk8s`, you will set the registry to `none` and the pull policy to: `Never`, and uploading to microk8s registry is done as follows:

```
docker pull....
docker save registry.genesys-services.com/gum/line_connector line_connector.tar
microk8s ctr image import line_connector.tar
```

Using external Redis or RabbitMQ

Genesys uses the Redis Chart and RabbitMQ Chart from Bitnami when deploying Universal Messaging, you can configure them as you wish in your YAML configuration.

If you prefer to use external Redis or RabbitMQ to host the databases or process the message queuing, you just need to disable them and configure the access in your YAML configuration.

For example, you might want to use the Universal Messaging platform with an external RabbitMQ Messaging, such as Amazon MQ or CloudAMQP.

You would simple disable the deployment of RabbitMQ and use the RabbitMQ FQDN from your provider:

```
global:
  rabbitmq:
    host: xyz.rmq.cloudamqp.com
    vhost: "vhostFromCloudAMQP"
    user: "userFromCloudAMQP"
rabbitmq:
  enabled: false # Turn off local deployment of Bitnami's RabbitMQ
```

And create the Kubernetes secret that will hold the password:

```
kubectl create secret generic rrr-rabbitmq \
  --namespace messaging \
  --from-literal rabbitmq-password="p@sswordFr0mCl0ud@mqp!"
  --from-literal rabbitmq-erlang-cookie="ErlangCooki000123From12344CloudAQMP"
```

where `rrr` is the Helm Release name

Resource limits and requests

All Helm Charts are configured with Kubernetes resource limits and requests. You can change them as you see fit in your YAML configuration, as follows (these numbers are just an example):

```
line-connector:
  resources:
    limits:
      cpu: 500m
      memory: 512Mi
    requests:
      cpu: 200m
      memory: 128Mi
```

You can find the default values in the Helm Chart of each service. The connector services use these values:

```
resources:
  limits:
    cpu: 50m
    memory: 64Mi
  requests:
    cpu: 20m
    memory: 32Mi
```

The storage providers use these values:

```
resources:
  limits:
    cpu: 100m
    memory: 64Mi
  requests:
    cpu: 20m
    memory: 32Mi
```

These values were calculated by load testing messages to a Kubernetes instance. The test consisted of blasting 10,000 text messages to the Social Messages over a few seconds and 1,000 1-MiB images and monitoring the CPU and memory used by the services.

Here is the memory usage (in MiB) of the LINE service, during the image test:

And the CPU usage:
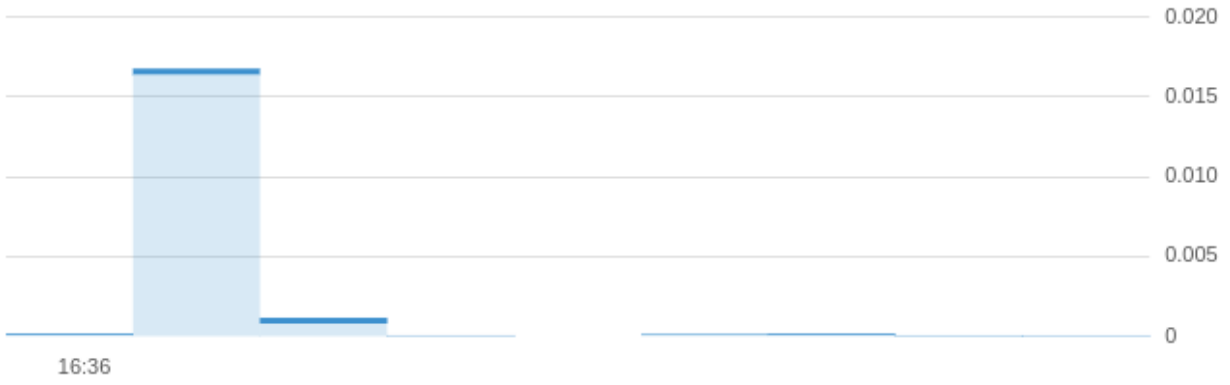
Similarly, for the Azure provider (memory):
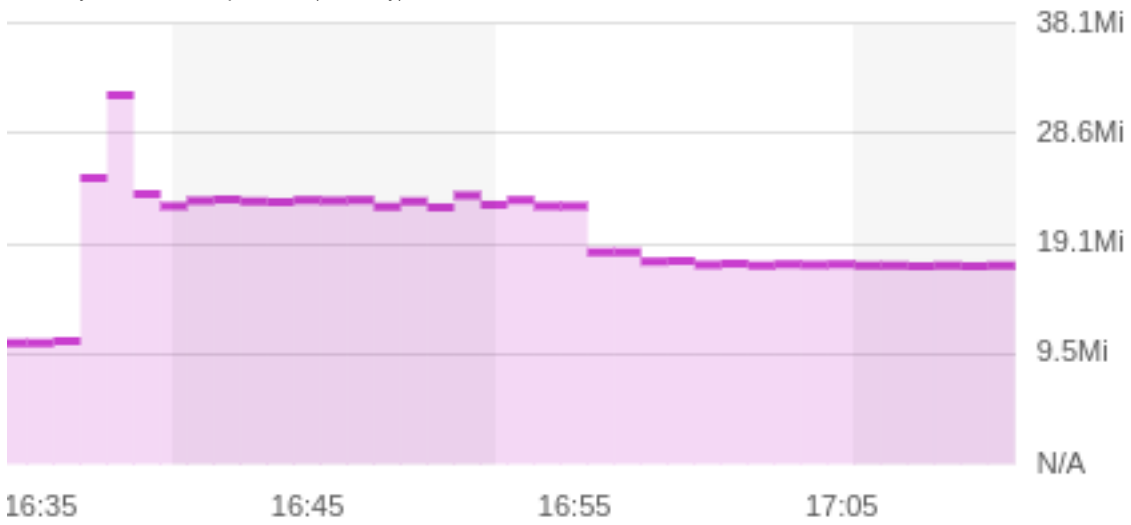
And CPU usage:
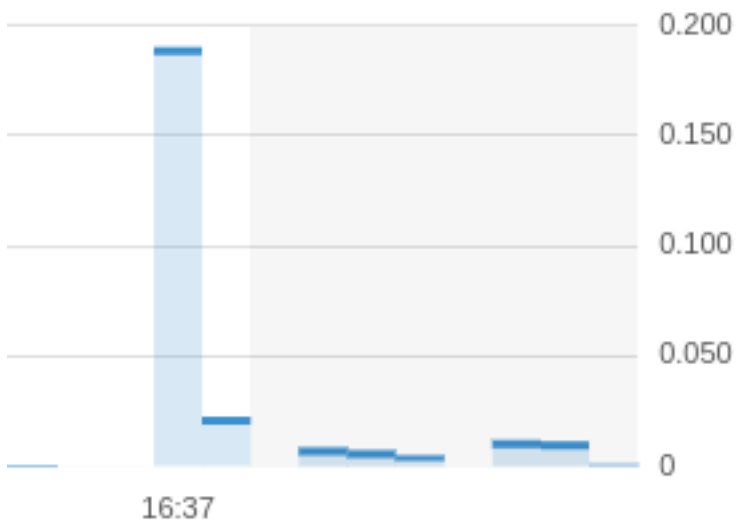
Note: The services are written in Go language. Go has a particular way of handling memory allocation. It does not release the memory released by applications quickly. This allows Go applications to allocate memory back very quickly in case another need comes soon. That explains why we do not see the memory going down rapidly on the memory graphs.

Configuring HTTPS

In order to connect Universal Messaging to Social Messaging services, you will need to configure outside accesses to the Social Messaging connectors.

The communication is ensured via https. Which means you will need a Fully Qualified Domain Name (FQDN).

Depending on your deployment and your existing infrastructure, this can be achieve in different ways.

1. You already have an external DNS on the Internet and can set an FQDN there,
2. You do not have that or do not want to, you will need to ask the Cloud provider to create an FQDN for you.

The first case is fairly straight forward as it will require only some DNS editing once you get the static Internet IP address. Also, if the cloud is your own Datacenter (Virtual Machines, Private Cloud, Bare Metal, etc), this will be, most probably, your only solution.

The second case depends heavily on the cloud you use and can be set up only after the LINE integratio is already installed (as we need some services installed). We will describe the process for a few clouds in a moment.

Getting an FQDN with AWS

You first need to own an IP Domain. If you do not have any, the easiest is to get one from Amazon's Route 53 service.

To do that, from the AWS Console go to the Route53 service, click on "Register Domain" and follow the instructions.

Then go to the "Certificate Manager" service, click on "Request a Certificate", request a public certificate and add your domain "*.mydomain.com".

Once you get the domain and the certificate, you can get the arn of the latter and update the Kubnetes Ingress annotations:

```
metadata:
  annotations:
    alb.ingress.kubernetes.io/listen-ports: '[{"HTTPS":443}, {"HTTP":80}]'
    alb.ingress.kubernetes.io/certificate-arn: arn:aws:acm:ap-northeast-1:123456:certificate/xxxx
    alb.ingress.kubernetes.io/actions.ssl-redirect: '{"Type": "redirect",
        "RedirectConfig": { "Protocol": "HTTPS", "Port": "443", "StatusCode": "HTTP_301"}}'
```

Make sure the third line fits in one line, we had to split it in two to fit on the document page.

Use the arn of the certificate from the AWS console.

Finally add a new CNAME to your DNS zone and alias it to the ALB DNS from the ingress. This can be done directly in the AWS Console or with the command line:

```
target=$(kubectl get ingress \
  --namespace messaging \
  prod-messaging \
  --output jsonpath='{.status.loadBalancer.ingress[0].hostname}' \
)
cat > cname.json <<EOM
{
  "Changes": [{
    "Action": "CREATE",
    "ResourceRecordSet": {
      "Name": "messaging.acme.com",
      "Type": "CNAME",
      "AliasTarget": {
        "DNSName": "${target}",
        "EvaluateTargetHealth": false
      }
    }
  }]
}
EOM
zoneid=$(aws route53 list-hosted-zones --output json \
  jq -r '.HostedZones[] | select(.Name == "acme.com.") | .Id' |\
  cut -d/ -f3
)
```

```
aws route53 change-resource-record-sets \
  --hosted-zone-id $zoneid \
  --change-batch file://cname.json
```

Note: static IP addresses, names, and certificates are paying options with Amazon Web Services and they are not the responsibility of Genesys.

Getting an FQDN with Azure

First you need to get the `Helm` release name and the external IP address that was assigned to the `ingress` service (reverse proxy):

```
RELEASE=$(helm list --namespace messaging --output json | jq -r '.[0].name')
PIP=$(kubectl get ingress \
  --namespace messaging \
  --selector "app.kubernetes.io/instance=$RELEASE" \
  --output jsonpath="{.items[0].status.loadBalancer.ingress[0].ip}")
```

Then you need to ask Azure for an FQDN:

```
RESOURCEGROUP=$(az network public-ip list
  --query "[?ipAddress!=null]|[?contains(ipAddress, '$PIP')].[resourceGroup]"
  --output tsv)
PIPNAME=$(az network public-ip list \
  --query "[?ipAddress!=null]|[?contains(ipAddress, '$PIP')].[name]" --output tsv)
az network public-ip update \
  --resource-group $RESOURCEGROUP  \
  --name  $PIPNAME \
  --output table \
  --dns-name acme-messaging
```

Replace the `dns-name` option with the name you want to use. After a few minutes you should get a response from `aks` in the form of a JSON object.
The property `.dnsSettings.fqdn` will contain the fqdn you can use for your LIS installation.

The next step is to get a certificate that will authenticate this FQDN. This will depend on the certificate provider you use.

If you forget the FQDN, you can always get it from Azure:

```
az network public-ip list \
  --query "[?ipAddress!=null]|[?contains(ipAddress, '$PIP')].[dnsSettings.fqdn]" \
  --output tsv
```

Note: static IP addresses and certificates are paying options with Microsoft Azure and they are not the responsibility of Genesys.

Getting an FQDN with Google Cloud Platform

First, you need to get a static IP address for your Kubernetes cluster:

```
gcloud compute addresses create my-static-ip --global
```

Once this is done, you can get its value like this:

```
gcloud compute addresses describe my-static-ip --global
```

Update the Ingress definition with the following annotation:

```
kubernetes.io/ingress.global-static-ip-name: my-static-ip
```

Also, all services accessible via the ingress must have the type `NodePort` , so you should update them in your config.yaml. For example:

```
api:
  service:
    type: NodePort
```

Note: It can take a few minutes for the Ingress to reflect the IP address.

For the FQDN, you have several choices:

- you can ask Google for an FQDN at Google Domains
- you have your own FQDN, you just need to point your nameserver to the static from Google, and configure your SSL certificate accordingly.
- You can also use nip.io or sslip.io to get an FQDN automatically (typically, the FQDN is `<static-ip>.nip.io`)

You can also use a Google-managed SSL certificate.

Note: static IP addresses, names, and certificates are paying options with Google and they are not the responsibility of Genesys.

Logging Configuration

We recommend you to use the logging solution that is available with your cloud provider.

- On Google Cloud, the best is to use the default Stackdriver logging.
- On Microsoft Azure, Log Analytics will be the preferred way.
- On Amazon Web Services, using the Amazon Elastic Container Service for Kubernetes (EKS), the logs are integrated automatically with Amazon CloudWatch and CloudTrail.

Now, if you want to run your own logging support, it is possible as well. There is a good tutorial on how to deploy Elasticsearch and Kibana on the Kubernetes website itself.

See also:

- Setting Up Logging with Kubernetes

You can also read the logs via kubectl logs directly:

```
kubectl logs -n messaging -1 --tail --follow -l connector=gcloudcx | \
  bunyan -L -o short
```

To pretty-read the logs, please get the bunyan log reader or run it through bunyan in Docker

Tips and Tricks for Checking the deployment

Access the Redis Database

To connect to the Redis database (from within the cluster), you can run a container with the Redis Client (`XXX` is the name of the Helm Release):

```
REDIS_PASSWORD=$(kubectl get secret \
  --namespace xyz RRR-redis -o jsonpath="{.data.redis-password}" \
  | base64 --decode)
kubectl run --namespace xyz redis-client --rm -it \
  --image bitnami/redis:6.2.5 -- redis-cli \
    -h RRR-MMM-redis-master-0.xyz.svc.cluster.local \
    -a $REDIS_PASSWORD
```

Replace `xyz` by the Kubernetes namespace, `RRR` by the Helm release name of your deployement, and `MMM` the microservice name ("gcloudcx-connector", for example).

Or, if you have `redis-cli` installed on your machine, you can forward ports with Kubernetes:

```
kubectl port-forward service/xyz-MMM-redis-master 6379:6379
redis-cli -a $REDIS_PASSWORD
```

Then, you can use the REDIS query language to check things, e.g.:

```
keys config:*
```

Backing up the Redis Database

Genesys provides a client tool to backup and restore the database.

You can download it from:

- Linux: https://artifacts.genesyslab.com/gum-cli-1.0.17.linux.7z
- MacOS Intel: https://artifacts.genesyslab.com/gum-cli-1.0.17.darwin-amd64.7z
- MacOS M1: https://artifacts.genesyslab.com/gum-cli-1.0.17.darwin-arm74.7z

- Windows: https://artifacts.genesyslab.com/gum-cli-1.0.17.windows.7z
- Docker: registry.genesys-services.com/gum/gum-cli

To backup the database, you just run the following:

```
gum-cli backup --host host --password xxx --output backup.json
```

Where `xxx` is the administrator password for the API.

You can run some more complex command that will compress the backup and use the current date in the filename:

```
gum-cli backup --host host --password xxx | \
  gzip >| backup-$(date +%Y%m%d%H%M%S).json.gz
```

To restore the database, simply run:

```
gum-cli restore --host host --password xxx -i config.json
```

If the backup was compressed, then you would run this:

```
zcat backup-20210805230406.json.gz | gum-cli restore --host host --password xxx
```

You can store the default values for `gum-cli` in $HOME/.config/gum-cli/config or $HOME/.gum-cli in YAML:

```
host: host.local.net
password: xxx
```

Then you do not need them on the command line anymore: `gum-cli backup -o config.json`

You can get logs from `gum-cli` execution by doing thing:

```
gum-cli backup --log mylog.log --host host --password xxx --output backup.json
```

To get more verbose logs:

```
DEBUG=1 gum-cli backup --log mylog.log --host host --password xxx --output backup.json
```

To read the logs, please get the bunyan log reader or run it through bunyan in Docker

Finally, you can get the documentation by running:

```
gum-cli --help
```

Access the RabbitMQ Dashboard

If you did not set your own user, the chart will set it to `user`.

if you did not set your own administrator password, the chart will create one. To retrieve the password:

```
RABBITMQ_PASSWORD=$(kubectl get secret \
  --namespace xyz RRR-rabbitmq -o jsonpath="{.data.rabbitmq-password}" \
  | base64 --decode)
```

The RabbitMQ service is accessible to the pods in the cluster on port `5672` and its web dashboard on port `15672` at the DNS name: `RRR-rabbitmq.xyz.svc.cluster.local`

Then forward the dashboard port of RabbitMQ:

```
kubectl port-forward --namespace xyz service/RRR-rabbitmq 15672:15672
```

Finally, go to [http://127.0.0.1:15672] to get to the RabbitMQ Management site.

Replace `xyz` by the Kubernetes namespace and `RRR` by the Helm release name of your deployement.

AWS EKS Fargate pitfalls

If you deploy Universal Messaging on AWS EKS Fargate, there will be some manual quick fixes to implement.

This is due to the relative immaturity of Fargate when it comes down to managing a Kubernetes Cluster (EKS).

The idea of Fargate is to add flexibility and automatic growth of the Kubnertes Cluster by removing the need of pre-allocation of the number of Kubernetes Workker nodes (also called minions historically.)

While being very interesting theoritically, it has one major drawback: Fargate is extremely slow. To achieve that growth, it assigns 1 Kubernetes Pod to 1 Kubernetes Worker Node. THat means every time a Pod is created, AWS will first instantiate a Virtual Machine (EC2), configure it to be an EKS node, connect it to the Kubernetes network, and finally start the Pod. Typically, a Pod starts in about 90-120 seconds.

Another drawback comes from the lack of support of standard Kubernetes objects. It is impossible to start a DaemonSet for instance. Persistent Volumes must be created manually before their Persistent Volume Claim and the Pods that will use them. The Helm Charts for Universal Messaging help with the latter, please check their configuration (When deployed with Appveyor, it is even simpler, it's a simple config item).

The issue with Pods and Persistent Volumes creates a very common situation when Helm deploys Universal Messaging, AWS EKS Fargate creates the Redis Pods before their Peristent Volumes. When that happens, the symptoms will be Pods in the `pending` state for a very long time (even for Fargate). To fix it, simply delete the existing Pods, they will be recreated properly by AWS.

For all these reasons, while it is doable, we do not recommend AWS EKS Fargate. On AWS, you should go with EKS Managed.

Chapter 3
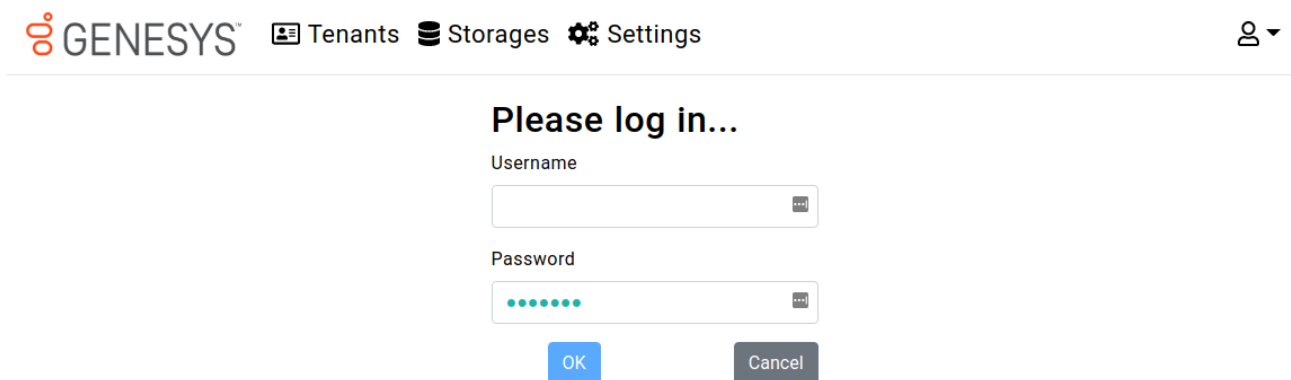
Configuration

The Universal Messaging application can be configured either via a website or its own REST API. The latter will be explained in the next chapter.

The configuration website is available at the main URL of the application. For example:

```
https://messaging.acme.com
```

When you get connected, you need to enter the administrator's credentials that were configured during the installation:

Once you are logged in, you can configure the Universal Messaging application.

Note: You can change the language of the website with the Language Menu in the bottom-right.

Common Settings

In the "Settings" page, you can configure things such as:

- the Outbound Proxy. All services that access the Internet will use that value.
- the Notifier Destinations. Allows services to "tell" when they start/stop, …
- the Default Commander (See the section about Commanders)

Notifier Destinations

You can set any number of supported Notifier Destinations. Services will send messages to all of them.

Typically, services will send messages when after they have started successfully and before they shutdown. If a service uses more than one Kubernetes Pod, every Pod will send notifications.

A Notifier Destination is also bound to a topic. This means that the Notifier Destination will only receive messages from that topic. Here are the supported topics:

- `Subscriptions` : Messages are sent when a subscription experiences an error,
- `Default` : All other messages are sent to that topic.

Note: If there is no Notifier Destination for the `Subscriptions` topic, the related messages will be sent to the `Default` topic.

Note: Setting Notifier Destinations, while very useful, does **not** replace proper monitoring and alerting that you should set up at the Kubernetes level (AWS Cloudwatch, Azure Monitor, Google Cloud Monitoring, etc). When Pods cannot start *enough*, they will never send notifications, and supporting Kubernetes Deployments like Redis and RabbitMQ, being third party applications, cannot send notifications. For example, if the Redis database does not start, the connector services that uses it will also not start properly, but since it needs the database to retrieve its configuration, it will not be able to send notifications.

Genesys Cloud

You can configure a Notifier Destination to send messages to Genesys Cloud. To do so, you need to install the Generic Webhook AppFoundry App in your Genesys Cloud organization.

Then you need to create a Group in Genesys Cloud and add some members. The members will receive the notifications.

Once the group is created, you need to configure the Generic Webhook in the Genesys Cloud Integrations page. Note down the Webhook Notification link on the Details page and configure a new Mapping:

Finally, add a new Notifier Destination in Universal Messaging with the Webhook link:

Slack

To setup Slack, you need to create an account on their website (you can use your Google Account or your AppleID, or any email account).

Once you have created an account, click on the "CREATE A NEW WORKSPACE" button on the top right of the page. You can also do this from the Slack application on your Desktop or your smartphone.

Type the name of the workspace you want to create:

Step 1 of 3

What's the name of your company or team?

This will be the name of your Slack workspace — choose something that your team will recognize.

Genesys/Test	38

☑ Let anyone with an **@genesys.com** email join this workspace.

Next

Add a new channel:

Step 2 of 3

What's your team working on right now?

This could be anything: a project, campaign, event, or the deal you're trying to close.

production	70

Next

You can add team members or skip the next step:

Step 3 of 3

Who do you email most about production?

To give Slack a spin, add a few coworkers you talk with regularly.

Ex. ellis@genesys.com

⊕ Add another   🔗 **Get a shareable invite link** instead

Add Teammates   Skip this step

After the workspace is started, go to its settings:
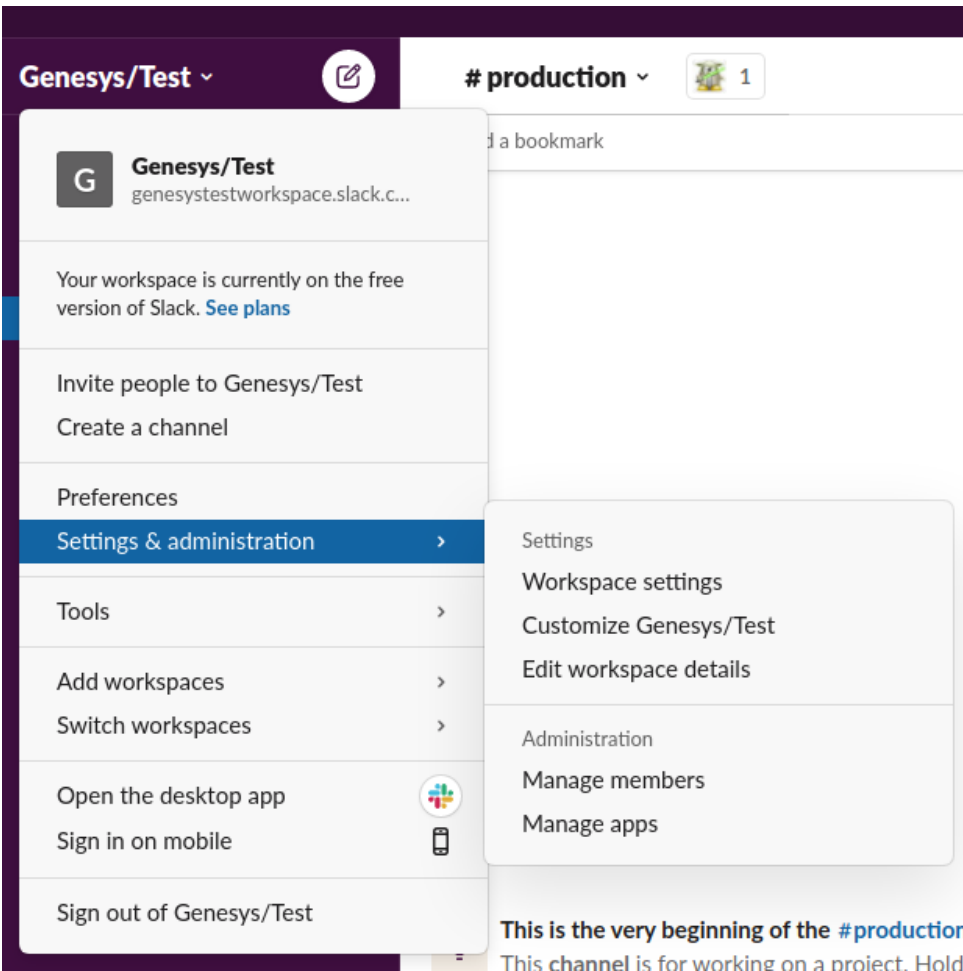
Click on "Configure Apps", and start typing "Incoming Webhooks" in the "Search App Directory" input on the top right of that page:
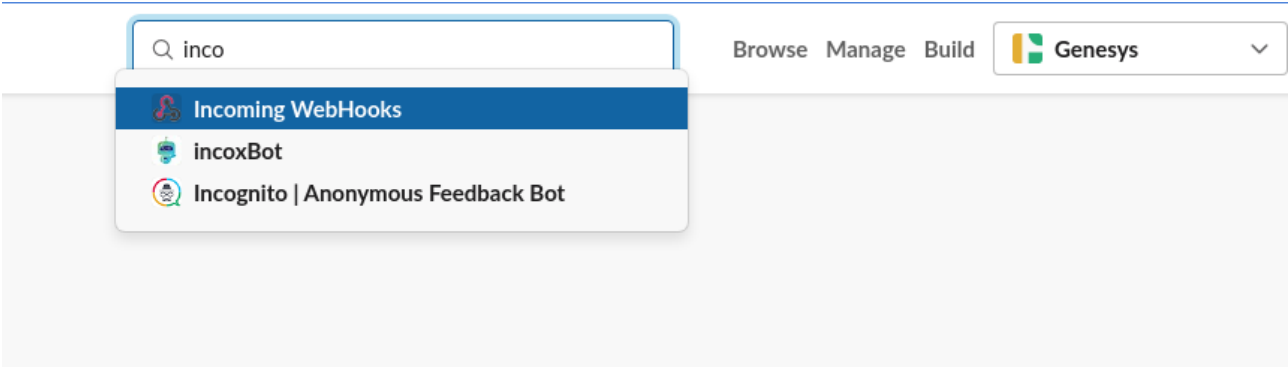
And click on the "Add to Slack" button.

Once added, select the channel that will receive notifications, and click on "Add Incoming Webhooks integration":
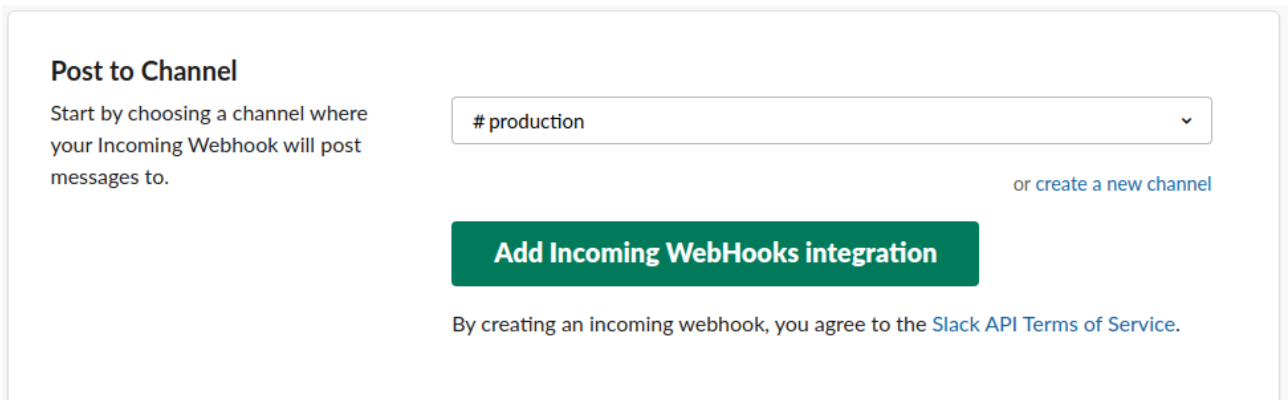
Once added, copy the Webhook URL from that page and paste it on the Universal Messaging Settings page:
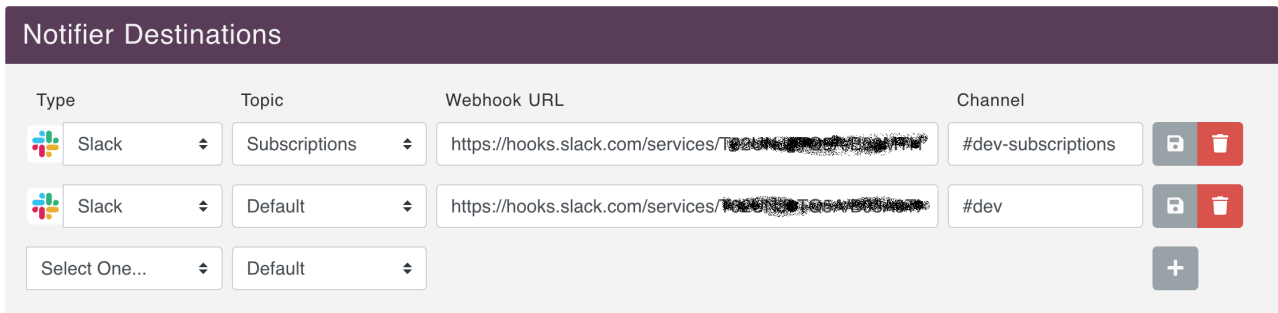
And click on the "+" sign.

Discord
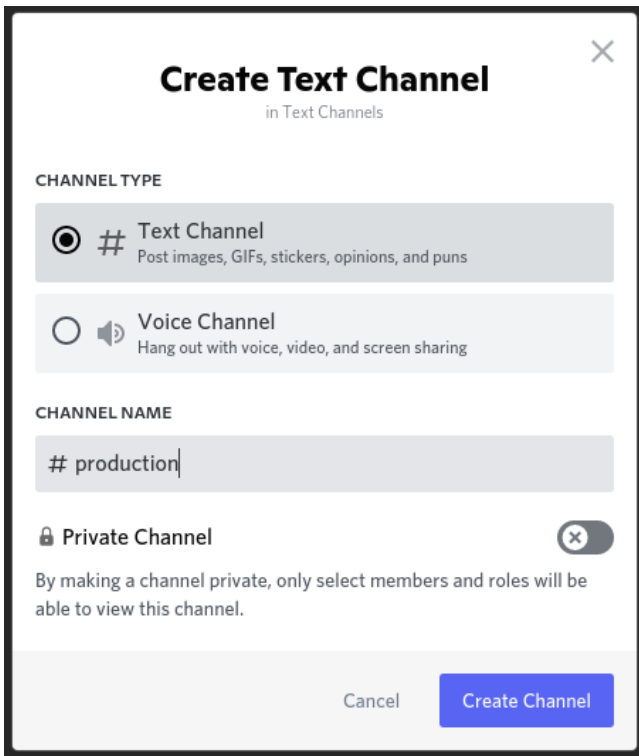
To setup Discord, you need to create an account on their website.

Then, click on the "+" button on the left bar, to "Add Server" and follow the dialogs:
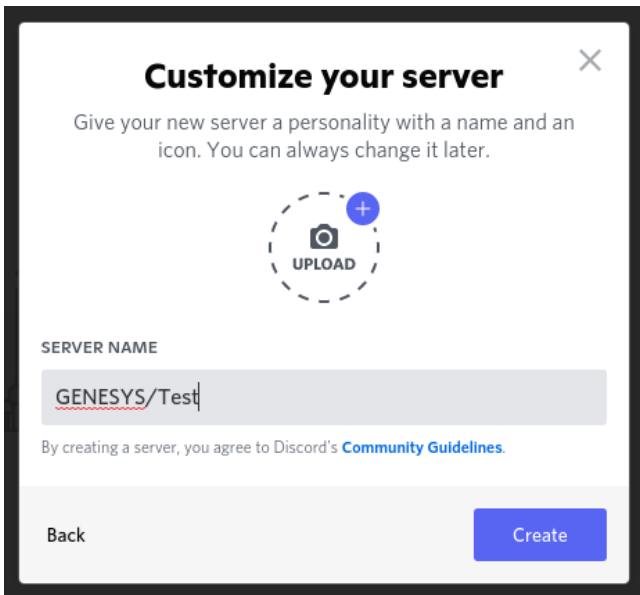
Once created, you can add new text channel, by click on the "+" sign in the "TEXT CHANNELS" collection:
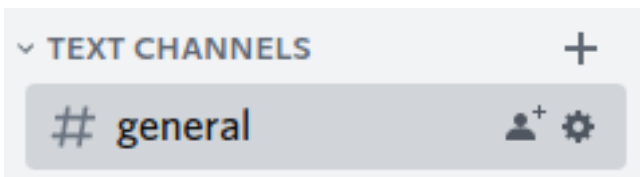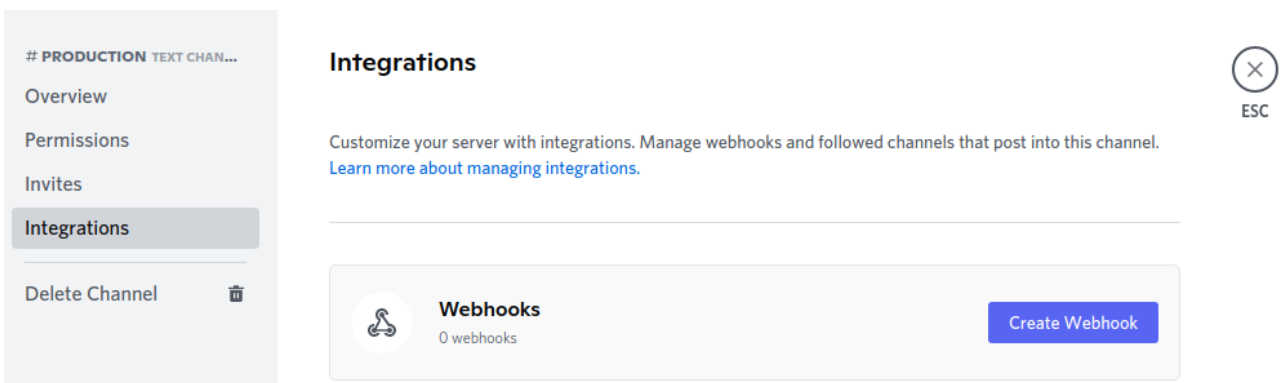
Once created, click on the icon, to edit the settings:
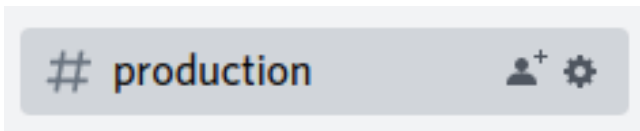
And select "Integrations":

Click on the "Create Webhook" button, and copy the Webhook URL:
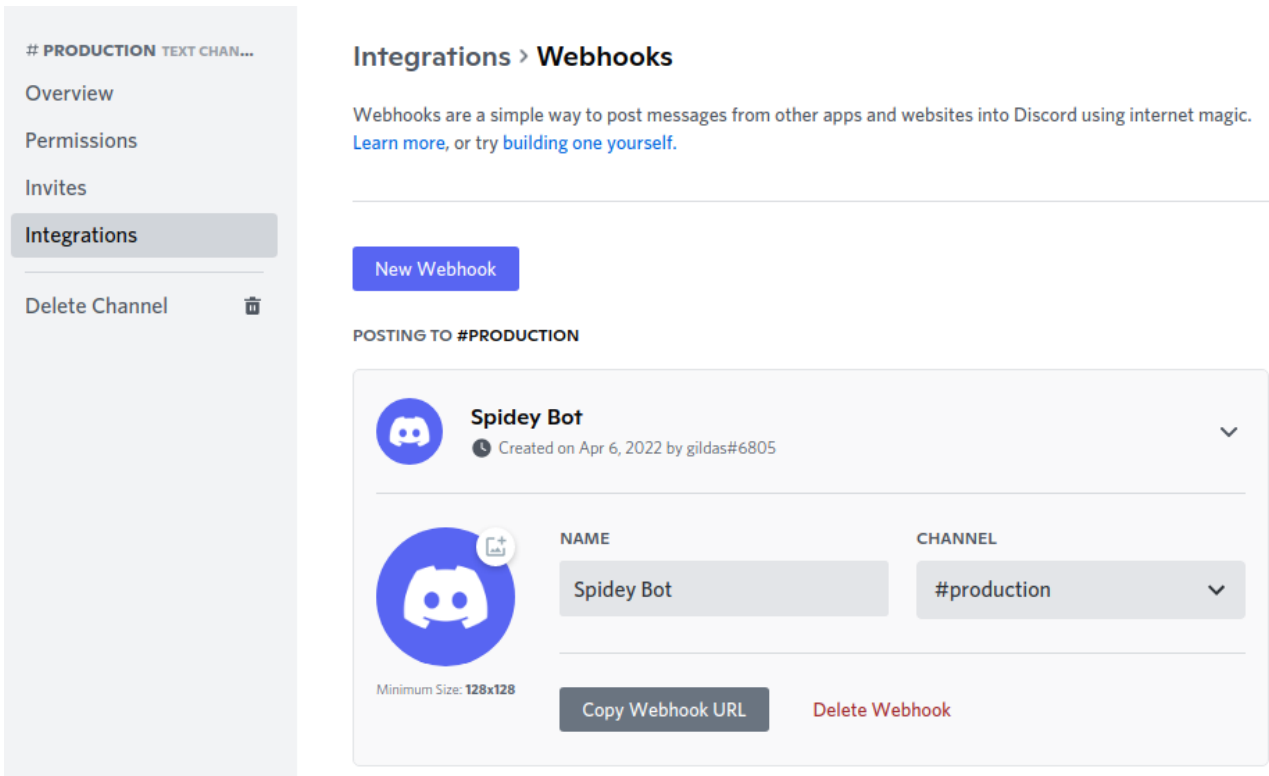
Paste the URL on the Universal Messaging Settings page:
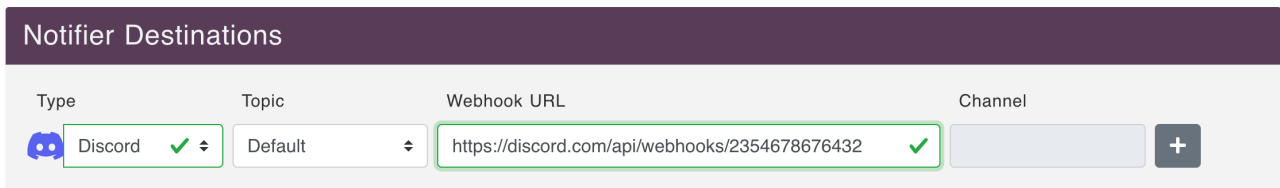
And click on the "+" sign.

PagerDuty

To setup PagerDuty, you need to create a **routing key** on their console.

Paste the routing key on the Universal Messaging Settings page:
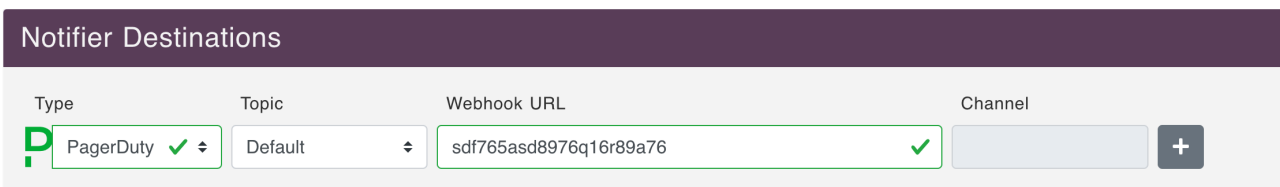
Commanders

Commander is the service that can execute commands sent by the agents from the CX Platforms.

The most common command, as of today, is used to allow the agent to continue a conversation with their guest at a later moment.

Commander configuration can be done on the Settings page, for the Default Commander, or in the Tenant's Commander tab.

The Commander configuration looks like this:

Google Dialogflow™

Dialogflow Agent Code?

dialogflow-command-continue-1.0.0.zip

Location

Select One...

Language

Select One...

Environment Id

Enter a Dialogflow Environment Identifier (e.g.: draft)

Google Cloud Authentication

Project Identifier

Enter an identifier

Client Identifier

Enter an identifier

Client Email

Enter an email

Private Key Identifier

Enter an identifier

Private Key

Enter a private key

The link to the zip file contains the code to import in Google Dialogflow. The location, language, Environment Id are given when you configure the agent in Google Dialogflow.

To configure Google Dialogflow, you first need to create an Agent on your Google Dialogflow console:

Dialogflow Essentials   Global ▾   ◈ Agents   CREATE AGENT

⊕ Create new agent

≡ View all agents

Search agents

Click on "CREATE AGENT" and fill in the form:

Here, we gave the agent the name of the Commander command. The actual name does not matter much.

Once created, you need to go to the settings of the agent and click on the "IMPORT FROM ZIP":

The default Environment (from the same settings page) is called "draft". Its name needs to be filled in the Universal Messaging config webpage.

On the Google Dialogflow console, go to the agent's settings and click on the Project:
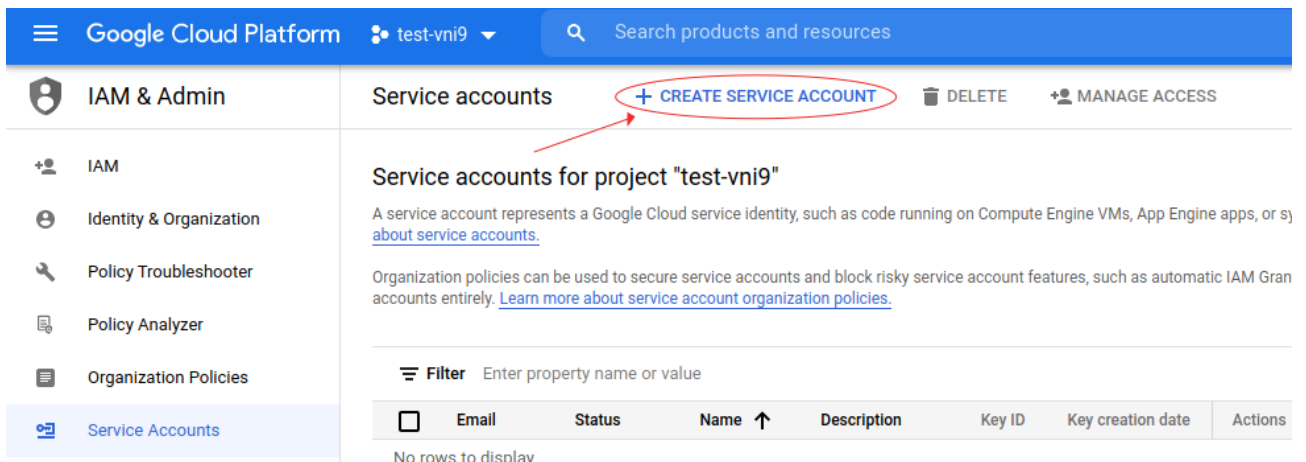
This will bring you to the Google Cloud Platform console. Go to the "IAM & Admin" section, and to "Service Accounts", and click on "CREATE SERVICE ACCOUNT":

And give the service account a name, then click on "DONE":

Create service account

1 Service account details

Service account name
simcommander

Display name for this service account

Service account ID
simcommander @test-vni9.iam.gserviceaccount.com ✕ ⟳

Service account description

Describe what this service account will do

CREATE AND CONTINUE

2 Grant this service account access to project (optional)

3 Grant users access to this service account (optional)

DONE   CANCEL

Then go in the account setting by click on it on the service accounts list and go to the keys tab:

← simcommander

DETAILS   PERMISSIONS   KEYS   METRICS   LOGS

Keys

⚠ Service account keys could pose a security risk if compromised. We recommend you avoid downloading service account keys and instead use the Workload Identity Federation . You can learn more about the best way to authenticate service accounts on Google Cloud here .

Add a new key pair or upload a public key certificate from an existing key pair.

Block service account key creation using organization policies.
Learn more about setting organization policies for service accounts

ADD KEY ▾

Type	Status	Key	Key creation date	Key expiration date
No rows to display				

Click on the "ADD KEY" and create a JSON key:
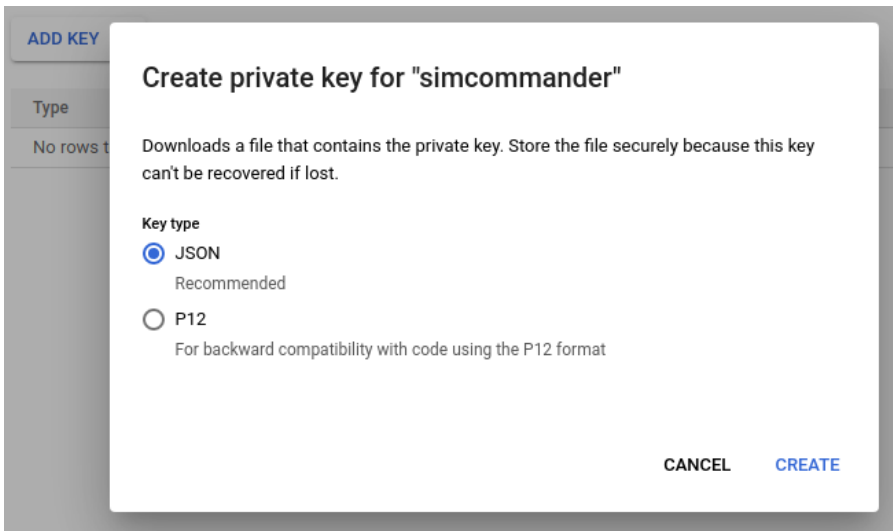
The JSON authentication file is automatically saved on your computer. On the Universal Messaging Config page, you can either fill in each field of import the authentication JSON file using the small icon under "Google Cloud Authentication" on the config website.

Storages

Storages are used to store attachments from Social Media and/or CX Platforms when no public URL is available or restrictions within these systems preventing attachment access.

On the "Storage" tab, Just select the type of Storage you want to define and fill in its credentials from the provider.

Also, set the "Purge Delay" in seconds. This will used by the Storage Provider services to remove attachments that are not used anymore. After a chat is closed, the services will wait for the given purge delay and then delete the attachments related to that chat from their cloud storage.

You should select one of the storages you defined as the "Default Storage". That will be the one used by Tenant's Messaging and CX Platform by default.

AWS Storage

First, we need to configure the Storage on Amazon Web Services.

Create a new AWS S3 bucket (we will store its name in an environment variable):

```
export BUCKET=mybucket
aws s3api create-bucket \
  --bucket $BUCKET \
```

```
    --region ap-northeast-1 \
    --create-bucket-configuration LocationConstraint=ap-northeast-1
```

If you will use AWS CloudFront to serve files, you should also block public access to the bucket:

```
aws s3api put-public-access-block \
--bucket $BUCKET \
--public-access-block-configuration \
  'BlockPublicAcls=true,
  IgnorePublicAcls=true,
  BlockPublicPolicy=true,
  RestrictPublicBuckets=true'
```

Create an IAM Policy for managing objects in the bucket:

```
cat >| ${BUCKET}-bucket-policy.json <<-EOF
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "ManageObject",
    "Effect": "Allow",
    "Action": [
      "s3:ListBucket",
      "s3:GetObject",
      "s3:GetObjectAcl",
      "s3:GetObjectTagging",
      "s3:PutObject",
      "s3:PutObjectAcl",
      "s3:PutObjectTagging",
      "s3:DeleteObject"
    ],
    "Resource": [
      "arn:aws:s3:::${BUCKET}",
      "arn:aws:s3:::${BUCKET}/*"
    ]
  }]
}
EOF
export MYBUCKET_POLICY=$(aws iam create-policy \
  --policy-name ${BUCKET}-s3 \
  --policy-document file://${BUCKET}-bucket-policy.json \
  --query 'Policy.Arn' \
  --output text \
)
echo "Policy ARN: $MYBUCKET_POLICY"
```

Then, create an AWS IAM User and assign it the new policy:

```
aws iam create-user \
  --user-name ${BUCKET}-bucket
aws iam attach-user-policy \
  --user-name ${BUCKET}-bucket \
  --policy-arn $MYBUCKET_POLICY
```

Assign an access key for that user, which will be used later to configure the Storage in the Universal Messaging Config:

```
aws iam create-access-key --user-name ${BUCKET}-bucket --output json
```

Write down the `AccessKey.AccessKeyId` and `AccessKey.SecretAccessKey` properties.

Next, we need to give AWS CloudFront access to the bucket.

We need an Origin Access Identity (OAI). Try to reuse one you already have, since you can have only 100 OAI in a given AWS Account! If you need to create one:

```
aws cloudfront create-cloud-front-origin-access-identity \
  --cloud-front-origin-access-identity-config \
    CallerReference="gum-deployments",Comment="For GUM AWS S3"
```

Then create a CloudFront distribution:

```
export BUCKET_LOCATION=$(aws s3api get-bucket-location \
  --bucket $BUCKET \
  --query  LocationConstraint  \
  --output text
)
aws cloudfront create-distribution --generate-cli-skeleton >| distribution-config.json
aws cloudfront create-distribution \
  --distribution-config "$(jq \
    --arg oai "$OAI" \
    --arg bucket_fqdn "${BUCKET}.s3.${BUCKET_LOCATION}.amazonaws.com" \
    '.DistributionConfig |
    .CallerReference = "distribution-for-gum-s3" |
    .Origins.Items[0].Id = $bucket_fqdn |
    .Origins.Items[0].DomainName = $bucket_fqdn |
    .Origins.Items[0].S3OriginConfig.OriginAccessIdentity="origin-access-identity/cloudfront/"+$oai |
    .DefaultCacheBehavior.TargetOriginId = $bucket_fqdn
    ' distribution-config.json)" \
  --output json
```

Write down the `.Distribution.DomainName` , it will be used to configure the Storage in Universal Messaging's Config.

Give CloudFront Read Access to the Bucket:

```
aws s3api put-bucket-policy \
  --bucket $BUCKET \
  --policy "$(jq \
    --arg oai "$OAI" \
    --arg bucket "${BUCKET}" \
    '.Statement[0].Principal.AWS = "arn:aws:iam::cloudfront:user/CloudFront Origin Access Identity "+$oai |
    .Statement[0].Resource = "arn:aws:s3:::"+$bucket+"/*"
    ' <<<'
{
  "Version": "2008-10-17",
  "Id": "PolicyForCloudFrontPrivateContent",
  "Statement": [{ "Sid": "1", "Effect": "Allow", "Action": "s3:GetObject" }]
}')"
```

Now, just go to the Universal Messaging Config and add your AWS Storage:

Tenants

Tenants describe the relationship between a Social Media (Messaging) and a Customer Experience Platform.

Eacch Tenant has a name and a subscription Identifier. The latter comes from Genesys Customer Subscription Manager.

The API Authentication allows IVR engines to send messages to Social Medias by sending REST Requests to the Universal Messaging API. The avaiable Authentication types are:

- Basic
- None (which means no authentication necessary, not recommended)

The CX Platform tab supports the following:

- Genesys Cloud CX,
- Genesys PureConnect

The Messaging tab supports the following:

- Apple Messages for Business,
- Bedore Web,
- BizM KakaoTalk,
- Disqus,
- Docomo +Messages,
- Google Business Messages,
- Google Chat,
- Infobank KakaoTalk,
- LINE,
- Media4U,
- Microsoft Teams,
- Slack,
- Telegram,
- Viber,
- WeChat,
- Zalo.

Genesys Cloud CX

On the third tab of the Tenant, you can choose the Customer Experience Platform (CX Platform). The first choice allows a connection with Genesys Cloud CX via its Open Messaging API.

Credentials

In the Genesys Cloud Administration console, go to "People & Permissiom/Roles", and add a new Role, with the following permissions:

- `architect:flow:view` ,

- `conversation:communication:view` ,

- `conversation:message:receive` ,

- `conversation:message:create` ,

- `conversation:message:view` ,

- `conversation:participant:wrapup` ,

- `messaging:integration:all` ,

- `responses:library:view` ,

- `responses:response:view` ,

- `routing:message:manage` .

People & Permissions / Roles / Permissions / OpenMessaging Integration

	Permission	Description	License	Conditions	Division Aware
☑	Conversation > message > Create	Create a message	PureCloud 3		
☑	Conversation > message > Receive	Receive a message	PureCloud 3		
☑	Conversation > message > View	View conversation messages	PureCloud 3		
☑	messaging > integration > All Permissions	Assigns all integration permissions, including any future permissions	PureCloud 3		
☑	messaging > integration > Add	Create/Add an integration with a Messaging provider	PureCloud 3		
☑	messaging > integration > Delete	Delete an integration with a Messaging provider	PureCloud 3		
☑	messaging > integration > Edit	Update an integration with a Messaging provider	PureCloud 3		
☑	messaging > integration > View	View integrations created with a Messaging provider	PureCloud 3		

Then, go to "Integrations/OAuth", and add a new Client and give it the new Role:

Upon saving, you will receive a Client ID and a Client Secret. Note these as you will need them in Universal Messaging's Config.

Now open the Universal Messaging Config and add a new Tenant, then go to the "CX Platform" tab, and enter the values you received earlier:

Open Messaging

Then, give your GCloudCX Open Messaging Integration a name, a token (You can choose any token you like, a UUID for example) and fill in the Integration WebHook URL:

Integration Name

| Universal Messaging With XXX | ✔ |

Integration ID

| |

Integration WebHook

| https://there.acme.com/openmessaging | ✔ |

Integration Token

| sup3r_s3cr3t | ✔ |

Once you saved the Tenant, you should be able to set the Inbound Message Flow. You can import the sample from the config website to your Genesys Cloud Organization (do not forget to fill in the proper queue in that flow).

The Custom Data will be added to the Open Messaging's Participant Data as seen on the Agent's UI:

Some data is already set depending on the Messaging Connector that is configured with the Tenant:

- `media` ,
  The name of the Messaging Connector, one of:
  Apple Messages for Business, Bedore, BizM KakaoTalk, Disqus, Google Business Messages, Google Chat, Infobank KakaoTalk, LINE, Media4U, Slack, Teams, Telegram, Viber, WeChat, Zalo
- `capabilities` ,
  The comma-separated list of capabilities supported by the guest device as provided by the Social Media,
  Apple Messages for Business
- `country` ,
  The country where the guest is located,
  Viber, WeChat
- `city` ,
  The city where the guest is located,
  WeChat
- `deviceOS` ,
  The OS of the device of the guest, Viber
- `deviceType` ,
  The device type of the guest, Viber
- `enterpriseId` ,
  The Slack Enterprise Identifier, if applicable,
  Slack
- `enterpriseName` ,
  The Slack Enterprise Name, if applicable,
  Slack
- `forum` ,
  The forum where the conversation was started,
  Disqus
- `googleAgent`
  The Google Agent that was used, Google Business Messages
- `googleConversationId`
  The Google Conversation ID, Google Business Messages
- `googleEntryPoint`
  The Google Entry Point if present,
  Google Business Messages
- `googleGTIN`
  The Google Shopping Global Trade Item Number (GTIN) if present,
  Google Business Messages
- `googleMerchantOfferId`
  The Google Shopping Merchant Id if present, Google Business Messages
- `googleNearPlaceId`

The Google Near Place Identifier if present,
Google Business Messages

- `googlePlaceId`
The Google Place Identifier if present,
Google Business Messages
- `googleProductCategory`
The Google Shopping Product Category if present,
Google Business Messages
- `googleProductImageURI`
The Google Shopping Product Image URI if present,
Google Business Messages
- `googleProductTitle`
The Google Shopping Product Title if present,
Google Business Messages
- `googleProductURI`
The Google Shopping Product URI if present,
Google Business Messages
- `group` ,
The group identifier as provided by the Social Media.
Apple Messages for Business
- `intent` ,
The Intent that started this conversation (if any).
Apple Messages for Business
- `locale`
Apple Messages for Business, Google Business Messages, Viber, WeChat
- `province` ,
The province where the guest is located,
WeChat
- `subscribeScene` ,
The place where the guest is,
WeChat
- `thread` ,
The thread where the conversation was started,
Disqus
- `workspaceId` ,
The Slack Workspace Identifier,
Slack
- `workspaceName` ,
The Slack Workspace Name,
Slack

Agent Name

You can also choose the default name of the agent to show on the Social Media side (provided you choose "Prefix Agent Name" in the Messaging tab).

Commander usage

Check the "Use Commander" box, if this Tenant will use Commander. Uncheck "Default Commander", if you want to use a specific Commander configuration (see the Commander tab of that Tenant).

Enter a Commander Prefix (for now stick with `>`) that will be used by agents to send commands to the Commander:

Commander

☑ Use Commander

☑ Default Commander

Commander Prefix

> ✔

Google maps

You can configure this section to display a nice map when the guests send their location to the agents, as show here:
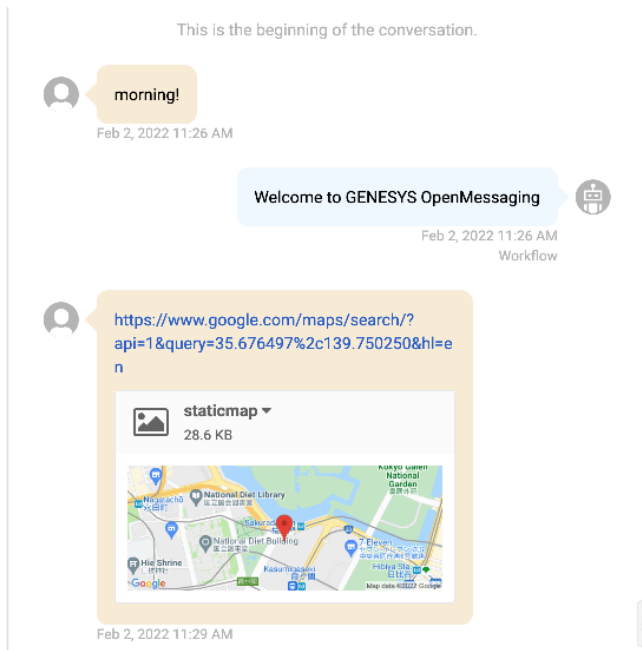
To compute the map, you need to enter your Google Maps Key and, eventually, Secret here:

Additional Configuration in Genesys Cloud CX

Once the Tenant is saved, a new Platform will appear in Genesys Cloud CX Administrator.

Connect to your Genesys Cloud CX Administration console, and go to the Message / Platforms section:
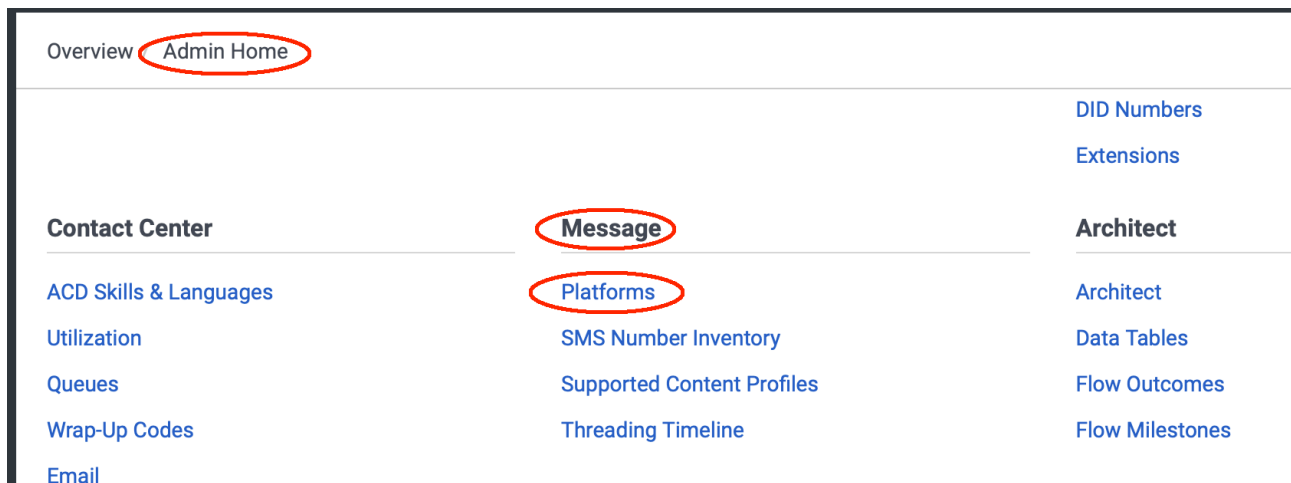
You should see a new Open Messaging Platform with the same name as the one you entered in Universal Messaging Config:
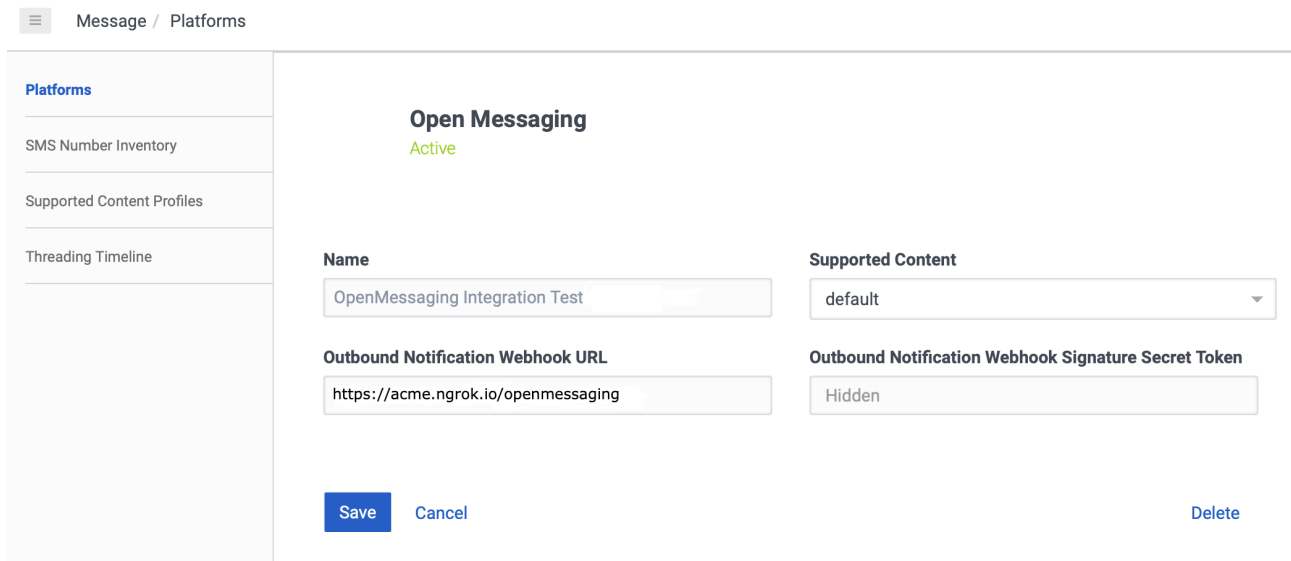
There is nothing to do here, just verify the values entered in Universal Messaging Config were carried over properly.

Then create a new Inbound Message Flow in Architect. Here is an example (called "TestOpenMessaging"):
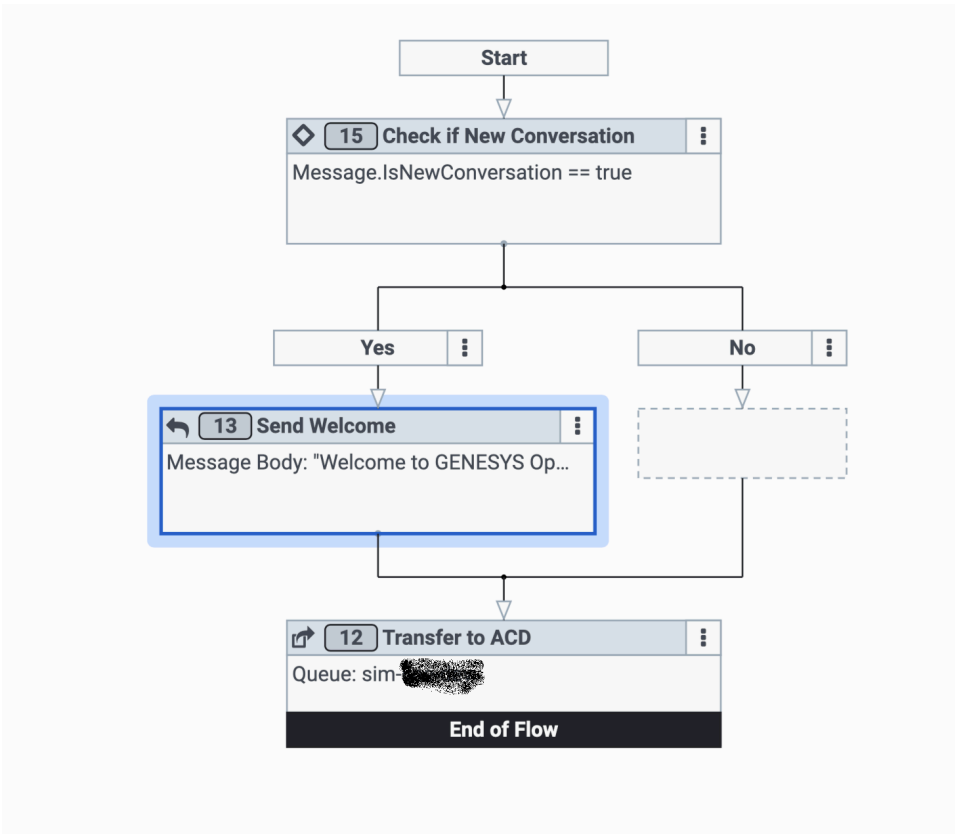
Once you publish it, it will appear in the section Routing / Message Routing of your Administration console:

Select the flow that was created with Architect, and assign the address from the Open Messaging Platform:

Once, this is done, if you go back to the Open Messaging Platform, you will see it is "linked" to the flow:

≡ Message / Platforms

Platforms

SMS Number Inventory

Supported Content Profiles

Threading Timeline

Open Messaging
Active

Associated with the TestOpenMessaging flow.

Name

OpenMessaging Integration Test

Supported Content

default

Outbound Notification Webhook URL

https://acme.ngrok.io/openmessaging

Outbound Notification Webhook Signature Secret Token

Hidden

Save   Cancel   Delete

Genesys Cloud CX is now configured to send Open Messaging conversation to Universal Messaging.

Note: If you want to link several Tenants to the same Inbound Message Flow, just choose distinct Open Messaging Names and add them to the same Message Route.

Apple Messages for Business

The first thing is to obtain an Apple Message Service Provider Identifier from Apple. This should be done only once per running instance of Universal Messaging by the people that will deploy Universal Messaging.

And give the Universal Messaging webhook URL to Apple:

Notes:

- The Platform API Base URL should be the URL of the cluster with `/amb` appended to it.
  For example: `https://mycluster.mydomain.com/amb`
- The Client Landing Page URL should be the Platform API Base URL appended with `/register'.   For example:` https://mycluster.mydomain.com/amb/register'

Then, you need to create a new Apple Messages for Business Tenant in Universal Messaging Config: Send the new MSP to Apple for review. Once approved, you will receive a new MSP Identifier and its secret.

Once you have these values, configure your Helm Chart before deploying Universal Messaging:

```
config:
  ambMSPId:     "653abce3-dead-beef-7fe3-247634adce53"
  ambMSPSecret: "b+Cq/hsghTYSGTYH456456DGNCVN456456/dfgfdsGg="
```

The next part involves the customer, they need to go to register themselves with Apple. The approval process is conducted by Apple and can take a few days.

Once their Business Account has been approved by Apple, the customer will need to link their Business Identifier to Universal Messaging. When configuring the Messaging Platform, they will enter the Universal Messaging register webhook URL, given by a Genesys Representative, https://gum-xxxx.genesyscsp.com/amb/register:

Messaging Platform

Select the messaging platform to use for this Messages for
Business account.

○ Apple authorized commercial messaging platform

> Choose   ⌄

Advanced ⌃

● URL provided by your messaging platform

Select this option if your messaging platform provider gave a specific URL
to connect your Messages for Business account.

> Enter URL
> https://   .com/amb   ✓

○ Proprietary or internal platform

Select this option if your organization will self-manage message routing
and agent desktop.

> Choose   ⌄

Cancel   Submit

They should be directed automatically to the Universal Messaging registration where they can enter their information so a Genesys Representative can approve them and create the Tenant in Universal Messaging Config.

To configure Apple Pay, the customer will need to go to Apple Pay, sign up with a Payment Service Provider and create a Merchant ID. They will need to provide their Merchand ID, name, certificate, Private Key, and Payment Gateway to Universal Messaging. The domain to use with the Payment Service Provider is the one used by Universal Messaging (the webhook hostname).

Note: Apple Pay configuration is still a bit experimental and will need manual intervention from the deployment team. This is due to the fact no real payment gateway is available for testing.

To configure the Integrated OAuth2 Authentication, the customer will need to sign up with an OAUTH2 provider (like Google or Auth0). They will need to provide their response type, OAuth scope, and Client Secret to Universal Messaging.
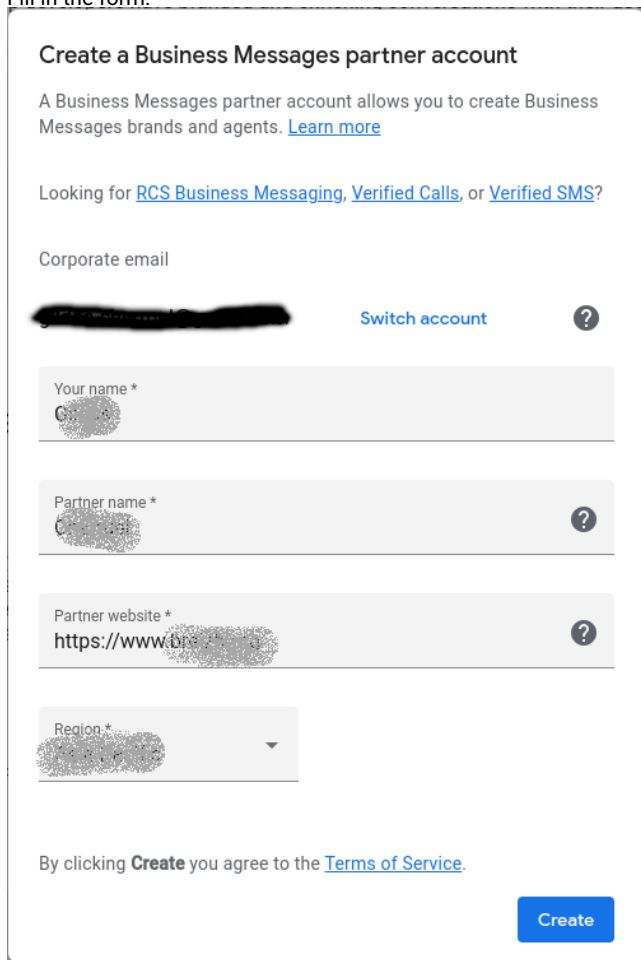
Google Business Messages

Here, we are going to create a new Google Partner to connect to our Universal Messaging deployment.

You will need a Google Account on Google Cloud that has some Billing attached.

Once you have that, just go to the Google Business Messages Console and click on the `Create partner account` button.

Fill in the form:

Create a Business Messages partner account

A Business Messages partner account allows you to create Business Messages brands and agents. Learn more

Looking for RCS Business Messaging, Verified Calls, or Verified SMS?

Corporate email

Switch account ?

Your name *

Partner name * ?

Partner website * ?
https://www.bre

Region * ▾

By clicking **Create** you agree to the Terms of Service.

Create

Once the account is created, go to the `Partner account settings` page, and complete the Settings page. You do not need to set a webhook now. This will be done later when we add Google Agents.

Go to the `Brands` page and create one. Then go to the main page of the console by clicking on `Business Communications` on the top-left of the page.

Go to the `Service account` page and create a key. Keep the JSON file that is automatically downloaded safe!

Now, create an `agent` and click on it to go to the configuration page. You can configure the `Agent information` as you see fit.

Go to the `Integrations` page, and configure the `webhook` , generate a `Client Token` . Do not proceed further.

Now go to the Universal Messaging Config website, and add a new Tenant with Google Business Messages:

Messaging Type

Google Business Messages   ⇕

> *Please, register your partner with Google Business Messages to obtain your Partner Id and Key:*
> *Register with Google Business Messages*
> *You can configure Google Business Agents on the Google Business Communications Console:*
> *Google Business Communications Console*

Partner Id

Enter Partner Id

Partner Key

Enter Partner Key

Client Token

Enter Client Token

Webhook

https://breizh.ngrok.io/gbm   📋

Upload the `Google Cloud Authentication` from the JSON file you saved earlier.

The `Partner Id` comes from the Partner Settings page of the Google Console:
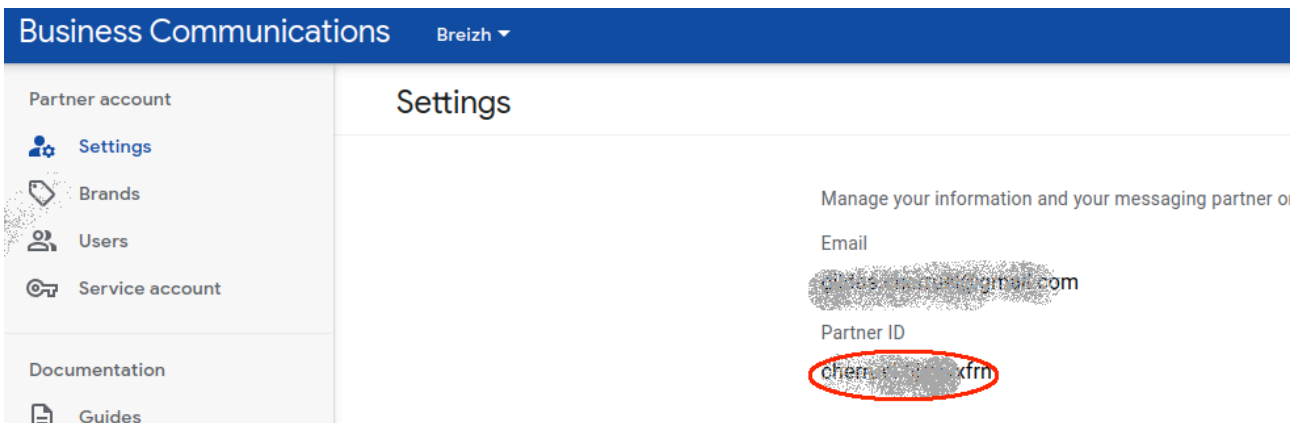
The `Client Token` is the value you created on the webhook configuration for the agent in the Google console.

Since we did not create a webhook for the Partner, just enter a dummy text in the `Partker Key` . **Note:** In a future version, Universal Messaging will stop asking for that Parner Key. It is simply a `Client Token` at the Partner level.

Save the Tenant. Of course, this supposed you also configured the CX Platform tab as explain before.

Copy the webhook from the Universal Messaging Config and paste it in the `Configure your webhook` dialog box we left unfinished earlier and hit `Verify` .

You are now ready to start conversations between Google guests and CX Agents.

For testing, you can go to the Agent Overview page of the Google Business Messages Console and click on the `Android` , `iOS` or `Send` button to get the instructions on how to start a conversation.

You can also add some `Locations` for that agent. The new locations will also have some tests buttons. See the CX Section to learn how to route conversation depending on these locations.

Once the testing is done, you should `Verify` and `Launch` your agents so real-life guests can connect.

Google Chat

To configure Google Chat, you need to signup for a Google Workspace account with access to Google Chat. In the Google Cloud Project that governs your Google Workspace, you need to enable the Chat API and the Drive API, if the users will be allowed to send documents to Universal Messaging via their Google Drive.

Note: Make sure to set the "Billing" correctly for your Google Cloud Project.

Once enabled, you need to create some credentials for Universal Messaging to use. On the Google Cloud Console, after the Google Chat API is enabled, you should see a message asking you to create some credentials. Click on the `Create Credentials` button and select `OAuth client ID`.
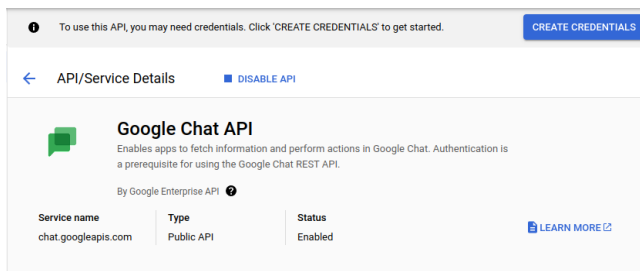
Select `Application Data` as the data that will be accessed and `No, I'm not using them` for using GCE, GKE, or GCF:

Click `Next`. Give a name and an ID to the service account (and, eventually, a description):
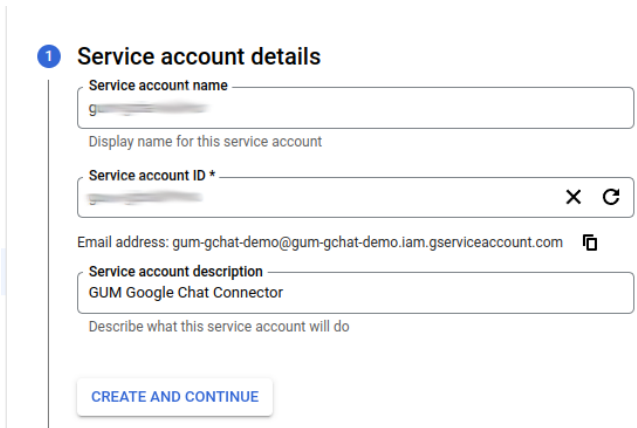
And click `Create and Continue` and `Done` . (We do not need more roles nor user access than the default ones.)

Back on the Google Chat API screen, in the `Credentials` section, you should see the new service account. There is a warning asking you to configure the consent screen. Click on the `Configure Consent Screen` button and choose `Internal` as the user type. Then configure the App information to your liking and click `Save` .

Now, click on the `Service Accounts` section and click on the service account you just created. Click on the `Keys` tab and click on the `Add Key` button. Select `JSON` as the key type and click `Create` . A JSON file will be downloaded. Keep it safe!
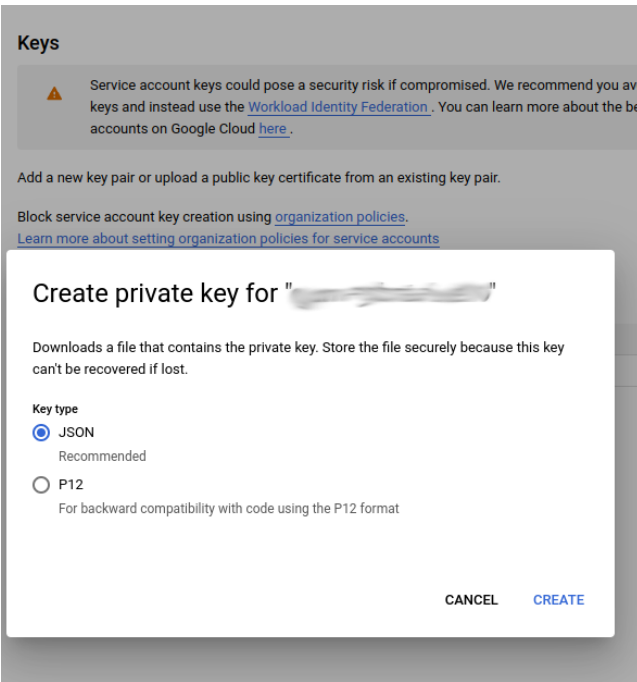
Now, go back to the `Enabled APIs & Services` section, select the `Google Chat API` , and click on the `Configuration` tab of the Google Chat API. Write down the App ID (the text is gray just under `Application Info`), give a name, an Avatar URL, and a description to the bot; add the functionalities for `Receive 1:1 messages` and `Join spaces and group conversations` ; configure the `App URL` to the URL of your Universal Messaging deployment (something like https://gum-xxxx.genesyscsp.com/gchat, where xxxx is the actual Universal Messaging deployment); choose the `Permissions` , either everyone in your workspace organization or specific people; finally, click `Save` .

Now, go to the Universal Messaging Config website, and add a new Tenant with Google Chat, provide the App ID and the credential JSON file.

LINE

The customer will need a Business Account with LINE Business, they can create one at LINE for Business.

They will need to pick the channel they want to use with Universal Messaging and note down the **Channel ID** and the **Channel Secret** (further down on the same web page) from the `Basic Settings` page:

Then, the **Bot basic ID** (with its @ sign) from the `Messaging API` page:

They should send you these values.

If they do not use the LINE Switcher API, do not enter anything in the **Swicher Service Code** field.

If they want to use the LINE Switcher API, they also will need to write down several configuration values from the LINE Switcher API page and send them to you.

For instance, here is a sample configuration:

They will need the **Switcher Service Code** from their LINE, inc. representative.

They will need the **Secret** from the destination that represents Genesys Universal Messaging:

Basic settings   Messaging API   LIFF   Security   Switcher API   Statistics   Roles

Destination

Refer and update information regarding each destination

Destination name   ジェネシス-【CCDemo】GENESYS-Manual

Edit

Status

Destination ID

Switcher secret

Webhook URL   https://   genesys   /line

Edit

And the **Switcher Destination Id** from the third-party Bot. That destination will be used by Universal Messaging to "switch" the customer when the conversation is done:

Basic settings   Messaging API   LIFF   Security   Switcher API   Statistics   Roles

Destination

Refer and update information regarding each destination

Destination name   DEMO endpoint 【CCDemo】GENESYS

Edit

Status

Destination ID

Once the configuration is done, send back to the customer the Webhook URL from the Tenant's Messaging web page:
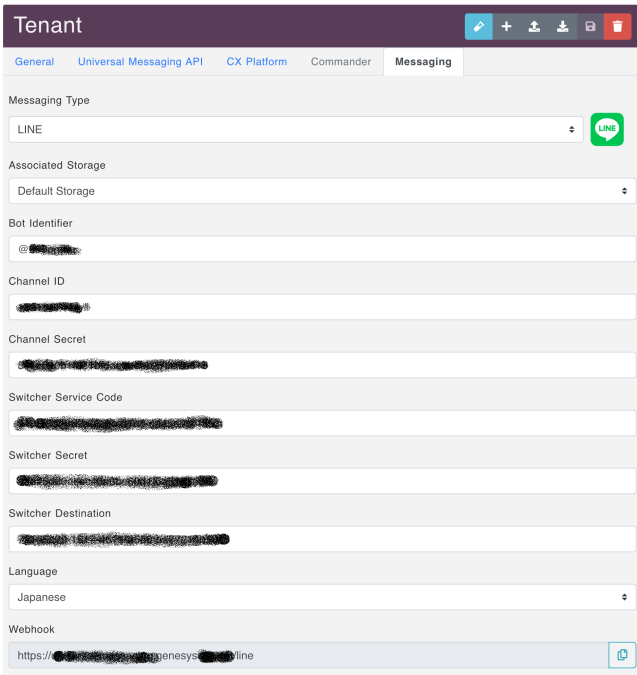
You will also notice a QR-Code at the bottom of the Tenant's LINE Configuration page. It can be used to test the configuration from your phone.

They will need to add it to their LINE configuration. If they do not use the LINE Switcher API, they will need to configure the webhook URL in the `Messaging API` page:
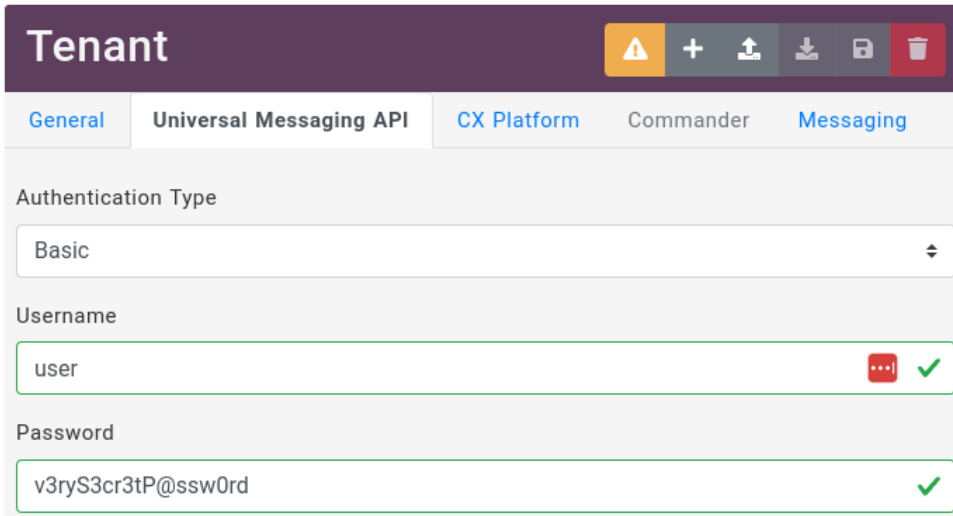
If they use the LINE Switcher API, they will need to configure the webhook URL in the `Switcher API` page under the destination that represents Genesys Universal Messaging:
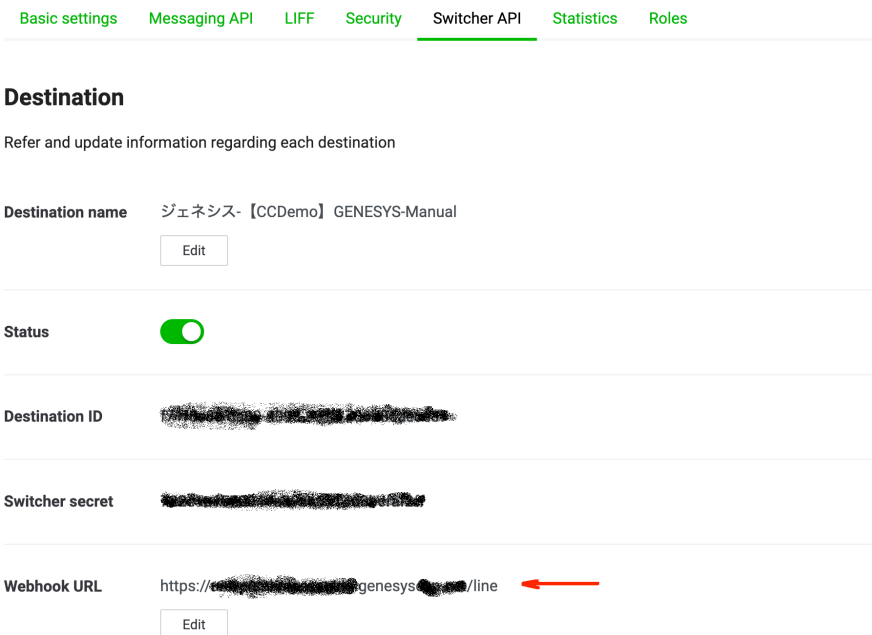
Using LINE's Call To LINE

Call To LINE is a feature from LINE allowing a third-party software, such as an IVR, to send messages to LINE users only by knowing their mobile phone numbers. If the customer has subscribed to this feature, they can use it with Universal Messaging.

You will need to add some credentials to the Tenant's configuration:

Then, you should send these credentials and the URL of the LINE service to them. It is a URL of the form `https://xxx/api/v1/messages`. Where `xxx` is the FQDN of the Universal Messaging cluster.

The customer should have received the User Documentation that explains how to use this feature. (Please refer to that documentation for more details.)

Microsoft Teams

To configure Microsoft Teams, the customer must reach out the register link for Universal Messaging. Depending on the customer's region, the link will be different. The link is provided by the Universal Messaging team.

The first time the customer clicks on the link, they will be asked to fill in a form to register their Microsoft Teams organization:

Your Microsoft Teams Tenant is not yet registered with us

Please, fill in the form below to register it

Business Name

Enter Business Name

Firstname

Enter your first name

Lastname

Enter your last name

Email

Enter your email address

Microsoft Teams Tenant Identifier

Enter your Microsoft Teams Tenant ID

Microsoft Teams Bot Name

Enter the name used by your users to call the Teams Bot

Microsoft Teams Bot Accent Color

Microsoft Teams Bot Color Icon (png only, 192x192 pixels)

Choose File | No file chosen

Image Preview

Microsoft Teams Bot Outline Icon (png only, 32x32 pixels, transparent or while)

Choose File | No file chosen

Submit

Note: If the customer sends several request to the register link, upon filling the form, they will see a message saying Genesys is already in the process of registering their organization. They will need to wait for the email from Genesys. If they are already registered, the message will give them the link to download the manifest file.

A Genesys representative will create a new tenant in Universal Messaging and contact the customer on the given email address. In that email there will be a link to download the manifest file to import in their Teams organization.

When the Universal Messaging Tenant is created, the Teams Connector will automatically associate the Teams Bot name, accent color, and icon to the tenant. The customer can change those values in the Universal Messaging Teams register page.

Success: Your Microsoft Teams Tenant is now registered with us

Please download the Microsoft Teams Manifest and import it to your Teams Organization. The manifest will define a new Microsoft Teams Bot named GUM-DEV the users can contact.

You can modify the following Teams Bot information (you will need to download the manifest again and re-install the Bot in your Teams Organization, however):

Microsoft Teams Tenant Identifier

Microsoft Teams Bot Name

Microsoft Teams Bot Accent Color

Microsoft Teams Bot Color Icon (png only, 192x192 pixels)

Choose File | No file chosen

Image Preview

Microsoft Teams Bot Outline Icon (png only, 32x32 pixels, transparent or while)

Choose File | No file chosen

Update

Modifying the Bot name, accent color, and icons does not require the customer to contact Genesys. However, if the manifest has been loaded already to Microsoft Teams, the customer will need to re-upload the manifest file to Microsoft Teams. The manifest version is automatically incremented when the customer changes the Bot name, accent color, and icons.
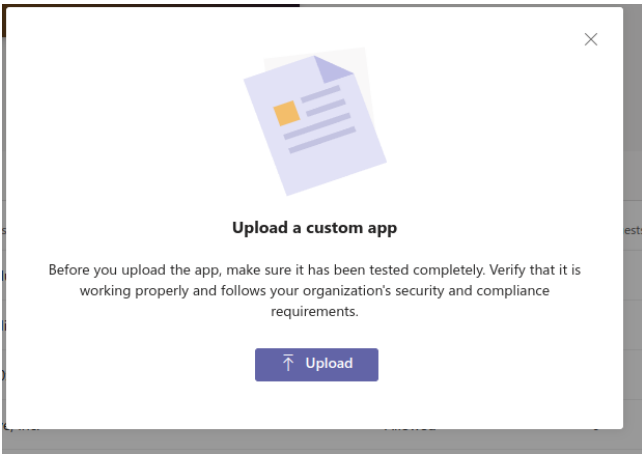
If the Teams App needs to be approved by a Teams Administrator. They need to go to Microsoft Teams Admin Portal and go to the "Manage Apps" section.

They should find the app with a "Submitted" Publishing status and a "Blocked" status:
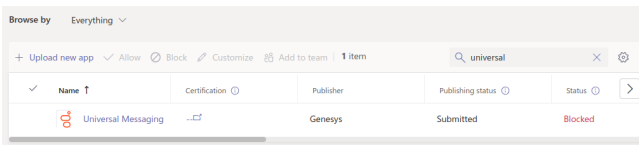
They should click on the App and then "Publish" it. After a few minutes, the app should be available in the Teams App Catalog:
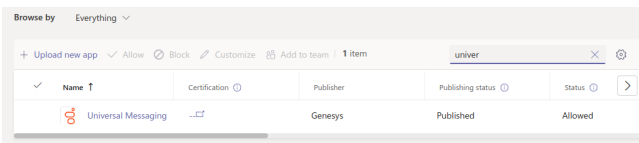
Now, you can configure the bot in Universal Messaging. Go to the Universal Messaging Config website, and add a new Tenant with Microsoft Teams and use the Bot ID and the secret you created earlier:
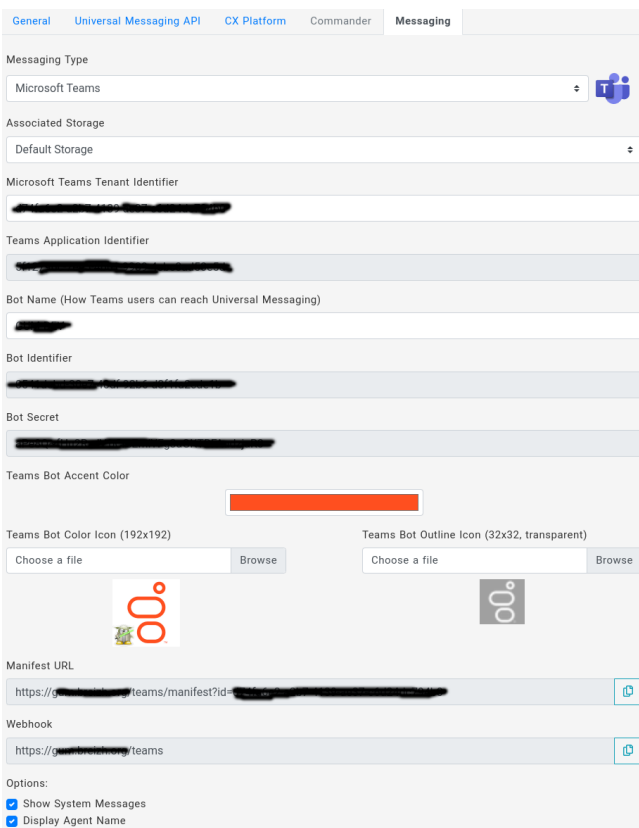
Crafting the Bot

It is also possible to create the Microsoft Teams Bot from scratch. Customers that would want to host the Bot definition on their own Microsoft Azure account would typically belong to this category.

The Bot definition will be created nn the Microsoft Teams Developer Portal. After logging in, click on the `New App` button in the `Apps` section:
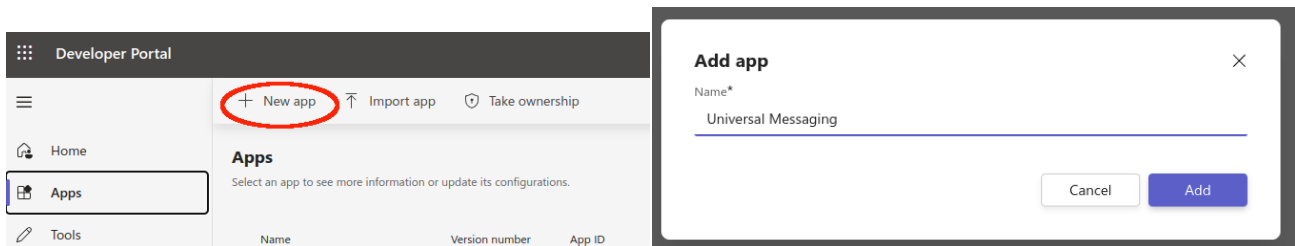
Then, fill in the Basic Information for: App names, Description, and App URIs. The short name will be used by Microsoft Teams users to invoke the Bot:

Note: If the Microsoft Teams App will be served by Universal Messaging, You do not have to fill in the Descriptions, Developer, and URLs fields. Universal Messaging will take care of that.

Then, click on the `Save` button.

In the Branding section, you can upload an icon for your app. This icon will be used by Teams users to identify your bot.

Note: If the Microsoft Teams App will be served by Universal Messaging, You do not have to fill in the Branding section. Universal Messaging will take care of that.

In the App Feature section, select and add the feature "Bot".

In the Bot form, click on "create a new bot" and in the next screen click on "New Bot" button in the toolbar and give it a name. Once it is created, which can take a few seconds, fill in the Endpoint address with the webhook URL given by your Genesys representative. This is the URL that will be used by Microsoft Teams to send messages to Universal Messaging:

Note: Microsoft Teams does not verify the webhook URL. Which means you can create the bot before configuring Universal Messaging.

Still in the Bot section, go to the "Client secrets" section and click on "Add a client secret for your bot". This will generate a secret that you will need to configure the bot in Universal Messaging.

Note: The secret is only shown once. If you lose it, you will need to create a new one. Save it in a safe place. You will need to send it to your Genesys representative.
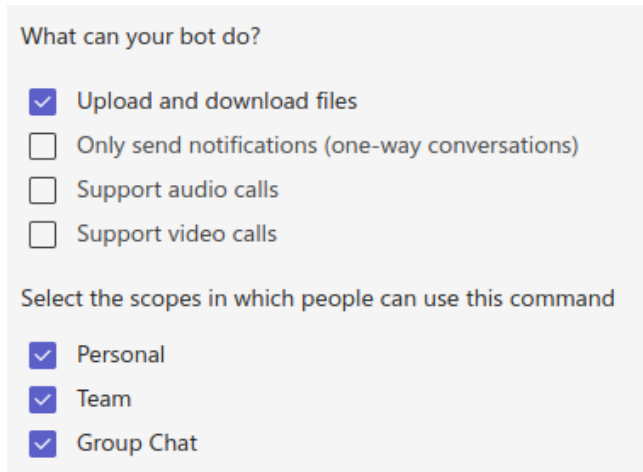
Go back to the Teams App page, section "App features", and add the Bot by its name and ID. Remember the ID of the bot as you will

need it to configure the bot in Universal Messaging.

Check "Upload and download files" in the question "What can your bot do?". And select the scopes you want to give to the bot. Typically, "Personal", "Team", and "Group Chat".

In the "Permissions section", add some "Team Permissions":

- ChannelMessage.Read.Group (Application level)

And some "Chat/Meeting Permissions":

- ChatMessage.Read.Chat (Application level)

Then, click on the `Save` button.

Finally, go to the Publish/Publish to org section and click on the `Publish your app` button. (**Note:** If the Microsoft Teams App will be served by Universal Messaging, You do not need to do that.)

Go back to the Apps page, click on the `...` button of your app and select `Download app package`. Send that file to your Genesys representative along with the Bot Secret value you got earlier.

Slack

To configure Slack, you need to create a new Slack App. You can do that by going to the Slack App Directory and clicking on the
`Create New App` button:
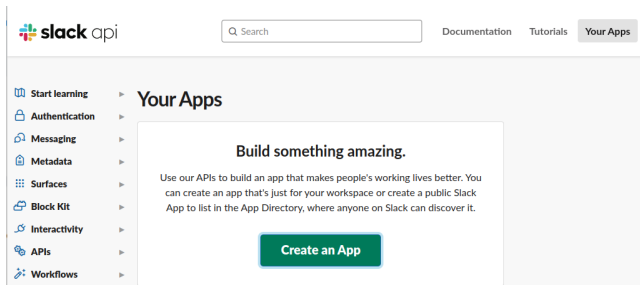
And create the app from an app manifest. Pick a workspace to create the app in:
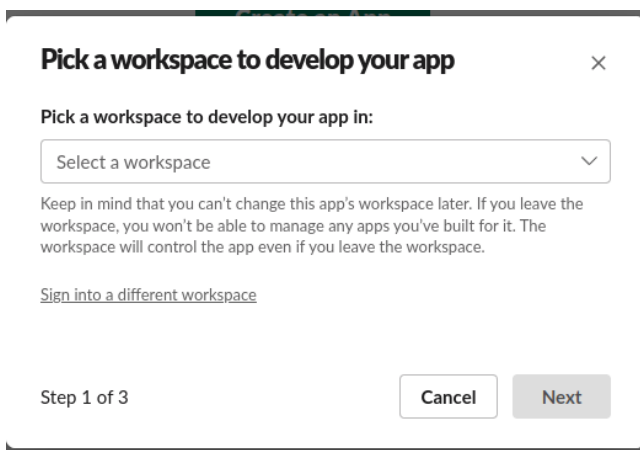
Get the manifest from the Universal Messaging Config website:
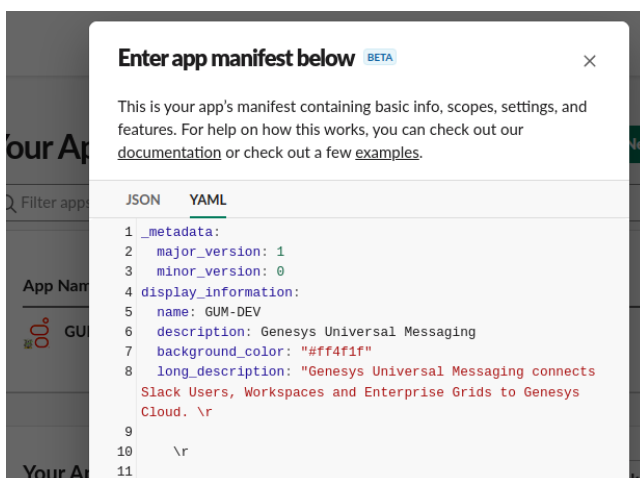
Then replace the template by pasting your manifest in the dialog box. Review the changes and click on `Create` .

Once created, Go to the Universal Messaging Config website and add a new Tenant with Slack:

Copy the App ID, the Client ID, the Client Secret, and the Signing Secret from the Slack `Basic Information` page.

Once the Tenant is saved, you can go to the `Event Subscriptions` page and hit the `Retry` button to validate the Request URL.

You can also modify the App Display information on the `Basic Information` and `App Home` pages.

Give the Tenant ID to the customer calling it a `Customer Identifier`. Then, have them register their workspaces by going to https:///slack/register.

Once they approve the App in their workspace, they will be able to start conversations with CX Agents.

Telegram

To connect to Telegram, you create a new Bot with Telegram's BotFather.

First, you need to *talk* to the BotFather, open the search and type `@BotFather` :
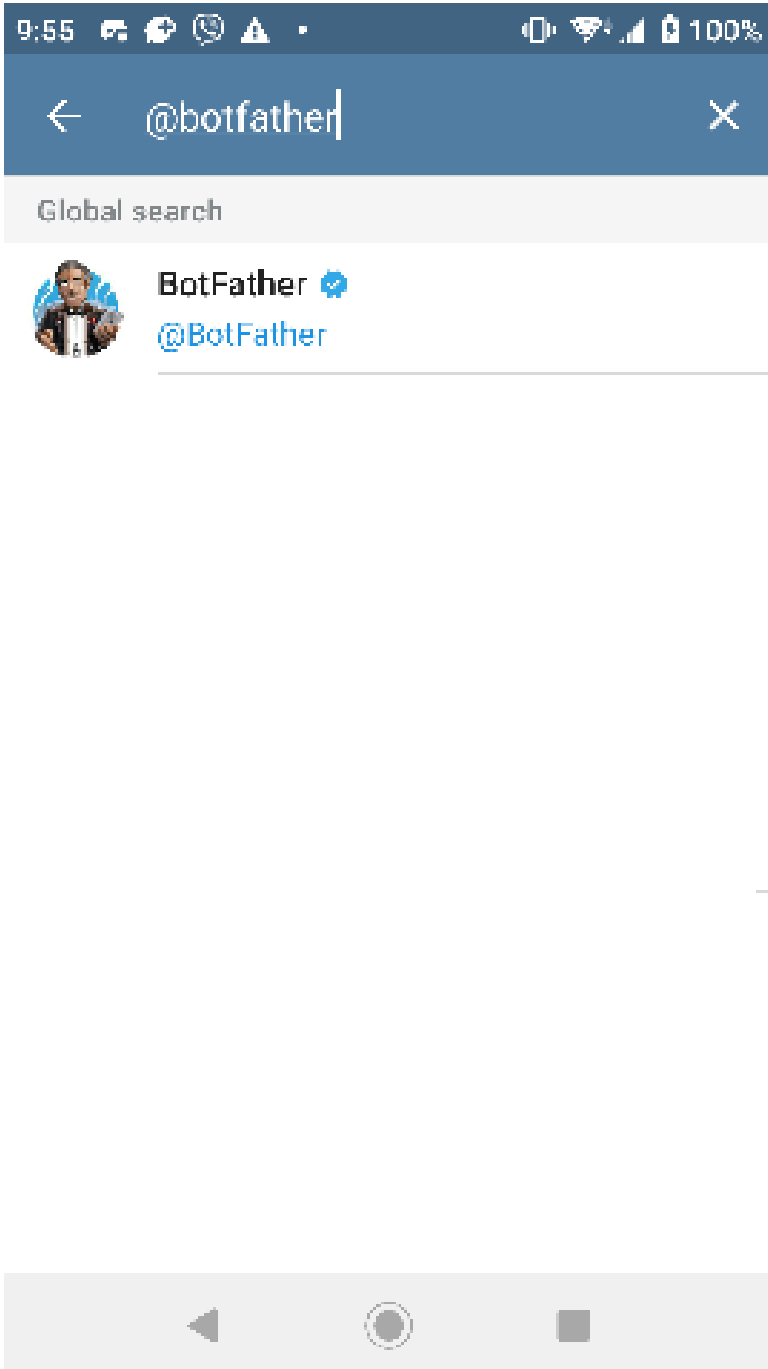
Once connected, start a new conversation (hit the `START` button) and create your new Bot by simply typing:

```
/newbot
```

This starts a dialog where you need to enter the name of your new Bot and its username. At the end of the dialog, the BotFather gives you an API Token:
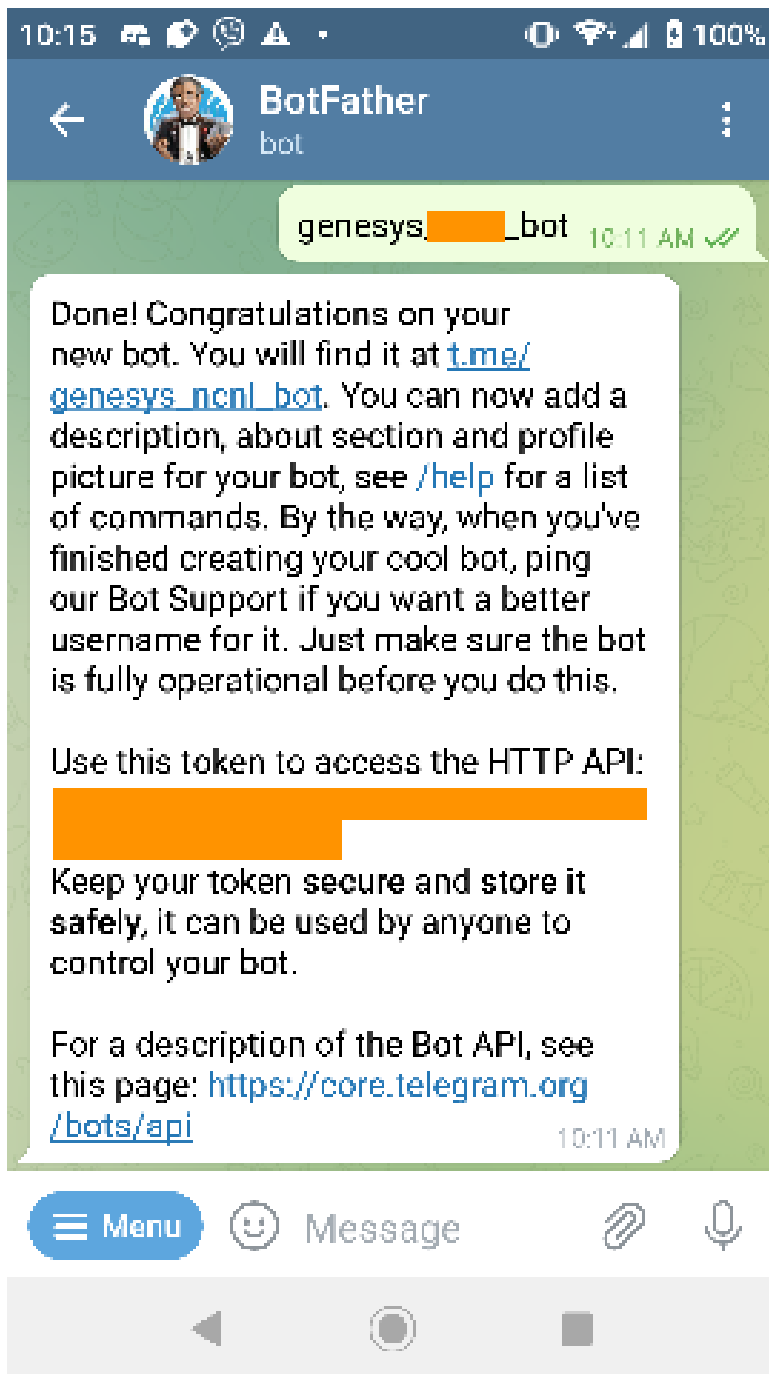
Copy that API Token and enter it in the Universal Messaging Config:

Guest users will need to start a conversation with the Bot by mentioning its username prepended with a `@` .

Viber

First create a Viber Partner Account and create a Viber Chatbot.

Then, log on your account at http://partners.viber.com:
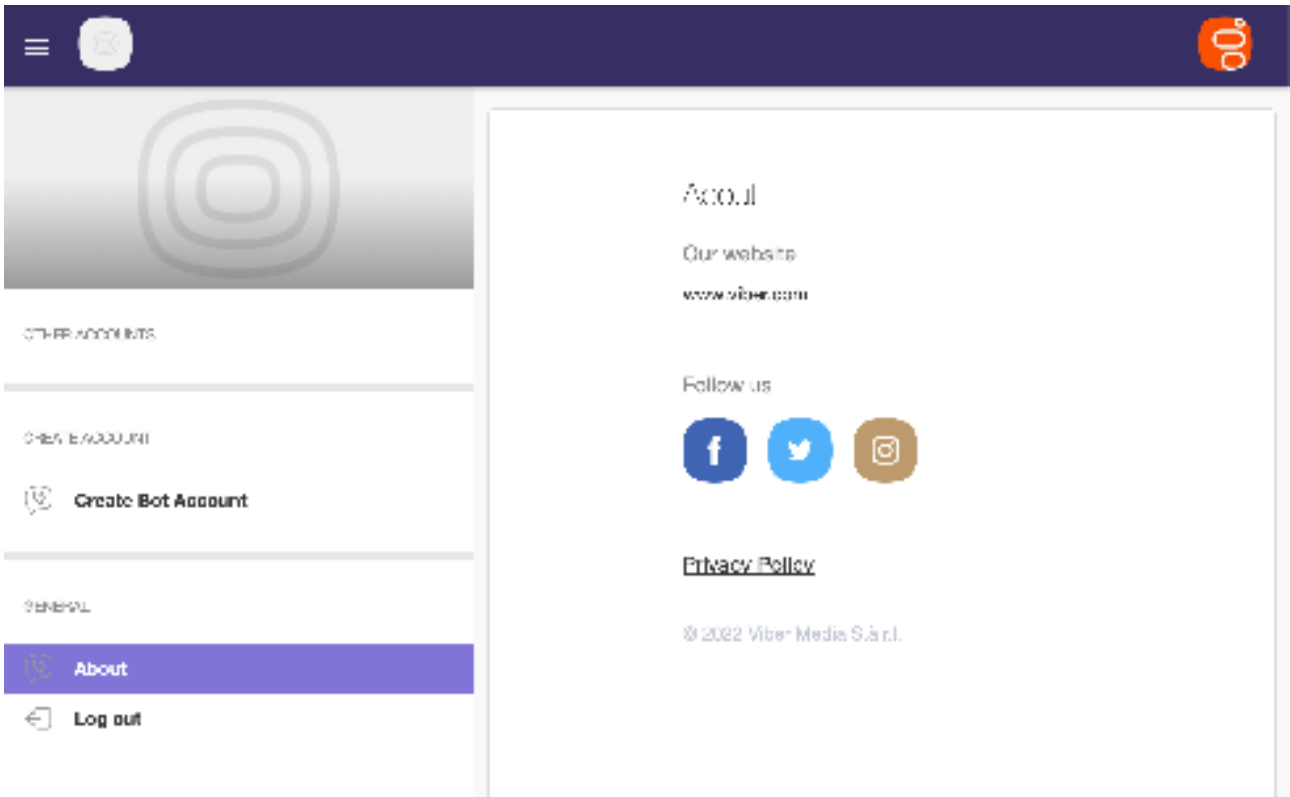
And Create a Bot Account.
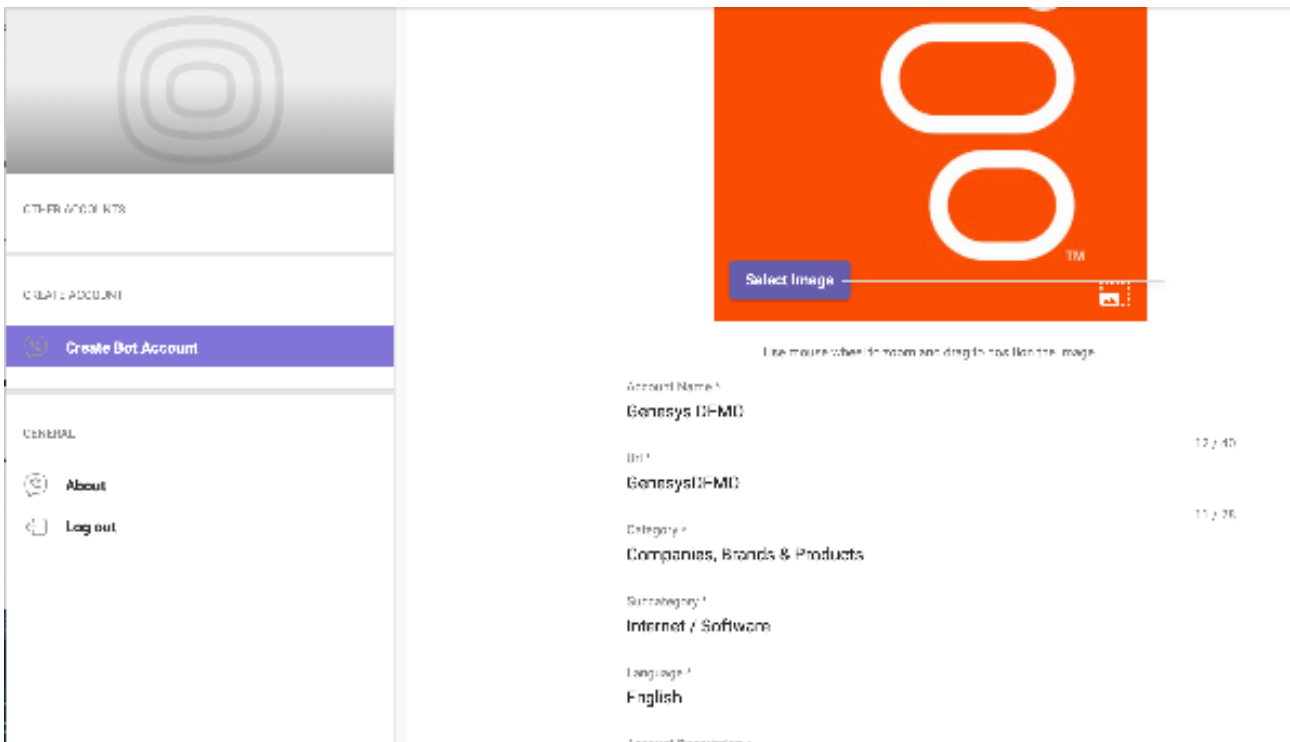
Upon creation, you will receive a Chat API Token that you will need to enter in the Universal Messaging Config:

Zalo

To connect to Zalo, the Zalo Official Account Administrator needs to connect to the Zalo console and create a new Application.

Once the application is created, the administror needs to register its usage with the Zalo Official Account API.
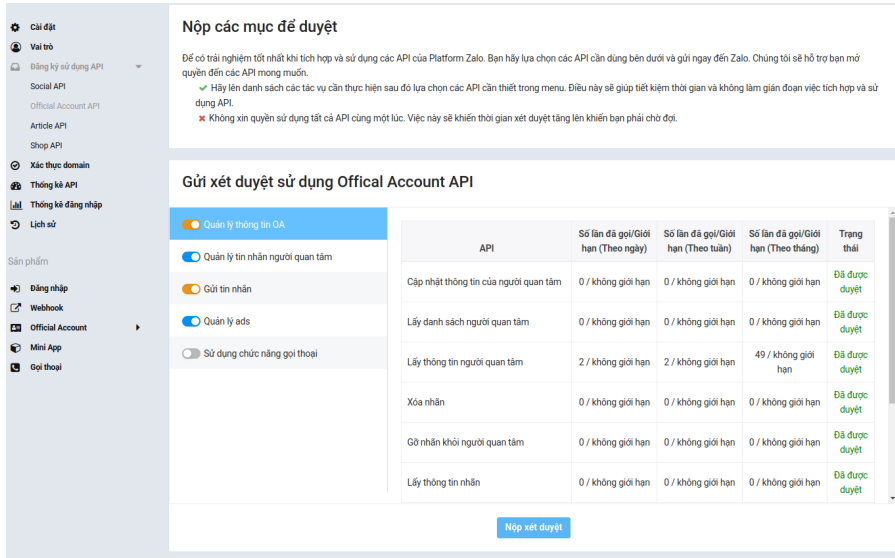
Then the administrator needs to link the Zalo Official Account to this application (that account needs to exist already and require some legal documents to be submitted to Zalo):
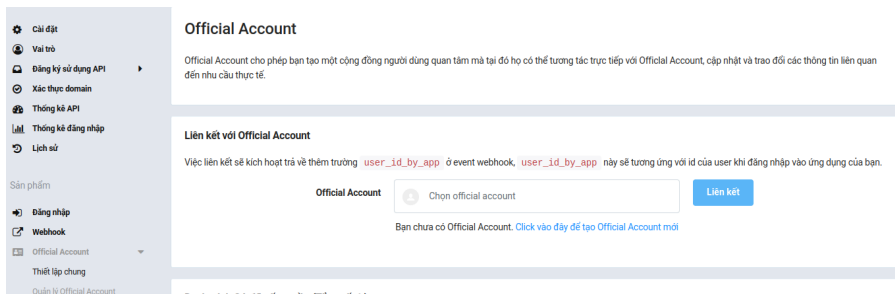
Once this is done, it is time to create a new Tenant in Universal Messaging Config. At first, the form will show a message asking to save the tenant once to get the link to request for tokens, this is normal.
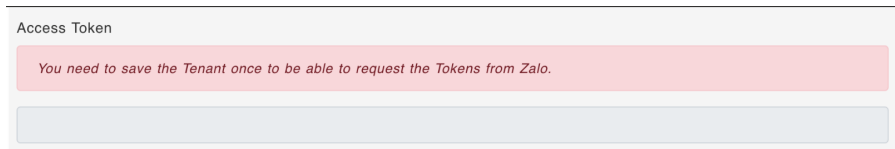
Fill in the Application Id, secret and Official Account Id, secret and save the tenant. Refresh the page and you will see the link to request for tokens:
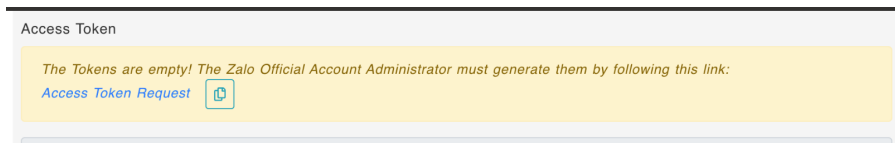
Next, the administrator needs to verify the domain used by Universal Messaging by entering the webhook from Universal Messaging Config in the `Webhook` field.

The domain verification string, which is the content of the meta tag `zalo-platform-site-verification`, will be needed in the Universal Messaging configuration.

Once the domain is verified, we can send the authorization link to the Zalo Official Account administrator for validation, once they click and After a few seconds, you can refresh the page again and you will see the tokens.

Follow the instructions about configuring the webhook in the webhook section of the same page. The Zalo Official Account admnistator needs to enter the webhook from Universal Messaging Config in the `Webhook` page on Zalo and give the listed permissions.

Chapter 4

Canned Responses

Canned Responses are used in Universal Messaging to send predefined messages to social media users. These messages can be simple text or complex JSON objects that are either common to Genesys Cloud or specific to each social media.

Create a Canned Response

To add a canned response for Universal Messaging, you must create a canned response in Genesys Cloud, in the `Admin/Canned Responses` section:
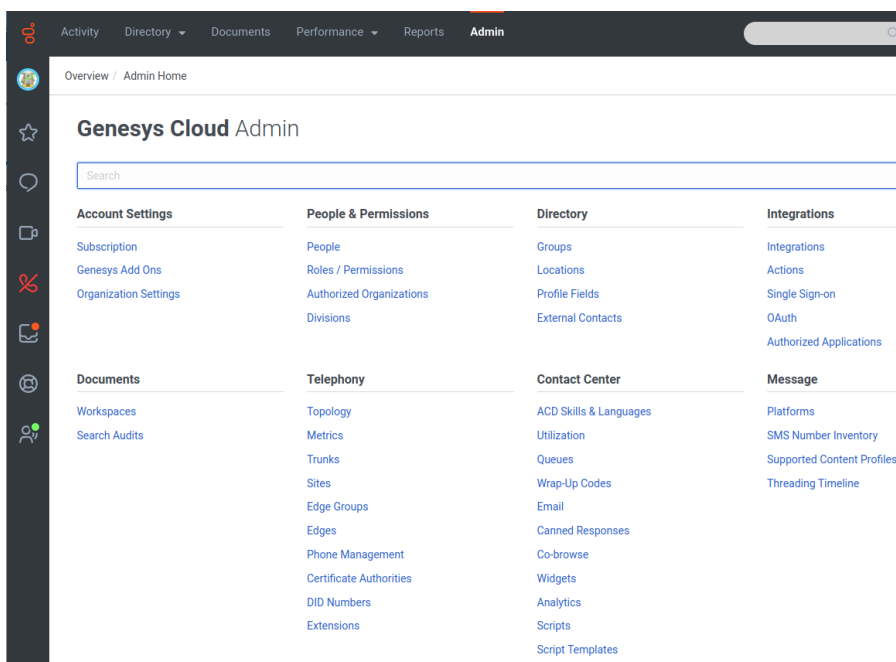
You can add or reuse a Response Library as you see fit. A Response Library is simply a collection of canned responses. When adding a Response, make sure you choose `Campaign SMS Template` as the Response Type. This is the only type supported by Universal Messaging (if you use another type, you will get an error when trying to send the message):

Send a Canned Response

Once created, you can send the canned response from a Bot or an agent by sending the following command to Universal Messaging:

```
> send "My Response"
```
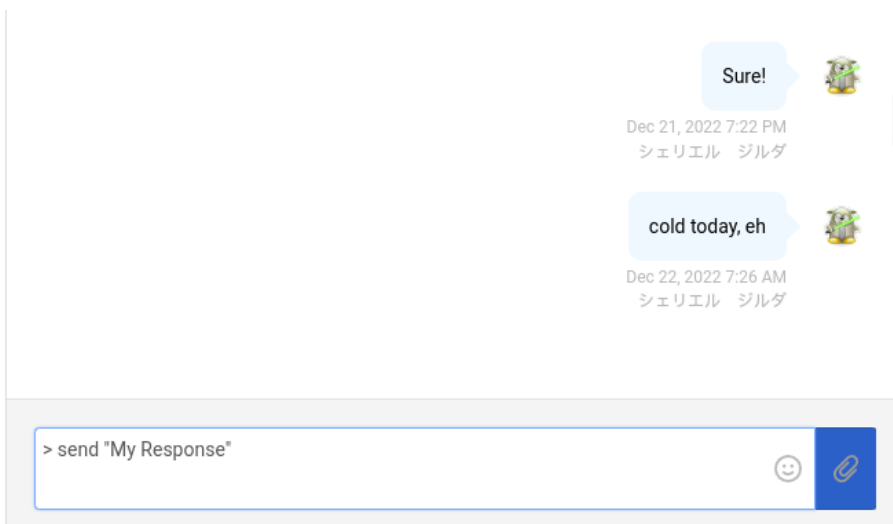

You can event create a Canned Response for sending another Canned Response, so you do not have to type '> send "My Response" every time.

At Genesys, we typically name the original Canned Response `@My Response` and save it in a Response Library called `XXX (repository)` and then create a new Canned Response called `My Response` in a Response Library called `XXX`.

As shown here:
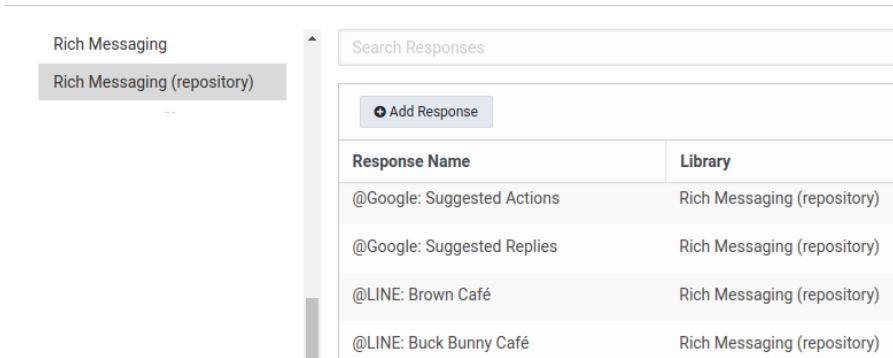
In `My Response`, we simply add the following text:

```
> send "@My Response"
```

Complex Canned Responses

Sending a complex object (Cards, Carrousels, etc) will be done similarly, as follows:

Response Name

@Apple: Quick Reply

Library

Rich Messaging (repository)

Response Type

Standard   Message Template   **Campaign SMS Template**   Ca

```json
{
  "data": {
    "quick-reply": {
      "summaryText": "What can I help you with?",
      "items": [
        {
          "identifier": "1",
          "title": "Watch our video"
        },
        {
          "identifier": "2",
          "title": "Ask for office hours"
        },
        {
          "identifier": "3",
          "title": "Find more about pet adoption"
        },
        {
          "identifier": "4",
          "title": "Talk to an agent"
        }
      ]
    }
  }
}
```

Social Media have different ways of encoding the JSON object. For example, the Quick Reply is encoded by Apple Messages for Business as follows:

```json
{
  "data": {
    "quick-reply": {
      "summaryText": "What can I help you with?",
      "items": [
        {
          "identifier": "1",
          "title": "Watch our video"
        },
        {
          "identifier": "2",
          "title": "Ask for office hours"
        },
        {
```

```json
            "identifier": "3",
            "title": "Find more about pet adoption"
        },
        {
            "identifier": "4",
            "title": "Talk to an agent"
        }
      ]
    }
  }
}
```

While Google Business Messaging would be like:

```json
{
  "text": "Hello, world!",
  "fallback": "Hello, world!\n\nReply with \"Hello\" or \"Hi!\"",
  "suggestions": [
    {
      "reply": {
        "text": "Hello",
        "postbackData": "hello-formal"
      }
    },
    {
      "reply": {
        "text": "Hi!",
        "postbackData": "hello-informal"
      }
    }
  ]
}
```

General information about the JSON objects can be found here:

- Genesys
  Work with Quick Replies in Bot Conversations
  Work with Cards in Bot Conversations (upcoming)
  Work with Carousels in Bot Conversations (upcoming)
  Universal Messages will transform the Genesys objects into the appropriate JSON object for each social media.

- Apple Messages for Business
  Rich Link Messages
  Interactive Messages
  Notes: You typically need the `data` object in the Canned Response (as shown earlier). Universal Messaging will add the `bid`, `destinationId`, `id`, `requestIdentifier`, `sourceId`, `version` automatically.

- Google Chat for Workspaces
  Cards
  Google has a Card builder to create the JSON object: https://goo.gle/card-builder
  Once built in their editor, simply copy and paste the JSON in the Canned Response.
  Universal Messages will bundle the card in a `cardsV2` array and give it a `cardId`.
  To start a Dialog from a Card, you should store the Dialog in a Canned Response, and have the card referencing it:

```json
"buttons": [
  {
    "text": "Edit",
    "onClick": {
      "action": {
        "interaction": "OPEN_DIALOG",
        "function": "@GChat: Edit Contact Dialog", // the name of the Canned Response
        "parameters": [
          {
```

```
            "key": "type",
            "value": "cannedResponse" // Tells Universal Messaging to fetch
                                      // a Canned Response
          },
          {
            "key": "id",
            "value": "sasha"
          }
        ]
      }
    }
  },
  ...
]
```

- Google Business Messages
  Send Messages
  Message Reference API
  Universal Messages will automatically add the `messageId` for you.

- LINE
  Flex Messages
  While the other formats are also supported, Flex Messages are the most powerful and flexible. We strongly encourage you to use them over their older formats (they are supported by Universal Messages anyway) LINE provides a Flex Message Builder to create the JSON object: https://developers.line.biz/flex-simulator. **Note**: Once the message is built, get the JSON object (via the `View as JSON` button) and embed it with `{ "type": "flex", "altText": "title", "contents": ... }` before adding it to the Canned Response (See: Call the Messaging API to send a Flex Message).

- Microsoft Teams
  Microsoft Adaptive Cards
  Microsoft also provides an Adaptive Card Designer to create the JSON object: https://adaptivecards.io/designer

- Slack
  Composing Layouts
  Slack also provides a Card builder, called the Block Kit Builder: https://app.slack.com/block-kit-builder

- Telegram
  Telegram Buttons are used via Genesys Cloud Quick Replies.

- Viber
  Viber Carousels
  Keyboards
  Stickers
  Universal Messages will add the `receiver`, `sender`, and `min_api_version` fields automatically.

- Zalo
  Gửi thông báo theo mẫu đính kèm danh sách
  Gửi thông báo kèm Sticker
  Cấu trúc của tham số buttons

Have a look at the Samples section for more details.

Templated Canned Responses

It is also possible to use placeholders and complex expressions in Canned Responses. The placeholder can be configured as you would in a Genesys Cloud SMS Campaign, also called "Substitutions". Here is a sample:

```
{
  "genesys_prompt": "{{question}}",
  "genesys_quick_replies": [{
{{- if .OK_payload}}
    "text": "{{OK}}","payload": "{{OK_payload}}"
{{- else}}
    "text": "{{OK}}","payload": "answer={{OK}}"
{{- end}}
  },{
{{- if .Cancel_payload}}
    "text": "{{Cancel}}","payload": "{{Cancel_payload}}"
{{- else}}
    "text": "{{Cancel}}","payload": "answer={{Cancel}}"
{{- end}}
  }]
}
```

Insert Substitution ⑦		☰
C Cancel	☑	🗑
C Cancel_payload	☑	🗑
O OK	☑	🗑
O OK_payload	☑	🗑
q question	☑	🗑
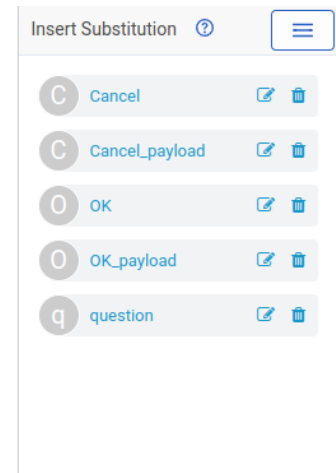
You can see the substitution `question` is mentioned in the JSON as `{{question}}` . You can set its default value in Genesys Cloud:

Name

question

Description

The question to ask the customer

Default Value

would you like to proceed?

[Done]   Cancel

The Go Template syntax can also be used to create more complex expressions. Here is an example:

```
{
  "name": "{{name}}",                                 // GC placeholder
{{ if gt .total 10}}                                  // Go Template
  "text": "{{.name}}, You are eligible for a discount", // Go Template Placeholder
{{ else }}
  "text": "Next time maybe",
{{ end }}
  "type": "Button"
}
```

You can also use the Sprig Template Library:

```
{
  "text": "{{default "Hello" .name}}"
}
```

Then, from a Bot or an agent you can call the Canned Response with parameters:

> send "cannedresponse" with name="John" total="12"

If you do not provide a value for a placeholder, the default value will be used, from Genesys Cloud.

Samples

Here we will show a non-exhaustive list of samples from social media documentations and how they should look like in a Canned Response.

In these samples, when the line is too long for the documentation, we cut it with the ↵ character. You should not cut them when you create your Canned Responses.

Genesys

Location (does not work with all social media):

```
{
  "type": "location",
  "title": "Genesys HQ",
  "address": "2001 Junipero Serra Boulevard Daly City, California 94014, United States",
  "latitude":  37.7011118,
  "longitude": -122.4719997
}
```

Quick Replies (using templates):

```
{
  "genesys_prompt": "{{question}}",
  "genesys_quick_replies": [{
{{- if .OK_payload}}
    "text": "{{OK}}","payload": "{{OK_payload}}"
{{- else}}
    "text": "{{OK}}","payload": "answer={{OK}}"
{{- end}}
  },{
{{- if .Cancel_payload}}
    "text": "{{Cancel}}","payload": "{{Cancel_payload}}"
{{- else}}
    "text": "{{Cancel}}","payload": "answer={{Cancel}}"
{{- end}}
  }]
}
```

Stickers (the sticker Identifier must mean something on the target Social Media):

```
{
  "type": "sticker",
  "title": "Hug",
  "stickerId": 40130
}
```

This is a "Hug" sticker in Viber, for example.

Apple Messages for Business

Payment Request (Note: For this to work, you need to set up a Payment Provider with Apple, Genesys cannot provide this for you, and send its configuration to your Genesys representative):

```
{
  "data": {
    "images": [
      {
       "url": "https://picsum.photos/250",
        "identifier": "1"
      }
    ],
    "payment": {
      "paymentRequest": {
        "countryCode": "US",
        "currencyCode": "USD",
        "lineItems": [
          {
            "amount": "1.00",
            "label": "Stickers",
            "type": "final"
          },
          {
            "amount": "0.50",
            "label": "Shipping",
            "type": "final"
          }
        ],
        "requiredBillingContactFields": [
        ],
        "requiredShippingContactFields": [
        ],
        "shippingMethods": [
          {
            "amount": "0.00",
            "detail": "Available within an hour",
            "identifier": "in_store_pickup",
            "label": "In-Store Pickup"
          },
          {
            "amount": "0.50",
            "detail": "5-8 Business Days",
            "identifier": "flat_rate_shipping_id_2",
            "label": "UPS Ground"
          },
          {
            "amount": "29.99",
            "detail": "1-3 Business Days",
            "identifier": "flat_rate_shipping_id_1",
            "label": "FedEx Priority Mail"
          }
        ],
        "total": {
          "amount": "1.50",
          "label": "Genesys Test",
          "type": "final"
        }
      }
    }
  },
  "receivedMessage": {
    "imageIdentifier": "1",
    "style": "large",
    "subtitle": "$1.50 at Genesys Test",
```

```json
      "title": "Stickers"
    }
}
```

Authentication Request (Note: for this to work, you need to set up an Authentication Provider with Apple, Genesys cannot provide this for you, and send its configuration to your Genesys representative):

```json
{
  "data": {
    "images": [
      {
        "url": "https://avatars.githubusercontent.com/u/2824157?s=200&v=4",
        "identifier": "1"
      }
    ],
    "authenticate": {}
  },
  "receivedMessage": {
    "imageIdentifier": "1",
    "style": "large",
    "title": "Sign in to Auth0"
  },
  "replyMessage": {
    "title": "You are now signed in"
  }
}
```

List Picker:

```json
{
  "data": {
    "listPicker": {
      "sections": [
        {
          "items": [
            {
              "identifier": "1",
              "order": 0,
              "style": "default",
              "subtitle": "Red and delicious",
              "title": "Apple"
            },
            {
              "identifier": "2",
              "order": 1,
              "style": "default",
              "subtitle": "Vitamin C boost",
              "title": "Orange"
            }
          ],
          "order": 0,
          "title": "Fruit",
          "multipleSelection": true
        },
        {
          "items": [
            {
              "identifier": "3",
              "order": 0,
              "style": "default",
              "subtitle": "Crispy greens",
              "title": "Lettuce"
```

```
            },
            {
              "identifier": "4",
              "order": 1,
              "style": "default",
              "subtitle": "Not just for your eye lids",
              "title": "Cucumber"
            }
          ],
          "order": 1,
          "title": "Veggies"
        }
      ]
    }
  }
}
```

Rich Link:

```
{
    "body": "https://www.apple.com/ipad-pro/",
    "type": "richLink",
    "richLinkData": {
        "url": "https://www.apple.com/ipad-pro/",
        "title": "iPad Pro",
        "assets": {
            "image": {
              "url": ↵
"https://support.apple.com/library/content/dam/edam/applecare/images/en_US/↵
ipad/ipadpro/2021-ipad-pro-12-9-colors.png",
              "mimeType": "image/jpeg"
            }
        }
    }
}
```

Time Picker:

```
{
  "data": {
    "event": {
      "identifier": "1",
      "title": "Please Pick a date for your dentist appointment",
      "location": {
        "title": "Genesys Japan",
        "latitude": 35.6650161,
        "longitude": 139.7432184,
        "radius": 17
      },
      "timezoneOffset": 5400,
      "timeslots": [
        {
          "identifier": "0",
          "startTime": "2022-07-10T17:00+0000",
          "duration": 3600
        },
        {
          "identifier": "1",
          "startTime": "2022-07-31T15:00+0000",
          "duration": 3600
        },
        {
```

```
          "identifier": "2",
          "startTime": "2022-08-12T14:00+0000",
          "duration": 3600
        }
      ]
    }
  },
  "receivedMessage": {
    "imageIdentifier": "0",
    "style": "icon",
    "title": "Please pick a time",
    "subtitle": "This should easy"
  },
  "replyMessage": {
    "style": "icon",
    "title": "Thank you!"
  }
}
```

It is also possible to use third-party applications. In that case you will need to fill in the `bid` of the application, for example. Due to the complexity of these, it would be better for you to be in contact your Genesys representative:

```
{
  "bid":       "com.apple.messages.MSMessageExtensionBalloonPlugin:U9S8EQXSA2: ↵
com.example.apple-samplecode.PackageDeliveryU9S8EQXSA2.MessagesExtension",
  "appId":     "123456789",
  "appName": "Package Delivery",
  "URL":       "?name=WWDC%20Goodies&deliveryDate=09-06-2017& ↵
destinationName=Moscone%20Convention%20Center&street=747%20Howard%20St& ↵
state=CA&city=San%20Fransisco&country=USA&postalCode=94103& ↵
latitude=37%2E7831&longitude=%2D122%2E4041&extraCharge=15%2E00",
  "receivedMessage": {
    "title":     "WWDC Goodies",
    "subtitle": "Scheduled for delivery - Today"
  },
  "useLiveLayout": true
}
```

Google Chat for Workspaces

Card:

```
{
  "header": {
    "title": "Sasha",
    "subtitle": "Software Engineer",
    "imageUrl": "https://developers.google.com/chat/images/quickstart-app-avatar.png",
    "imageType": "CIRCLE",
    "imageAltText": "Avatar for Sasha"
  },
  "sections": [
    {
      "header": "Contact Info",
      "collapsible": true,
      "uncollapsibleWidgetsCount": 1,
      "widgets": [
        {
          "decoratedText": {
            "startIcon": {
              "imageType": "CIRCLE",
              "knownIcon": "EMAIL"
```

```
        },
        "text": "sasha@example.com"
      }
    },
    {
      "decoratedText": {
        "startIcon": {
          "imageType": "CIRCLE",
          "knownIcon": "PERSON"
        },
        "text": "<font color=\"#80e27e\">Online</font>"
      }
    },
    {
      "decoratedText": {
        "startIcon": {
          "imageType": "CIRCLE",
          "knownIcon": "PHONE"
        },
        "text": "+1 (555) 555-1234"
      }
    },
    {
      "buttonList": {
        "buttons": [
          {
            "text": "Share",
            "onClick": {
              "openLink": {
                "url": "https://example.com/share"
              }
            }
          },
          {
            "text": "Edit",
            "onClick": {
              "action": {
                "interaction": "OPEN_DIALOG",
                "function": "@GChat: Edit Contact Dialog",
                "parameters": [
                  {
                    "key": "type",
                    "value": "cannedResponse"
                  },
                  {
                    "key": "id",
                    "value": "sasha"
                  }
                ]
              }
            }
          },
          {
            "text": "Print",
            "onClick": {
              "action": {
                "function": "print",
                "parameters": [
                  {
                    "key": "type",
```

```
                              "value": "PDF"
                        }
                    ]
                }
              }
            }
          ]
        }
      }
    ]
  }
]
}
```

Dialog (initiated by the previous card):

```
{
  "sections": [
    {
      "header": "Edit Contact",
      "widgets": [
        {
          "textInput": {
            "type": "SINGLE_LINE",
            "name": "name",
            "label": "Name"
          }
        },
        {
          "textInput": {
            "type": "MULTIPLE_LINE",
            "name": "address",
            "label": "Address"
          }
        },
        {
          "dateTimePicker": {
            "type": "DATE_AND_TIME",
            "name": "datetime",
            "label": "Pick a date and time"
          }
        },
        {
          "decoratedText": {
            "text": "Add to favorites",
            "switchControl": {
              "controlType": "SWITCH",
              "name": "saveFavorite"
            }
          }
        },
        {
          "decoratedText": {
            "text": "Merge with existing contacts",
            "switchControl": {
              "controlType": "SWITCH",
              "name": "mergeContact",
              "selected": true
            }
          }
        },
```

```
        {
          "selectionInput": {
            "type": "RADIO_BUTTON",
            "label": "Contact type",
            "name": "contactType",
            "items": [
              {
                "text": "Work",
                "value": "Work",
                "selected": false
              },
              {
                "text": "Personal",
                "value": "Personal",
                "selected": false
              }
            ]
          }
        },
        {
          "buttonList": {
            "buttons": [
              {
                "text": "Submit",
                "color": {
                  "red": 0.4,
                  "green": 0.717,
                  "blue": 0.227,
                  "alpha": 0
                },
                "onClick": {
                  "action": {
                    "function": "saveContact"
                  }
                }
              }
            ]
          }
        }
      ]
    }
  ]
}
```

Google Business Messages

Suggested Replies:

```
{
  "text": "Hello, world!",
  "fallback": "Hello, world!\n\nReply with \"Hello\" or \"Hi!\"",
  "suggestions": [
    {
      "reply": {
        "text": "Hello",
        "postbackData": "hello-formal"
      }
    },
    {
      "reply": {
        "text": "Hi!",
```

```
                "postbackData": "hello-informal"
            }
        }
    ]
}
```

Card:

```
{
  "fallback": "Hello, world!\nReply with \"Suggestion #1\" or \"Suggestion #2\"",
  "richCard": {
    "standaloneCard": {
      "cardContent": {
        "title": "Hello, world!",
        "description": "Sent with Business Messages.",
        "media": {
          "height": "TALL",
          "contentInfo":{
            "altText": "Google logo",
            "fileUrl": ↵
"https://www.google.com/images/branding/googlelogo/2x/googlelogo_color_272x92dp.png",
            "forceRefresh": false
          }
        },
        "suggestions": [
          {
            "reply": {
              "text": "Suggestion #1",
              "postbackData": "suggestion_1"
            }
          },
          {
            "action": {
              "text": "Suggestion #2",
              "postbackData": "suggestion_2",
              "openUrlAction": {
                "url": "https://www.google.com"
              }
            }
          },
          {
            "action": {
              "text": "Suggestion #3",
              "postbackData": "suggestion_3",
              "dialAction": {
                "phoneNumber": "+13178723000"
              }
            }
          }
        ]
      }
    }
  }
}
```

Carousel:

```
{
  "fallback": "Card #1\nDescription for #1\n\nCard #2\nDescription for #2\n\n↵
Reply with \"Card #1\" or \"Card #2\"",
  "richCard": {
    "carouselCard": {
```

```
      "cardWidth": "MEDIUM",
      "cardContents": [
        {
          "title": "Card #1",
          "description": "The description for card #1",
          "suggestions": [
            {
              "reply": {
                "text": "Card #1",
                "postbackData": "card_1"
              }
            }
          ],
          "media": {
            "height": "MEDIUM",
            "contentInfo": {
              "fileUrl": "https://storage.googleapis.com/kitchen-sink-sample-images/cute-dog.jpg",
              "forceRefresh": false
            }
          }
        },
        {
          "title": "Card #2",
          "description": "The description for card #2",
          "suggestions": [
            {
              "reply": {
                "text": "Card #2",
                "postbackData": "card_2"
              }
            }
          ],
          "media": {
            "height": "MEDIUM",
            "contentInfo": {
              "fileUrl": "https://storage.googleapis.com/kitchen-sink-sample-images/elephant.jpg",
              "forceRefresh": false
            }
          }
        }
      ]
    }
  }
}
```

LINE

Flex Message:

```
{
  "type": "flex",
  "altText": "flex message",
  "contents": {
    "type": "bubble",
    "hero": {
      "type": "image",
      "url": "https://scdn.line-apps.com/n/channel_devcenter/img/fx/01_1_cafe.png",
      "size": "full",
      "aspectRatio": "20:13",
      "aspectMode": "cover",
      "action": {
```

```
      "type": "uri",
      "uri": "http://linecorp.com/"
    }
  },
  "body": {
    "type": "box",
    "layout": "vertical",
    "contents": [
      {
        "type": "text",
        "text": "Brown Cafe",
        "weight": "bold",
        "size": "xl"
      },
      {
        "type": "box",
        "layout": "baseline",
        "margin": "md",
        "contents": [
          {
            "type": "icon",
            "size": "sm",
            "url": "https://scdn.line-apps.com/n/channel_devcenter/img/fx/review_gold_star_28.png"
          },
          {
            "type": "icon",
            "size": "sm",
            "url": "https://scdn.line-apps.com/n/channel_devcenter/img/fx/review_gold_star_28.png"
          },
          {
            "type": "icon",
            "size": "sm",
            "url": "https://scdn.line-apps.com/n/channel_devcenter/img/fx/review_gold_star_28.png"
          },
          {
            "type": "icon",
            "size": "sm",
            "url": "https://scdn.line-apps.com/n/channel_devcenter/img/fx/review_gold_star_28.png"
          },
          {
            "type": "icon",
            "size": "sm",
            "url": "https://scdn.line-apps.com/n/channel_devcenter/img/fx/review_gray_star_28.png"
          },
          {
            "type": "text",
            "text": "4.0",
            "size": "sm",
            "color": "#999999",
            "margin": "md",
            "flex": 0
          }
        ]
      },
      {
        "type": "box",
        "layout": "vertical",
        "margin": "lg",
        "spacing": "sm",
        "contents": [
```

```json
              {
                "type": "box",
                "layout": "baseline",
                "spacing": "sm",
                "contents": [
                  {
                    "type": "text",
                    "text": "Place",
                    "color": "#aaaaaa",
                    "size": "sm",
                    "flex": 1
                  },
                  {
                    "type": "text",
                    "text": "Miraina Tower, 4-1-6 Shinjuku, Tokyo",
                    "wrap": true,
                    "color": "#666666",
                    "size": "sm",
                    "flex": 5
                  }
                ]
              },
              {
                "type": "box",
                "layout": "baseline",
                "spacing": "sm",
                "contents": [
                  {
                    "type": "text",
                    "text": "Time",
                    "color": "#aaaaaa",
                    "size": "sm",
                    "flex": 1
                  },
                  {
                    "type": "text",
                    "text": "10:00 - 23:00",
                    "wrap": true,
                    "color": "#666666",
                    "size": "sm",
                    "flex": 5
                  }
                ]
              }
            ]
          }
        ]
      },
      "footer": {
        "type": "box",
        "layout": "vertical",
        "spacing": "sm",
        "contents": [
          {
            "type": "button",
            "style": "link",
            "height": "sm",
            "action": {
              "type": "uri",
              "label": "CALL",
```

```
          "uri": "https://linecorp.com"
        }
      },
      {
        "type": "button",
        "style": "link",
        "height": "sm",
        "action": {
          "type": "uri",
          "label": "WEB",
          "uri": "https://linecorp.com"
        }
      }
    ],
    "flex": 0
  }
 }
}
```

Microsoft Teams

Card:

```
{
  "$schema": "https://adaptivecards.io/schemas/adaptive-card.json",
  "type": "AdaptiveCard",
  "version": "1.0",
  "body": [
    {
      "type": "Container",
      "items": [
        {
          "type": "TextBlock",
          "text": "Publish Adaptive Card schema",
          "weight": "bolder",
          "size": "medium"
        },
        {
          "type": "ColumnSet",
          "columns": [
            {
              "type": "Column",
              "width": "auto",
              "items": [
                {
                  "type": "Image",
                  "url": ↵
"https://pbs.twimg.com/profile_images/3647943215/d7f12830b3c17a5a9e4afcc370e3a37e_400x400.jpeg",
                  "size": "small",
                  "style": "person"
                }
              ]
            },
            {
              "type": "Column",
              "width": "stretch",
              "items": [
                {
                  "type": "TextBlock",
                  "text": "Matt Hidinger",
                  "weight": "bolder",
```

```
                    "wrap": true
                  },
                  {
                    "type": "TextBlock",
                    "spacing": "none",
                    "text": "Created {{DATE(2017-02-14T06:08:39Z, SHORT)}}",
                    "isSubtle": true,
                    "wrap": true
                  }
                ]
              }
            ]
          }
        ]
      },
      {
        "type": "Container",
        "items": [
          {
            "type": "TextBlock",
            "text": "Now that we have defined the main rules and features of the format, ↵
we need to produce a schema and publish it to GitHub. The schema will be the starting ↵
point of our reference documentation.",
            "wrap": true
          },
          {
            "type": "FactSet",
            "facts": [
              {
                "title": "Board:",
                "value": "Adaptive Card"
              },
              {
                "title": "List:",
                "value": "Backlog"
              },
              {
                "title": "Assigned to:",
                "value": "Matt Hidinger"
              },
              {
                "title": "Due date:",
                "value": "Not set"
              }
            ]
          }
        ]
      }
    ],
    "actions": [
      {
        "type": "Action.ShowCard",
        "title": "Comment",
        "card": {
          "type": "AdaptiveCard",
          "body": [
            {
              "type": "Input.Text",
              "id": "comment",
              "isMultiline": true,
```

```
          "placeholder": "Enter your comment"
        }
      ],
      "actions": [
        {
          "type": "Action.Submit",
          "title": "OK"
        }
      ]
    }
  },
  {
    "type": "Action.OpenUrl",
    "title": "View",
    "url": "https://adaptivecards.io"
  }
  ]
}
```

Slack

Approval:

```
{
  "blocks": [
    {
      "type": "section",
      "text": {
        "type": "mrkdwn",
        "text": "You have a new request:\n*<google.com|Fred Enriquez - Time Off request>*"
      }
    },
    {
      "type": "section",
      "text": {
        "type": "mrkdwn",
        "text": "*Type:*\nPaid time off\n*When:*\nAug 10-Aug 13\n ↵
*Hours:* 16.0 (2 days)\n*Remaining balance:* 32.0 hours (4 days)\n ↵
*Comments:* \"Family in town, going camping!\""
      },
      "accessory": {
        "type": "image",
        "image_url": "https://api.slack.com/img/blocks/bkb_template_images/approvalsNewDevice.png",
        "alt_text": "computer thumbnail"
      }
    },
    {
      "type": "actions",
      "elements": [
        {
          "type": "button",
          "text": {
            "type": "plain_text",
            "emoji": true,
            "text": "Approve"
          },
          "style": "primary",
          "value": "click_me_approve"
        },
        {
          "type": "button",
```

```
        "text": {
          "type": "plain_text",
          "emoji": true,
          "text": "Deny"
        },
        "style": "danger",
        "value": "click_me_deny"
      }
    ]
  }
 ]
}
```

Notification:

```
{
  "blocks": [
    {
      "type": "section",
      "text": {
        "type": "plain_text",
        "emoji": true,
        "text": "Looks like you have a scheduling conflict with this event:"
      }
    },
    {
      "type": "divider"
    },
    {
      "type": "section",
      "text": {
        "type": "mrkdwn",
        "text": "*<fakeLink.toUserProfiles.com|Iris / Zelda 1-1>*\n ↵
Tuesday, January 21 4:00-4:30pm\nBuilding 2 - Havarti Cheese (3)\n2 guests"
      },
      "accessory": {
        "type": "image",
        "image_url": "https://api.slack.com/img/blocks/bkb_template_images/notifications.png",
        "alt_text": "calendar thumbnail"
      }
    },
    {
      "type": "context",
      "elements": [
        {
          "type": "image",
          "image_url": ↵
"https://api.slack.com/img/blocks/bkb_template_images/notificationsWarningIcon.png",
          "alt_text": "notifications warning icon"
        },
        {
          "type": "mrkdwn",
          "text": "*Conflicts with Team Huddle: 4:15-4:30pm*"
        }
      ]
    },
    {
      "type": "divider"
    },
    {
      "type": "section",
```

```
      "text": {
        "type": "mrkdwn",
        "text": "*Propose a new time:*"
      }
    },
    {
      "type": "section",
      "text": {
        "type": "mrkdwn",
        "text": "*Today - 4:30-5pm*\nEveryone is available: @iris, @zelda"
      },
      "accessory": {
        "type": "button",
        "text": {
          "type": "plain_text",
          "emoji": true,
          "text": "Choose"
        },
        "value": "click_me_today_1630"
      }
    },
    {
      "type": "section",
      "text": {
        "type": "mrkdwn",
        "text": "*Tomorrow - 4-4:30pm*\nEveryone is available: @iris, @zelda"
      },
      "accessory": {
        "type": "button",
        "text": {
          "type": "plain_text",
          "emoji": true,
          "text": "Choose"
        },
        "value": "click_me_tomorrow_1600"
      }
    },
    {
      "type": "section",
      "text": {
        "type": "mrkdwn",
        "text": "*Tomorrow - 6-6:30pm*\nSome people aren't available: @iris, ~@zelda~"
      },
      "accessory": {
        "type": "button",
        "text": {
          "type": "plain_text",
          "emoji": true,
          "text": "Choose"
        },
        "value": "click_me_tomorrow_1800"
      }
    },
    {
      "type": "section",
      "text": {
        "type": "mrkdwn",
        "text": "*<fakelink.ToMoreTimes.com|Show more times>*"
      }
    }
```

```
    ]
}
```

Viber

Sticker (Bowing Cat):

```json
{
  "type": "sticker",
  "stickerId": "bfe458bf64fa8da4d4eb"
}
```

Carousel:

```json
{
    "type":"rich_media",
    "rich_media":{
        "Type":"rich_media",
        "ButtonsGroupColumns":6,
        "ButtonsGroupRows":7,
        "BgColor":"#FFFFFF",
        "Buttons":[
            {
                "Columns":6,
                "Rows":3,
                "ActionType":"open-url",
                "ActionBody":"https://www.google.com",
                "Image": ↵
"https://image.made-in-china.com/44f3j0OoHRakwIAaLzT/↵
Custom-Logo-Glowing-Stereo-Headphones-Computer-Gaming-Headset-with-Microphone.jpg"
            },
            {
                "Columns":6,
                "Rows":2,
                "Text":"<font color=#323232><b>Headphones with Microphone, ↵
On-ear Wired earphones</b></font>↵
<font color=#777777><br>Sound Intone </font><font color=#6fc133>$17.99</font>",
                "ActionType":"open-url",
                "ActionBody":"https://www.google.com",
                "TextSize":"medium",
                "TextVAlign":"middle",
                "TextHAlign":"left"
            },
            {
                "Columns":6,
                "Rows":1,
                "ActionType":"reply",
                "ActionBody":"https://www.google.com",
                "Text":"<font color=#ffffff>Buy</font>",
                "TextSize":"large",
                "TextVAlign":"middle",
                "TextHAlign":"middle",
                "Image": ↵
"https://img.favpng.com/25/23/15/green-area-angle-png-favpng-zEacBKmPkyuj2TORzFWhaeWHx.jpg"
            },
            {
                "Columns":6,
                "Rows":1,
                "ActionType":"reply",
                "ActionBody":"https://www.google.com",
                "Text":"<font color=#8367db>MORE DETAILS</font>",
```

```
            "TextSize":"small",
            "TextVAlign":"middle",
            "TextHAlign":"middle"
        },
        {
            "Columns":6,
            "Rows":3,
            "ActionType":"open-url",
            "ActionBody":"https://www.google.com",
            "Image": ↵
"https://shoppingu.today/wp-content/uploads/2018/06/I-am-Thinking-T-shirt.jpg"
        },
        {
            "Columns":6,
            "Rows":2,
            "Text":"<font color=#323232><b>Hanes Men's Humor Graphic T-Shirt</b></font>↵
<font color=#777777><br>Hanes</font><font color=#6fc133>$10.99</font>",
            "ActionType":"open-url",
            "ActionBody":"https://www.google.com",
            "TextSize":"medium",
            "TextVAlign":"middle",
            "TextHAlign":"left"
        },
        {
            "Columns":6,
            "Rows":1,
            "ActionType":"reply",
            "ActionBody":"https://www.google.com",
            "Text":"<font color=#ffffff>Buy</font>",
            "TextSize":"large",
            "TextVAlign":"middle",
            "TextHAlign":"middle",
            "Image": ↵
"https://img.favpng.com/25/23/15/green-area-angle-png-favpng-zEacBKmPkyuj2TORzFWhaeWHx.jpg"
        },
        {
            "Columns":6,
            "Rows":1,
            "ActionType":"reply",
            "ActionBody":"https://www.google.com",
            "Text":"<font color=#8367db>MORE DETAILS</font>",
            "TextSize":"small",
            "TextVAlign":"middle",
            "TextHAlign":"middle"
        }
    ]
  }
}
```

Zalo

List:

```
{
  "attachment": {
    "type": "template",
    "payload": {
      "template_type": "list",
      "elements": [
        {
          "title": "Official Account API",
```

```
          "subtitle": "Zalo API",
          "image_url": "https://stc-developers.zdn.vn/images/bg_1.jpg",
          "default_action": {
            "type": "oa.open.url",
            "url": "https://developers.zalo.me/docs/api/official-account-api-147"
          }
        },
        {
          "title": "Article API",
          "image_url": "https://stc-zaloprofile.zdn.vn/pc/v1/images/zalo_sharelogo.png",
          "default_action": {
            "type": "oa.open.url",
            "url": "https://developers.zalo.me/docs/api/article-api-151"
          }
        },
        {
          "title": "Social API",
          "image_url": "https://stc-zaloprofile.zdn.vn/pc/v1/images/zalo_sharelogo.png",
          "default_action": {
            "type": "oa.open.url",
            "url": "https://developers.zalo.me/docs/api/social-api-4"
          }
        },
        {
          "title": "Shop API",
          "image_url": "https://stc-zaloprofile.zdn.vn/pc/v1/images/zalo_sharelogo.png",
          "default_action": {
            "type": "oa.open.url",
            "url": "https://developers.zalo.me/docs/api/shop-api-124"
          }
        }
      ]
    }
  }
}
```

Chapter 5

Using the REST API

The Universal Messaging application is usually configured by its config website. In addition, a REST API can also be used that allow batch scripts to create storages, tenants, etc.

That REST API is available at the FQDN of your deployment under the path `/api/v1` . For example:

`https://www.acme.org/api/v1`

The REST API OpenAPI specification is available at `/api/v1/openapi` and the API can be experienced via a swagger-ui available at: `/api/v1/openapi-ui` .

Note: To make sure the `/api/v1/openapi-ui` points to the Universal Messaging API, make sure the `api_root` property was set to the proper FQDN during the Helm deployment:

```
api:
  admin:
    password: s1ms3cr3t
  api_root: https://www.acme.org
```

To use the API, you must first authenticate:

```
curl https://www.acme.org/api/v1/auth/login \
  -X POST \
  -H "Content-Type: application/json" \
  -d '{"user": "admin", "password": "s3cr3t"}'
```

The response will contain a token:

```
{
  "type": "Bearer",
  "token": "1234567876543wsdfghjgfvcxsq1234567ythgv"
}
```

Include the token in all future requests:

```
curl https://www.acme.org/api/v1/storages \
  -H "Authorization: bearer 1234567876543wsdfghjgfvcxsq1234567ythgv"
```

REST API for IVRs

The Universal Messaging API can also be used by IVR engines, such as Genesys Cloud CX Architect and Genesys PureCloud Interaction Attendant.

When guests call your IVR it can be desirable to send them some messages to their Social Media accounts. With the Universal Messaging API, this is now possible.

Note that you will need to ask or store the guest's Social Media identifier.

A noticeable exception is LINE. If you subscribe to their Phone Number Push option (and filled in the configuration properly in the config website), you can use the guests (fully qualified) mobile numbers directly from the IVR. Make sure to contact LINE for to be able to use that option.

To send messages, the IVR will need to use the Tenant's Authentication defined in the config website:

```
curl https://www.acme.org/api/v1/messages \
  -X POST \
  -u tenant_apiuser:tenant_apipassword \
  -H "Content-Type: application/json" \
  -d \
  '{
    "tenantId": "259bb4e2-f7e5-4b06-9a51-c528f412404b",
    "message": "Hello from the IVR!"
  }'
```

Where `123456789abcd` is the Guest's identifier with the Social Media associated with the `tenantId` given in the JSON payload.

You can also use the Tenant's name in the payload:

```
curl https://www.acme.org/api/v1/messages \
  -X POST \
  -u tenant_apiuser:tenant_apipassword \
  -H "Content-Type: application/json" \
  -d \
  '{
    "tenant": "Tenant with Social Media X and Genesys Cloud CX",
    "userId": "123456789ancd",
    "message": "Hello from the IVR!"
  }'
```

It is also possible to send more than one message to the guest:

```
curl https://www.acme.org/api/v1/messages \
  -X POST \
  -u tenant_apiuser:tenant_apipassword \
  -H "Content-Type: application/json" \
  -d \
  '{
    "tenantId": "259bb4e2-f7e5-4b06-9a51-c528f412404b",
    "userId": "123456789ancd",
    "messages": [
      "Hello from the IVR!",
      { "type": "sticker", "stickerId": "123345", "packageId": "456" }
    ]
  }'
```

Of course, in the last example, the Social Media need to support Stickers.

If the Social Media is LINE and you have subscribed to the LINE Switcher API, it is possible to also send a request to "switch" the User to another LINE Bot Destination (please consult your LINE provider about this option):

```
curl https://www.acme.org/api/v1/users/+819012345678/switch \
  -X POST \
  -u tenant_apiuser:tenant_apipassword \
  -H "Content-Type: application/json" \
  -d \
  '{
    "tenantId": "259bb4e2-f7e5-4b06-9a51-c528f412404b"
  }'
```

Chapter 6

Kubernetes Fundamentals

The best and most efficient way of deploying Kubernetes is to use a Cloud vendor. Most of the time, they will manage the Kubernetes control plane for you (they might even offer that for free), upgrade the cluster automatically, make sure it works, and even have some node-level scalability (meaning adding Kubernetes nodes automatically as needs grow).

Here we will describe how to deploy Kubernetes in the most common of the Cloud vendors as well as more manual methods.

Disclaimer: These are given only to help, make sure you read the documentation of your chosen platform for fine-tuning your Kubernetes cluster.

Google Cloud Platform (GKE)

You can use the Google Cloud Shell or install the Google Cloud SDK for your platform, then initialize it:

```
gcloud init
gcloud auth login
```

- Create a project on the Google Console,
- Add Billing to the project,
- Enable the StackDriver Logging API on the Google Console
- Enable Kubernetes on your Google Cloud console.

And create your Kubernetes cluster:

```
gcloud container clusters create messaging-cluster \
  --cluster-version=latest \
  --zone asia-northeast1-a
```

By default, 3 worker nodes are created and Role Base Access Control (RBAC) is enabled. Each node is a `n1-standard-1` (1 vCPU, 3.75 GB Memory) (See Pricing and Calculator)

If you prefer to let Google handle everything, you can create the cluster in "autopilot" mode:

```
gcloud container clusters create-auto messaging-cluster
```

You can check the status of the nodes with:

```
gcloud compute instances list
```

If you do not have `kubectl` installed already, go ahead and install it:

```
gcloud components install kubectl
```

Then download the Kubernetes configuration for `kubectl` :

```
gcloud container clusters get-credentials messaging-cluster
```

Microsoft Azure Container Service (AKS)

You can use the Azure Cloud Shell or install the Azure CLI on your machine:

- On bash:

```
brew install azure-cli
```

On Debian Linux:

```
sudo apt install azure-cli
```

On Red Hat and CentOS:

```
sudo rpm --import https://packages.microsoft.com/keys/microsoft.asc
sudo tee /etc/yum.repos.d/azure-cli.repo &>/dev/null <<EOM
[azure-cli]
name=Azure CLI
baseurl=https://packages.microsoft.com/yumrepos/azure-cli
enabled=1
gpgcheck=1
gpgkey=https://packages.microsoft.com/keys/microsoft.asc"
EOM
sudo dnf install azure-cli
```

- On Windows, via chocolatey:

```
chocolatey install azure-cli
```

- Or via PowerShell:

```
Invoke-WebRequest -Uri https://aka.ms/installazurecliwindows -OutFile .\AzureCLI.msi
Start-Process msiexec.exe -Wait -ArgumentList '/I AzureCLI.msi /quiet'
rm .\AzureCLI.msi
```

You can follow the Kubernetes walkthrough from Microsoft themselves. Here is a shortlist from their site.

First Login to your Azure account:

```
az login
```

If needed, enable Azure Container Services (AKS) and a few other things on your Azure Subscription (this can take some time!):

```
az provider register -n Microsoft.Compute
az provider register -n Microsoft.Storage
az provider register -n Microsoft.Network
az provider register -n Microsoft.ContainerService
az provider register -n Microsoft.OperationsManagement
az provider register -n Microsoft.OperationalInsights
```

If you are not sure AKS is delivered in your region, you can check what is available:

```
az aks get-versions --location 'japan east' \
  --query 'orchestrators[].orchestratorVersion'
```

If all is fine, create a resource group:

```
az group create --name genesys --location japaneast
```

Then, create your cluster:

```
az aks create --resource-group genesys \
  --name messaging-cluster \
  --node-count 3 \
  --enable-addons monitoring \
  --generate-ssh-keys
```

If you want specific Virtual Machine size, you should add a `---node-vm-size` .
If you want to load your own ssh key, use `--ssh-key-value /path/to/key.pub`

If you do not have `kubectl` installed already, go ahead and install it:

```
az aks install-cli
```

Then download the Kubernetes configuration for `kubectl` :

```
az aks get-credentials --resource-group genesys --name lis-cluster
```

Then verify you can get to the Kubernetes Cluser:

```
kubectl get nodes
```

Amazon Elastic Container Services for Kubernetes (EKS)

Install the AWS CLI on your machine then intall eksctl by following its user guide:

- On MacOS:

```
brew install awscli
brew tap weaveworks/tap
brew install weaveworks/tap/eksctl
```

- On Windows:

```
chocolatey install awscli
```

- On Linux:

```
curl -sSLO "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip"
unzip awscli-exe-linux-x86_64.zip
sudo ./aws/install
curl -sSL \
  "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" \
  | tar xz
sudo mv ./eksctl /usr/local/bin
```

Configure AWS with your Access Key ID and Secret Access Key (ask your AWS Administrator for these values):

```
aws configure
```

If you use an Multi-Factor Authentication device, before executing any aws/eksctl command, you should obtain temporary credentials:

```
function aws-auth() {
  token_code=$1
  unset AWS_ACCESS_KEY_ID AWS_SECRET_ACCESS_KEY AWS_SESSION_TOKEN
  aws_identity=$(aws sts get-caller-identity --output json)
  aws_account=$(echo "$aws_identity" | jq -r .Account)
  aws_user=$(echo "$aws_identity" | jq -r .Arn | cut -d/ -f2)
  aws_arn="arn:aws:iam::${aws_account}:mfa/${aws_user}"
  creds=$(aws sts get-session-token --serial-number "$aws_arn" --token-code $token_code --output json)
  export AWS_ACCESS_KEY_ID=$(echo "$creds"     | jq -r .Credentials.AccessKeyId)
  export AWS_SECRET_ACCESS_KEY=$(echo "$creds" | jq -r .Credentials.SecretAccessKey)
  export AWS_SESSION_TOKEN=$(echo "$creds"     | jq -r .Credentials.SessionToken)
  export AWS_TOKEN_EXPIRATION=$(echo "$creds"  | jq -r .Credentials.Expiration)
  unset aws_identity creds aws_account aws_user aws_arn
}
aws-auth 123456
```

The token code comes from your MFA Device. make sure this function belongs to your bashrc/zshrc, so it can modify the current environment (if it is a script, that will not work)

You can download this function there and its Powershell equivalent there.

Fargate

The simplest way to deploy a Kubernetes cluster is through Fargate as it manages worker nodes automatically, including scaling them up and down.

Create the cluster:

```
eksctl create cluster \
  --name   messaging-cluster \
  --region ap-northeast-1 \
  --fargate \
  --alb-ingress-access
```

After 15 to 25 minutes (for both Fargate and Managed Nodes), on average, your Kubernetes cluster is ready to use.

Finally, you can query the newly created cluster:

```
kubectl get nodes
```

We will also need to tell Fargate it should also host the Kubernetes namespace for our application:

```
eksctl create fargateprofile \
  --cluster   messaging-cluster \
  --name      fp-messaging \
  --namespace messaging
```

You can enable AWS Cloudwatch logging for the control plane, if you wish, by running this:

```
eksctl utils update-cluster-logging \
  --cluster messaging-cluster \
  --region  ap-northeast-1 \
  --enable-types all \
  --approve
```

Node: This implies some cost for the log storage. See https://eksctl.io/usage/cloudwatch-cluster-logging.

Next, we need to install the Amazon EFS CSI Driver so we can create `PersistentVolume` objects with Fargate.

If the cluster does not have an OIDC provider, yet:

```
eksctl utils associate-iam-oidc-provider \
  --cluster messaging-cluster \
  --region  ap-northeast-1 \
  --approve
```

AWS EFS CSI Driver for Persistent Volumes

Create a new IAM policy to allow the CSI driver to use the AWS API on your behalf:

```
oidc=$(aws eks describe-cluster \
  --name   messaging-cluster \
  --query  cluster.identity.oidc.issuer \
  --output text \
)
curl -sSLO \
  https://raw.githubusercontent.com/kubernetes-sigs/aws-efs-csi-driver/\
v1.3.2/docs/iam-policy-example.json
aws iam create-policy \
    --policy-name AmazonEKS_EFS_CSI_Driver_Policy \
    --policy-document file://iam-policy-example.json
```

Create an IAM role attached to that new policy:

```
policy=$(aws iam list-policies --output json | \
  jq -r '.Policies[]|select(.PolicyName=="AmazonEKS_EFS_CSI_Driver_Policy")|.Arn' \
)
eksctl create iamserviceaccount \
  --name      efs-csi-controller-sa \
  --cluster   messaging-cluster \
  --namespace kube-system \
  --region    ap-northeast-1 \
  --attach-policy-arn $policy \
```

```
  --override-existing-serviceaccounts \
  --approve
```

Deploy the CSI Driver:

```
kubectl create \
  -f https://raw.githubusercontent.com/kubernetes-sigs/aws-efs-csi-driver/\
master/deploy/kubernetes/base/csidriver.yaml
```

Create a security group and an inbound rule for NFS:

```
vpcid=$(aws eks describe-cluster \
  --name   messaging-cluster \
  --query  cluster.resourcesVpcConfig.vpcId \
  --output text \
)
range=$(aws ec2 describe-vpcs \
  --vpc-ids $vpcid \
  --query   "Vpcs[].CidrBlock"\
  --output text \
)
sec_group=$(aws ec2 create-security-group \
  --group-name  messaging-efs-security-group \
  --description "EFS Security Group for Universal Messaging" \
  --vpc-id $vpcid \
  --output text \
)
aws ec2 authorize-security-group-ingress \
  --group-id $sec_group \
  --cidr     $range \
  --protocol tcp \
  --port     2049
```

Create an Amazon EFS file system:

```
fs_id=$(aws efs create-file-system \
  --region           ap-northeast-1 \
  --performance-mode generalPurpose \
  --query  FileSystemId \
  --output text \
)
```

And a mount target for Amazon EFS for each subnet member of the Fargate Profile:

```
subnets=( $(eksctl get fargateprofiles --cluster messaging-cluster -- json |\
  jq -r '.[] | select(.name=="fp-messaging") | .subnets[]' \
) )
for subnet in ${subnets[@]}; do
  aws efs create-mount-target \
    --file-system-id  $fs_id \
    --subnet-id       $subnet \
    --security-groups $sec_group
done
```

If there is another default storage class, we first need to depromote it:

```
kubectl patch storageclass gp2 \
  --patch '{"metadata":{"annotations":{"storageclass.kubernetes.io/is-default-class":"false"}}}'
```

Now, we can create the Kubernetes StorageClass and make it the default storage class:

```
kubectl create -f - <<EOM
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
```

```
  name: efs-sc
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: efs.csi.aws.com
parameters:
  provisioningMode: efs-ap
  fileSystemId: ${fs_id}
  directoryPerms: "700"
EOM
```

When creating the config.yaml, we will need to ask the charts to create the PersistentVolume resources, since Fargate does not support
Dynamic Volume Provisioning, yet. Here we show the necessary configuration for a few micro-services:

```
global:
  storageClass:  efs-sc
  storageDriver: efs.csi.aws.com
  storageHandle: ${fs_id} # Replace this with the actual fs_id from your cluster
api:
  ...
  redis:
    volumePermissions:
      enabled: true
    master:
      persistence:
        ...
        selector:
          matchLabels:
            owner: api
        volume: { create: true }

gcloudcx-connector:
  ...
  redis:
    volumePermissions:
      enabled: true
    master:
      persistence:
        ...
        selector:
          matchLabels:
            owner: gcloudcx-connector
        volume: { create: true }

rabbitmq:
  ...
  persistence:
    ...
    selector:
      matchLabels:
        owner: rabbitmq
    volume: { create: true }
  # See: https://docs.bitnami.com/general/how-to/troubleshoot-helm-chart-issues/#permission-errors-when-enabling-pe
  volumePermissions:
    enabled: true
  # workaround for https://github.com/bitnami/bitnami-docker-rabbitmq/issues/86
  ulimitNofiles: ""
```

Deploy the ALB Ingress Controller

The next step is to deploy the AWS Load Balancer (ALB) to your new Kubernetes cluster.

Create a new IAM policy to allow ALB to use the AWS API on your behalf:

```
curl -sSLO \
  https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/\
v2.2.0/docs/install/iam_policy.json
aws iam create-policy \
    --policy-name AWSLoadBalancerControllerIAMPolicy \
    --policy-document file://iam_policy.json
rm iam_policy.json
```

If the cluster does not have an OIDC provider, yet:

```
eksctl utils associate-iam-oidc-provider \
  --cluster messaging-cluster \
  --region  ap-northeast-1 \
  --approve
```

Create an IAM role attached to that new policy:

```
policy=$(aws iam list-policies --output json | \
  jq -r '.Policies[]|select(.PolicyName=="AWSLoadBalancerControllerIAMPolicy")|.Arn' \
)
eksctl create iamserviceaccount \
  --name      aws-load-balancer-controller \
  --cluster   messaging-cluster \
  --namespace kube-system \
  --attach-policy-arn $policy \
  --override-existing-serviceaccounts \
  --approve
```

Install ALB via Helm:

```
helm repo add eks https://aws.github.io/eks-charts
helm repo update
kubectl apply -k "github.com/aws/eks-charts/stable/aws-load-balancer-controller/crds?ref=master"
vpcid=$(aws eks describe-cluster \
  --name messaging-cluster \
  --query cluster.resourcesVpcConfig.vpcId \
  --output text \
)
helm upgrade -i aws-load-balancer-controller eks/aws-load-balancer-controller \
  --namespace kube-system \
  --set clusterName=messaging-cluster \
  --set serviceAccount.create=false \
  --set serviceAccount.name=aws-load-balancer-controller \
  --set region=ap-northeast-1 \
  --set vpcId=$vpcid
kubectl rollout status deployment \
  --namespace kube-system \
  aws-load-balancer-aws-load-balancer-controller
kubectl get deployments.apps --namespace kube-system
```

The last two commands just check if ALB is deployed correctly.

When deploying the Universal Messaging application, you should configure the ingress as follows in the config.yaml:

```
ingress:
  enabled: true
  className: alb
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/target-type: ip
    alb.ingress.kubernetes.io/scheme: internet-facing
```

Also, all Kubernetes service types must be `NodePort` for ALB to route HTTP requests. In the config.yaml, configure the services as follows (here we show only the Universal Messaging API and config configurations):

```yaml
api:
  service:
    type: NodePort
    annotations:
      alb.ingress.kubernetes.io/healthcheck-port: "32000"
      alb.ingress.kubernetes.io/healthcheck-path: "/healthz/readiness"

config:
  service:
    type: NodePort
    annotations:
      alb.ingress.kubernetes.io/healthcheck-port: "3000"
      alb.ingress.kubernetes.io/healthcheck-path: "/"
      alb.ingress.kubernetes.io/success-codes: "200,302"
```

You can find more details about ALB Ingress on AWS site.

Delete the cluster

If you need to delete the cluster, run this:

```
eksctl delete cluster \
  --name   messaging-cluster \
  --region ap-northeast-1
```

Docker for Desktop

This is the simplest when you want to develop and/or learn and you run on Windows or macOs:

1. Just install Docker for Windows or Docker for Mac.
2. Turn on Kubernetes in the Docker settings.

You are done!

Check it:

```
kubectl get nodes
```

Note: Docker for Desktop's Kubernetes is fairly "naked" by default. This means there is no Ingress Controller, no Registry, no Logging stack. You have to install these by yourself, if you need them.

For example: If you want a local registry within the Kubernetes cluster. First install Helm and then deploy the registry chart:

```
helm install stable/docker-registry \
  --namespace kube-system \
  --set persistence.enabled=true \
  --set service.type=NodePort \
  --set service.nodePort=32000
```

To install an Ingress Controller, the best is to follow its installation guide:

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/\
controller-v1.0.0/deploy/static/provider/cloud/deploy.yaml
```

When deploying an ingress you will need to add the ingress class "nginx" in the annotation:

```yaml
metadata:
  annotations:
    kubernetes.io/ingress.class: nginx
```

With Universal Messaging's Helm charts, you simply need to add the class in your config.yaml:

```yaml
ingress:
  className: "nginx"
```

About the Logging stack, I have become a big fan of fluentd, Digital Ocean made an excellent documentation about deploying it.

MicroK8s

MicroK8s is a development/test Kubernetes deployment on Windows, MacOS, and Ubuntu. It has the advantage of simplicity and easiness. It is perfect for Development and Learning (not for production!)

To install MicroK8s, simply execute:

```
sudo snap install microk8s --classic
```

After a short while, you Kubernetes cluster is ready.

You will typically want to enable some services, like:

```
sudo microk8s.enable rbac dns registry ingress storage helm3
snap alias microk8s.kubectl kubectl
snap alias microk8s.helm3   helm
sudo iptables -P FORWARD ACCEPT
sudo DEBIAN_FRONTEND=noninteractive apt-get install -y iptables-persistent
```

Also check that `::1` does not resolve into `localhost` in `/etc/hosts` .

You might have to configure the DNS forwarder in `coredns` as it generally points to 8.8.8.8 and 8.8.4.4 and that might not be suitable in your environment, see https://microk8s.io/docs/addon-dns.

If `microk8s` is on the same host, you are done.

If you want to access `microk8s` remotely from your desktop, you need to install `kubectl` and `helm` locally and configure remote access.

To add the cluster configuration to your local kubectl, you can grab its config, from your machine:

```
ssh myuser@kubernetes-host microk0s.config view > ~/.kube/myk8s
```

Do not forget to properly merge `myk8s` and `~/.kube/config` if you don't want to set the environment variable `KUBECONFIG` .

To be able to push docker images to your cluster, you should also add its IP or FQDN to the insecure registries of your own docker configuration:

On Linux, you would edit `/etc/docker/daemon.json` and add:

```
{
  "insecure-registries": [ "kubernetes-host:32000" ]
}
```

Where `kubernetes-host` should be replaced by either the hostname or its IP address.

On Docker for Windows and Docker for Mac, you would edit add the registry in the Preferences dialog.

For more informations about Microk8s configuration and usage, see: https://microk8s.io/docs

Manual Deployment on Virtual Machines/Bare Metal

Sometimes, the usual Cloud vendors do not have any support for Containers in your region. In that case, the solution is to build the Kubernetes cluster manually in virtual machines or bare metal (We will use the term Virtual Machine in this paragraph, but, unless mentioned, everything applies to bare metal machines as well). While you get full control, this is really a lot more complex and it is very easy to make mistakes…

As an example, here we will build a cluster with 1 master and 3 worker nodes. You should change these numbers to your own needs.

First, get 4 virtual machines from your Cloud vendor or your own data center. I prefer going with Ubuntu 16.04 as it boasts a 4.x kernel already.
If you could not manually create partitions and have a swap, disable the swap on every node, afterwards:

```
sudo swapoff -a
sudo sed -i '/ swap /s/^/#/' /etc/fstab
```

Notes:

- Make sure the master node always gets the same IP address!!!!
- The following script will try to remove the swap if you forgot to do it.

On the Kubernetes master node, install all the software and start the cluster:

```
curl -sSL https://tinyurl.com/ubuntu-prep-k8s | bash -s
```

At the end of the script, you will get a token, the master's IP address, and a certification hash. Copy these values and use them on each worker node:

```
curl -sSL https://tinyurl.com/ubuntu-prep-k8s | bash -s -- \
  --join  123.456.78.9:6443 \
  --token 1046fd.a914436354c9c418 \
  --discovery-token-ca-cert-hash \
    sha256:af8b270b8745809dec666b8048663a087c3b9d9d6d8cbd6db62748503403f2ba
```

If you want to manage the cluster from your own machine, install `kubectl` and add the configuration from the master node:

```
scp myuser@123.456.78.9:.kube/config ~/.kube/config-mycluster
export KUBECONFIG="$HOME/.kube/config-mycluster:$HOME/.kube/config${KUBECONFIG+:$KUBECONFIG}"
kubectl config use-context mycluster
```

The name of the context can vary, check the actual values with `kubectl config get-contexts`. You should also add the export command to your `.bashrc` or `.zshenv`

Check the cluster:

```
kubectl get nodes
```

If you installed Kubernetes on Bare Metal or "Bare" Virtual Machines, you most probably will not have a Network Load-Balancer. This means all your Kubernetes `LoadBalancer` services will have their external IP set to `<pending>` without ever gaining an IP address.

The best is to install a Network Load-Balancer on your Kubernetes platform, such as MetalLB:

```
 kubectl apply \
  -f https://raw.githubusercontent.com/google/metallb/v0.5.0/manifests/metallb.yaml
```

```
 kubectl apply -f - << EOM
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
      - name: default
        protocol: layer2
        addresses:
          - x.y.z.1-x.y.z.100
EOM
```

Replace the addresse range with something that makes sense for your network.

If you do not have access to any Network Storage in your Cloud or if you run in your DataCenter, you will need to deploy a Kubernetes Storage. As an example, here we will deploy Rook over Ceph:

```
ROOK_URL=https://raw.githubusercontent.com/rook/rook/release-0.7/cluster/examples/kubernetes
kubectl apply -f $ROOK_URL/rook-operator.yaml
kubectl apply -f $ROOK_URL/rook-cluster.yaml
kubectl apply -f $ROOK_URL/rook-storageclass.yaml
kubectl apply -f $ROOK_URL/rook-object.yaml
kubectl apply -f $ROOK_URL/rook-filesystem.yaml
kubectl patch storageclass rook-block -p \
  '{"metadata":{"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

Check for the proper version of rook there: https://github.com/rook/rook/releases

Chapter 7

Getting Helm

Maintaining a plethora of YAML scripts to deploy and maintain you application is progress, but ultimately it becomes a mess. Even if you maintain them through version control (like git).

Enter the Package Manager and Release Deployment tool of choice: HELM (https://helm.sh)

First you need to acquire Helm.

On Windows, with chocolatey:

```
choco install -y kubernetes-helm
```

On MacOS, using Homebrew:

```
brew install kubernetes-helm
```

On Linux, using snap:

```
sudo snap install helm
```

In case you do not have access to a package manager, you can still get Helm there: https://github.com/kubernetes/helm/releases # Troubleshooting

Helm upgrade is stuck

Sometimes Helm fails in the middle of deploying a release. Usually this happens when the charts are misconfigured but can still be deployed.

After executing:

```
helm upgrade
```

It fails and you cannot execute it again, you keep getting:

```
$ helm upgrade ...
ERROR: UPGRADE FAILED: another operation (install/upgrade/rollback) is in progress
```

Although, there is nothing in the release list:

```
$ helm list --namespace messaging
NAME    NAMESPACE       REVISION        UPDATED STATUS  CHART   APP VERSION
```

The solution is to delete the information about that release in the Kubernetes cluster.

To do this, we need to understand where that information is stored. In Helm 3.0+, it is stored in some secrets of the namespace:

```
$ kubectl get secrets --namespace messaging --selector owner=helm
NAME                        TYPE                DATA    AGE
sh.helm.release.v1.demo.v1  helm.sh/release.v1  1       6d2h
sh.helm.release.v1.demo.v2  helm.sh/release.v1  1       2d1h
sh.helm.release.v1.demo.v3  helm.sh/release.v1  1       2d1h
sh.helm.release.v1.demo.v4  helm.sh/release.v1  1       2d1h
```

The problematic release can be found like this:

```
$ kubectl get secrets --namespace messaging  --selector owner=helm,status=pending-upgrade
NAME                          TYPE                DATA  AGE
sh.helm.release.v1.demo.v4    helm.sh/release.v1  1     2d1h
```

Delete that secret, and you can run helm upgrade again:

```
kubectl delete secrets sh.helm.release.v1.demo.v4
helm upgrade ...
```