Prepared for:

Federal Communications Commission

CMS Alliance to Modernize Healthcare
Federally Funded Research and Development Center

Contract No. 75FCMC18D0047

Task Order No. 273FCC19F0144

ACE Direct Platform Release Documentation

ACE Direct Installation and Configuration Guide

Version 4.0

July 1, 2020

Record of Changes

Version	Date	Author / Owner	Description of Change
1.0	November 4, 2016	CMS Alliance to Modernize Healthcare	Version 1.0 for release to Sponsor
1.1	February 17, 2017	CMS Alliance to Modernize Healthcare	Version 1.1 for release to Sponsor
2.0	November 1, 2017	CMS Alliance to Modernize Healthcare	Version 2.0 for release to Sponsor
2.1	May 24, 2018	CMS Alliance to Modernize Healthcare	Version 2.1 for release to Sponsor
3.0	October 26, 2018	CMS Alliance to Modernize Healthcare	Version 3.0 for release to Sponsor
3.1	April 9, 2019	CMS Alliance to Modernize Healthcare	Version 3.1 for release to Sponsor
4.0	July 1, 2020	Health FFRDC	Version 4.0 for release to Sponsor

Executive Summary

The Federal Communications Commission (FCC) Telecommunications Relay Service (TRS) Center of Expertise (COE) Project promotes the Commission's goal to foster innovations that advance functionally equivalent telecommunications. At the FCC's request, the Health FFRDC developed a Direct Video Calling (DVC) Auto-Routing Proof of Concept (POC) in support of the FCC's Accessible Communications for Everyone (ACE)[1] program. This DVC auto-routing platform enables direct calling from d/Deaf or hard of hearing individuals to an American Sign Language (ASL)-trained agent within the organization's call center. To demonstrate the capabilities of DVC, the FCC and the Health FFRDC have further advanced the original auto routing POC into a call center platform for 2 to 20 customer service representatives. This DVC platform is called ACE Direct.

Table ES-1 describes the new features released in this version of ACE Direct. Subsection 2.4 provides a complete history of ACE Direct releases and their associated features.

Table ES-1. New ACE Direct Features

Version	Release Date	New Infrastructure Feature or Capability
4.0	May 20, 2020	• **Kurento Media Server** – The media server provides advanced media processing, recording, monitoring and bandwidth controls. • **SIP Proxy Server (Mandatory component)**
3.2	September 6, 2019	• **ACE Quill Captioning** – Adds captioning to both the consumer and agent portals. • **Improved UI Layout** – Adjust button sizing and placement in the agent, consumer, and management portals. No infrastructure changes required
3.1	April 9, 2019	• **SIP Proxy Server (Optional component)** – The SIP Proxy server provides a single point of entry following Defense-in-Depth principles to create a layer between the ACE Direct environment and the Internet. This enhanced security provides a means to mitigate certain exploits and Distributed Denial of Service (DDoS) attacks.

[1] https://www.fcc.gov/ace

Version	Release Date	New Infrastructure Feature or Capability
3.0	October 26, 2018	• **Containers –** Containers simplify the overall ACE Direct installation, configuration, and deployment. They improve portability to different environments and add modularity. • **Management Portal Agent Provisioning UI** – The Management Portal Agent Provisioning screen makes it easy for call center managers to provision and maintain agent users in both OpenAM and ACE Direct. This allows customization of the default agent accounts. • **Data Logger Utility** – The Data Logger Utility captures and saves log files, trace information, and testing details automatically. This information facilitates troubleshooting interoperability and call quality issues. • **NGINX Custom Error Page** – The NGINX Custom Error Page is a more user-friendly page than the default NGINX error page. This ACE Direct page appears when the system is offline. • **ASL Video On Hold** – This feature allows the call center to display or advertise a custom message to a caller while on hold or after hours. • **Customizable ACE Direct URLs** – Customizable ACE Direct URLs allow owners, like the FCC, to customize the public URLs to match their corporate name or brand. An example is https://xyzcorp.org/XYZDirect/agent

Implementing the Direct Video Calling platform provides critical benefits toward achieving functionally equivalent telecommunications:

- **Improved Communications** – DVC improves privacy and decreases misrepresentation, which improves efficiency, effectiveness, and productivity.
- **Career Opportunities** – Employing native ASL users to handle customer service video calls expands hiring opportunities. Executive Order 13548 (July 2010) directed federal agencies to increase employment opportunities for people with disabilities.
- **Simple Implementation** – The technology to implement a DVC system is readily obtainable, affordable, and easy to set up.
- **Secure Communications** – With proper configuration, agencies can use high-speed broadband and their own internal networks without compromising security or contending with barriers created by firewalls.
- **Maintain ADA Compliance** – DVC ensures compliance with the Americans with Disabilities Act mandates.
- **Cost Savings** – Replacing three-way interpreted calls with two-way direct communication saves money by minimizing the need for repeat calls due to miscommunication and/or misunderstanding.

As part of this effort, the Health FFRDC developed and documented requirements and features, including user stories and associated use cases. The Health FFRDC also configured, tested, and integrated provider endpoint video devices with the ACE Direct platform. Detailed configuration and source code files are available for download and reproduction to improve solutions to

support the community. The public can download or clone these files at https://github.com/FCC/ACEDirect.

Table of Contents

List of Tables

1. Introduction

The Federal Communications Commission (FCC) Telecommunications Relay Service (TRS) Center of Expertise (COE) Project promotes the Commission's goal to foster innovations that advance functionally equivalent telecommunications. Toward that end, the project ensures that the Telecommunications Relay Service employs improved technology for persons who are d/Deaf[2], hard of hearing, DeafBlind, and/or have speech disabilities. In this document, "d/Deaf" describes individuals who are deaf in the audiological sense as well as those who identify as culturally Deaf.

The CMS Alliance to Modernize Healthcare Federally Funded Research and Development Center (the Health FFRDC), sponsored by the Centers for Medicare & Medicaid Services (CMS) and all divisions of the Department of Health and Human Services (HHS), is the first FFRDC dedicated to strengthening the nation's healthcare system. The MITRE Corporation (MITRE), an objective not-for-profit organization, operates the Health FFRDC in partnership with CMS and all HHS agencies to implement innovative ideas to solve our nation's toughest health problems.

1.1 Background

The FCC has embraced a research-based approach to achieve this goal by engaging the Health FFRDC to conduct independent engineering assessments that promote and demonstrate TRS's functional equivalence. As part of the Accessible Communications for Everyone (ACE) program, the Health FFRDC independently assesses voice telephone services, video access services, and Internet Protocol (IP)-based captioning technology; improvements to TRS efficiency; solutions for direct communication between people with communication disabilities and other telephone users; and the effectiveness, efficiency, and consumer response to current and future approaches for delivering TRS.

In continuing pursuit of the Commission's goal to advance functionally equivalent telecommunications, the Health FFRDC developed ACE Direct, an open source call center platform that supports DVC for 2 to 20 Agents. Implementing ACE Direct in a corporate production environment requires customization to ensure adherence to corporate practices and policies related to security, system configurations, cloud services, and availability.

The FCC encourages government agencies and private businesses to make DVC part of their call center strategy, because it offers significant gains for providing functionally equivalent telecommunications, including:

- **Improved Communications** – DVC improves privacy and decreases misrepresentation, which enhances efficiency, effectiveness, and productivity.
- **Career Opportunities** – Employing native American Sign Language (ASL) consumers to handle customer service video calls expands hiring opportunities. Executive Order 13548 (July 2010) directed federal agencies to increase employment opportunities for people with disabilities.

[2] MITRE is using d/Deaf as an umbrella term to describe individuals who are deaf in the audiological sense, as well as those who identify as culturally Deaf.

- **Simple Implementation** – The technology to implement a DVC system is readily obtainable, affordable, and easy to set up.
- **Secure Communications** – With proper configuration, agencies can use high-speed broadband and their own internal networks without compromising security or contending with barriers created by firewalls.
- **Maintain ADA Compliance** – DVC ensures compliance with the Americans with Disabilities Act (ADA) mandates.
- **Cost Savings** – Replacing three-way interpreted calls with two-way direct communication saves money by minimizing the need for repeat calls due to miscommunication and/or misunderstanding.

The Health FFRDC developed and documented ACE Direct requirements and features, including consumer stories and associated use cases. The FFRDC also configured, tested, and integrated provider endpoint video devices using the ACE Direct platform.

1.2 Purpose and Scope

This document is an installation and configuration guide for the components that make up the ACE Direct system. It describes how to install, configure, and deploy the entire ACE Direct application. Information on using ACE Direct can be found in the ACE Direct User Guide.

In addition to this release documentation, detailed configuration and source code files are available to the public at https://github.com/FCC/ACEDirect for download and reproduction of the platform to support and promote future platform enhancements for the d/Deaf, hard of hearing, DeafBlind, and speech-disabled community.

Your selection of hardware, operating system, hosting service and other infrastructure components is your decision. This document contains information pertaining to the installation and configuration of ACE Direct in an Amazon Web Services public cloud environment.

2. Installation and Configuration

This section presents guidance for the installation and configuration of the ACE Direct components except for OpenAM and the Kuando Busylight™. To reproduce this version of the platform, review the README.md file in the autoinstall folder at https://github.com/FCC/ACEDirect.

Several ACE Direct components (e.g., Asterisk and OpenAM) are deployable as Docker containers. Installation using Docker will be discussed in Section **Error! Reference source not found.**. Additional Docker instructions are in the README.md files of the relevant GitHub repositories.

Table 1 presents a list of initial requirements that must be met before installation.

Table 1. Initial Installation Requirements for ACE Direct

Prerequisite	Rationale
A domain name (your domain name must not contain an underscore)	Needed to provide end users with access to the ACE Direct Platform via a web browser.
Second-level DNS subdomain names and corresponding Address Records (A records) (e.g., “host.example.com”)	Needed to point the subdomains to a specific IP address used in components of ACE Direct.
Servers must have external Internet access	Needed for external connectivity and public-facing components.
The NGINX/STUN/TURN servers have a public and local IP address	Needed to facilitate application network traffic.
SELinux and IPv6 disabled on all hosts	ACE Direct does not currently support these technologies.
A “dial-in” number that has been registered in iTRS and/or a Session Initiation Protocol (SIP) trunk provider (such as Twilio)	To allow Consumers to place calls to ACE Direct. **Note:** FCC rules govern access to the iTRS database.
IPtables disabled for Asterisk and NGINX	Disable if not required.
Wildcard certificate (preferred)	Allows a single certificate to be used for all subdomains; otherwise, each Fully Qualified Domain Name (FQDN) will need its own certificate.
Create ‘acedirect’ user on servers	Required for node.js installation.
Access to system package mirrors (Needed for ‘yum install’)	Needed to install the necessary package dependencies for the ACE Direct Platform.
Chrome browser is required for Agent Desktop	Google Chrome browser leads the industry in WebRTC integration. Additionally, Chrome is used for all ACE Direct testing and provides the high level of stability.
Purchase Busylight™ hardware (optional)	The Busylight™ provides call centers with a visual indication of the Agent’s status.

2.1 SSL / TLS Certificate

The applications within ACE Direct use Hypertext Transport Protocol Secure (HTTPS) to provide security during web transmissions. To prevent major web browsers from potentially flagging the application as untrusted, it is recommended to acquire and implement a Secure Sockets Layer / Transport Layer Security (SSL/TLS) certificate from a trusted certificate authority (CA), such as LetsEncrypt (https://letsencrypt.org/). A wildcard certificate is preferred in order to reduce the number of certificates used by ACE Direct. After obtaining a certificate, follow the instructions in the respective README.md files in the various ACE Direct repositories to install the SSL certificate.

ACE Direct also employs the HTTP Strict Transport Security (HSTS) web policy mechanism to protect against protocol downgrade attacks and cookie hijacking.

2.2 Asterisk Installation and Configuration Script

An automated installation script has been developed to assist in the deployment and configuration of the Asterisk server. The Asterisk install script will install and configure Asterisk for ACE Direct on a Linux server. This script, which is available at https://github.com/mitrefccace/asterisk in the 'scripts' directory of the Asterisk repository, can be used to quickly create an Asterisk instance. The script will perform the following tasks automatically:

- Update current packages and install required packages for Asterisk
- Install PJSIP
- Install Asterisk
- Configure and apply Asterisk configs, media, scripts, and custom patches
- Start Asterisk

It is necessary to satisfy several prerequisites to the script before installing Asterisk. These prerequisites are listed in the top-level README.md file of the Asterisk repo. Review the README.md file in the Asterisk repository before running the script.

2.2.1 How It Works

The Asterisk PBX system relies on the configuration of three key files in the /etc/asterisk directory: pjsip.conf, extensions.conf, and http.conf. The pjsip.conf file contains the connection endpoints and parameters, and the extensions.conf file contains the syntax and programming for the extensions assigned to those endpoint connections. The http.conf file is the server configuration for Asterisk and defines the certificates Asterisk uses for secure communications.

2.2.1.1 Dial Plan Configuration

The dial plan is defined by the extensions.conf and pjsip.conf files in the /etc/asterisk directory, which work together to establish the connection between devices and route the calls to those devices. An endpoint device is any device that places or receives a call. Table 2 shows an example of common endpoint devices, the associated extension, and the required codecs for the sample dial plan configuration in this guide.

Table 2. Example Asterisk Endpoint Extensions

Extension	Purpose	Device	Codec
30001	ACE Direct Agent 1	softphone	h264/vp8/ulaw
30002	ACE Direct Agent 2	softphone	h264/vp8/ulaw
30003	ACE Direct Agent 3	softphone	h264/vp8/ulaw
30004	ACE Direct Agent 4	softphone	h264/vp8/ulaw

As displayed in Table 2, the extension is the number assigned to the endpoint. The ability of that endpoint to connect to the Asterisk instance is configured in the pjsip.conf file and the ability to dial to another endpoint is configured within the extensions.conf file. The device can be any phone capable of connecting to the Asterisk instance—a softphone, a desktop phone, a Cisco video communication phone, or other such device. At present, the configuration provided with ACE Direct can only be used with WebRTC and VRS Provider devices. The codec refers to the preferred media codec, or in some cases, the only available codec, used by the endpoint. The codec is also configured in the pjsip.conf file.

2.2.1.2 Extensions.conf Configuration File

The extensions.conf file is the configuration of your PBX. This file defines how the endpoint extensions communicate with each other or with the Asterisk server itself. The Asterisk server consists of many different applications working together to perform different functions. These functions can be called in the extensions.conf dial plan to route an endpoint device to a desired outcome. For example, a Consumer wants to dial voicemail. In extensions.conf, that configuration must state that the Consumer's phone number is to load the voicemail application, which is done by the following syntax:

```
exten => _6XXX,1,Answer()
     same => n,Voicemail(1234@ourpbx)
```

In this example, any extension matching the pattern _6XXX will be answered for immediate placement into the videomail application.

For specific phone numbers to route, as well as number schemes for unknown numbers, use the wildcard "X." In the following example as shown in the context "[from-internal]" of the extensions.conf file, all extensions ended with 6xxx are configured. The "[from-internal]" label refers to the context in which those extensions reside. A context is an organizational container that can host a group of extensions and extension patterns that can route and dial to other extensions within the same context. As shown in this example, the file needs no other changes.

```
[from-internal]
exten => _50XX,1,Dial(PJSIP/${EXTEN})
same => n,DumpChan()
same => n,HangUp()

exten => _6XXX,1,Dial(PJSIP/${EXTEN})
same => n,DumpChan()
same => n,HangUp()
```

```
exten => _70XX,1,Dial(PJSIP/${EXTEN})
same => n,DumpChan()
same => n,HangUp()
```

Extension patterns use wildcards to encompass a greater range of possible matches to the same string. To dial to an extension in another context, that context must be specifically identified in the coding. The context defined must also be the same for the "Context" attribute of the endpoints defined in the pjsip.conf file (please refer to subsection 2.2.1.3). For the Dial() function call, the extensions/numbers defined within the function call must be defined in pjsip.conf, as shown in the following subsection.

2.2.1.3 pjsip.conf Configuration File

The pjsip.conf file defines the endpoint connection parameters. Key elements must be configured in this file. The transport protocol settings and extension must be identified. The extension profile serves as its SIP identity and phone extension. In addition, the file must define the authentication method for this extension. PJSIP does not require the specification of a protocol and will dynamically select the best available option to use. The extension profile must specify the configuration settings for the endpoint, the network connection, the allowed codecs, and network address translation (NAT) traversal settings. The extension profile may also require that specific settings be disabled or not used. Visit the Digium Asterisk website to thoroughly understand the function of the pjsip.conf file and its associated settings.

The Asterisk PBX uses the pjsip.conf file to load the connection points from which the endpoint devices will acquire their information and connect. This means that this file is responsible for the endpoint connection, handling the contact that connects to that endpoint, routing the call, and passing information. The following critical attributes are defined in a profile within pjsip.conf:

- **Transport.** The transport defined within the pjsip.conf file specifies the configuration for TCP, UDP, WS, or WSS connections. These transport settings require configuration as to the external IP, internal IP, and in the case of web sockets, certificate information to be used for secure communication.
- **Endpoint.** An endpoint to the pjsip.conf file is a profile that matches to an endpoint device. The endpoint can be defined in pjsip independently or as a template that numerous extensions can call. This template helps to keep a condensed file because otherwise it can grow quite large. Endpoint templates are identified by (!) after the identifying name.
- **Register.** A registration is a type of authentication from an endpoint device. The device will send authentication information to Asterisk that will then "Register" the device to the PBX with the assigned endpoint profile. A registration is a temporary assignment of configuration options and network settings to identify the endpoint device and route calls.
- **Contact.** A contact is the network-identifying information of a device that has registered and its corresponding extension. Viewing the contact information within the Asterisk console verifies that an endpoint device has authenticated and been assigned its configuration correctly.

Once the profile has been defined, extensions and numbers can be associated with the profile, which will be referenced in the extensions.conf file to help route calls to their proper destination. Use the pre-defined profiles in the Asterisk repository because they are already configured specifically for use with ACE Direct.

2.2.2 Web Secure Sockets

This Consumer portal uses a technology called WebRTC. Asterisk must use web sockets and, for most browsers, secure web sockets to successfully communicate with WebRTC. Asterisk should only use a certificate from a valid certificate authority when using WebRTC.

2.2.3 Sample Configuration Files

Note: The following samples are not up to date with the latest configuration in the Asterisk repository. The intent is to provide a basic overview of some of the configuration and their associated parameters. To view the latest configuration files, visit the Asterisk GitHub repository.

2.2.3.1 Pjsip.conf Configuration

The following is an example of the Pjsip.conf file used by Asterisk:

```
---------------------------------top of pjsip.conf file----------
---------------------------------------
[transport-     wss]  ;name of transport configuration

type=transport             ; type being configured is of type
transport
protocol=       wss   ;protocol is web secure socket, can be
tcp,udp,ws,wss
bind=0.0.0.0:443      ;bound ip/port
external_media_address=52.45.109.230  ;external ip of Asterisk
cert_file  =/etc/asterisk/keys/star.pem     ;certificate pem
priv_key_file=/etc/asterisk/keys/star.key   ;certificate key
[endpoint-basic](!) ;Defines the name of the endpoint, the (!)
designates it as a template
type= endpoint   ;defines the type
transport=transport-wss     ;defines the transport to be used
context=   from-internal    ;defines the context, must correspond
to context in extensions.conf
disallow=  all   ;explicitly deny all media unless specified
allow=     ulaw  ;specify allow ulaw audio codec
allow=vp8        ;specify allow vp8 video
allow=h264       ;specify h264 video
allow=t140       ;specify allow t140 text protocol
force_rport=yes  ;network configuration, reflexive port
direct_media=no  ;network configuration, do not establish direct
media link (NAT)
rewrite_contact=yes
media_address=52.45.109.230
rtp_symmetric=yes
```

```
ice_support=yes
message_context =internal-im    ;context for internal messenger

[30001](endpoint-basic)    ;Extension information, extension
username is 30001
auth  =auth30001 ;The auth profile to use
aors  =30001     ;The aors profile to use

[auth30001](auth-userpass)
password=  changeit!  ;password to be changed
username=30001        ;username for authentication, typically
extension number

[30001](aor-single-reg)
remove_existing  =yes  ; Remove pre-existing contact
max_contacts     =1    ;Number of simultaneous contacts registered
to an AOR
qualify_frequency     =5    ;Interval in seconds of qualifying a
registered contact
authenticate_qualify=yes    ;Send authentication request on
qualify if required
---------------end of pjsip.conf file---------------------------
```

2.2.3.2 WebRTC Sample Configuration

The following is a sample WebRTC-enabled endpoint template in pjsip.conf:

```
[endpoint-webrtc](!)
type=endpoint
transport=transport-wss
context=from-internal
disallow=all
allow=ulaw
allow=h264
allow=vp8
allow=t140
force_avp=yes
use_avpf=yes
media_encryption=dtls
dtls_verify=fingerprint
dtls_fingerprint=SHA-1
dtmf_mode=auto
dtls_rekey=0
dtls_cert_file=/etc/asterisk/keys/asterisk.pem
dtls_ca_file=/etc/asterisk/keys/ca.crt
dtls_setup=actpass
ice_support=yes
media_use_received_transport=yes
rtp_symmetric=yes
force_rport=yes
rewrite_contact=yes
message_context=internal-im
```

```
rtcp_mux=yes
```

2.3 Node.js

Node.js is a platform built on Chrome's V8 JavaScript run-time engine, which was developed to build fast and scalable network applications. Node.js is event driven, lightweight, and efficient—ideal for data-intensive real-time applications that run across distributed devices.

Node.js servers provide the functionality of ACE Direct. More specifically, the servers host the Agent, Consumer, Management, and Call Detail Record (CDR). Node.js also provides Agent Data and Provider Data through RESTful application programming interfaces (API) to ACE Direct.

Examples of these RESTful services are VRS lookup and Agent verify functions. An Asterisk instance should already be up and running to support most of the Node.js instances.

Instructions for downloading and installing Node.js can be found on the official Node.js website (https://nodejs.org/en/). See the ACE Direct documentation in the public repos for the required versions for the operating system, Node.js, and other tools.

2.3.1 Node.js Components Installation

The manual installation procedures for the Node.js components of ACE Direct have been deprecated and are no longer supported. To install the Node.js components, follow the README.md file in the autoinstall folder located at https://github.com/FCC/ACEDirect.

2.4 Management Portal

The ACE Direct Management Portal provides key performance indicators for real-time monitoring by the call center Manager. This information may improve user experience, increase user satisfaction, and support monitoring of overall call center performance. Some of the configurations can be found in parameter_desc.json of the dat repo and in queues.conf of the asterisk repo.

2.4.1 Log Files

Log files are in managementportal/logs. The debug level is defined by the debug_level field in the configuration file. The valid options are as follows: ALL, TRACE, DEBUG, INFO, WARN, ERROR, and FATAL.

2.4.2 Management Dashboard

2.4.2.1 Web Server and Client Configuration

The Management Dashboard functionality relies on the Asterisk server to collect reports on call agents and queue status. The Management Dashboard (dashboard.js and dashboard.ejs) is hosted on the Node.js server (server-db.js). The following subsections describe the operational data flow.

2.4.2.1.1 Server Side

- **server-db.js** – Communicates with the Asterisk Server through the Asterisk Management Interface (AMI) protocol. AMI events provide the Management Portal with data on queue and agent status.

2.4.2.1.2 Client Side

- **dashboard.ejs** – Controls the User Interface (UI) for the Management Portal Dashboard.
- **dashboard.js** – Uses JavaScript data structures to store information to be displayed on the dashboard.ejs page.

2.4.2.2 Asterisk Management Interface

The AMI allows a client application to connect to an Asterisk instance and issue actions (requests) and receive events (responses). The Management Portal performs five unique AMI action calls to collect data on queue and agent status. Table 3 shows the AMI actions and descriptions.

Table 3. AMI Actions and Descriptions

AMI Action	Description	Syntax	Arguments	Triggered Events
Agents	Queries for information about all agents	Action: Agents ActionID: <value>	• ActionID – ActionID for this transaction. Will be returned.	Agents AgentsComplete
QueueReset	Resets the running statistics for a queue	Action: QueueReset ActionID: <value> Queue: <value>	• ActionID – ActionID for this transaction. Will be returned. • Queue – The name of the queue on which to reset statistics.	
Queues	Queries for queues information	Action: Queues		Queues

AMI Action	Description	Syntax	Arguments	Triggered Events
QueueStatus	Check the status of one or more queues	Action: QueueStatus ActionID: <value> Queue: <value> Member: <value>	• ActionID – ActionID for this transaction. Will be returned. • Queue – Limit the response to the status of the specified queue. • Member – Limit the response to the status of the specified member.	QueueStatus QueueStatusComplete QueueMember QueueParams
QueueSummary	Requests Asterisk to send a QueueSummary event	Action: QueueSummary ActionID: <value> Queue: <value>	• ActionID – ActionID for this transaction. Will be returned. • Queue – Queue for which the summary is requested.	QueueSummary QueueSummaryComplete

The dashboard server listens for the Asterisk AMI events shown in Table 4.

Table 4. Asterisk AMI Events and Descriptions

AMI Action	Description	Syntax	Fields
AgentsComplete	Generated when an agent has finishes servicing a member in the queue	Event: AgentComplete Queue: <value> Member: <value> MemberName: <value> HoldTime: <value> [Variable:] <value> TalkTime: <value> Reason: <value> Queue: <value> Uniqueid: <value> Channel: <value> Member: <value> MemberName: <value> HoldTime: <value>	• Queue – The name of the queue. • Member – The queue member's channel technology or location. • MemberName – The name of the queue member. • HoldTime – The time the channel was in the queue, expressed in seconds since 00:00, Jan 1, 1970 UTC. • Variable – Optional channel variables from the ChannelCalling channel • TalkTime – The time the agent talked with the member in the queue, expressed in seconds since 00:00, Jan 1, 1970 UTC. • Reason – consumer – agent – transfer • Queue • Uniqueid • Channel • Member • MemberName • HoldTime
AgentLogin	Generated when an agent logs in	Event: AgentLogin Agent: <value> Channel: <value> Uniqueid: <value>	• Agent – The name of the agent. • Channel • Uniqueid
AgentLogoff	Generated when an agent logs off	Event: AgentLogoff Agent: <value> Agent: <value> Logintime: <value> Uniqueid: <value>	• Agent – The name of the agent. • Agent • Logintime • Uniqueid
FullyBooted	Generated when all Asterisk initialization procedures complete	Event: FullyBooted Status: <value>	• Status

AMI Action	Description	Syntax	Fields
QueueMemberAdded	Generated when a new member is added to the queue	Event: QueueMemberAdded Queue: <value> Location: <value> MemberName: <value> StateInterface: <value> Membership: <value> Penalty: <value> CallsTaken: <value> LastCall: <value> Status: <value> Paused: <value> Queue: <value> Location: <value> MemberName: <value> StateInterface: <value> Membership: <value> Penalty: <value> CallsTaken: <value> LastCall: <value> Status: <value> Paused: <value>	• Queue – The name of the queue. • Location – The queue member's channel technology or location. • MemberName – The name of the queue member. • StateInterface – Channel technology or location from which to read device state changes. • Membership – dynamic – realtime – static • Penalty – The penalty associated with the queue member. • CallsTaken – The number of calls this queue member has serviced. • LastCall – The time this member last took call, expressed in seconds since 00:00, Jan 1, 1970 UTC. • Status – The numeric device state status of the queue member. – 0 - AST_DEVICE_UNKNOWN – 1 - AST_DEVICE_NOT_INUSE – 2 - AST_DEVICE_INUSE – 3 - AST_DEVICE_BUSY – 4 - AST_DEVICE_INVALID – 5 - AST_DEVICE_UNAVAILABLE – 6 - AST_DEVICE_RINGING – 7 - AST_DEVICE_RINGINUSE – 8 - AST_DEVICE_ONHOLD • Paused – 0 – 1 • Queue • Location • MemberName • StateInterface • Membership • Penalty • CallsTaken • LastCall • Status • Paused
QueueMemberRemoved	Generated when a queue member is removed from the queue	Event: QueueMemberRemoved Queue: <value> Location: <value> MemberName: <value> Queue: <value> Location: <value> MemberName: <value>	• Queue – The name of the queue. • Location – The queue member's channel technology or location. • MemberName – The name of the queue member. • Queue • Location • MemberName

2.4.3 Call Detail Record Dashboard

The Asterisk PBX system can collect and store call data for each call that traverses the system. These data records, referred to as CDRs, are collected and stored for use by business intelligence (BI) tools, statistical analysis, and even TRS compensation and reimbursement. The information collected is invaluable and can be retrieved and stored using various methods.

By default, the Asterisk server stores CDRs in a file called master.csv located in /var/log/cdr-csv. CDR records can be collected in real time, or near-real time, in various ways. The Asterisk server also supports different database connections. One method is to use Asterisk on an internal application for connecting to a database. Another is to configure an Open Database Connectivity (ODBC) connection for Asterisk. There is no restriction on the type of relational database used. For example, MySQL, MariaDB, and others are all viable options. The Health FFRDC elected to use a MySQL database server with Asterisk connecting through an ODBC connection.

2.4.3.1 MySQL Installation

To start, it is necessary to install additional packages for the MySQL server and the ODBC connector as follows:

```
sudo yum install unixODBC unixODBC-devel libtool-ltdl libtool-
ltdl-devel mysql-connector-odbc
wget http://repo.mysql.com/mysql-community-release-el7-
5.noarch.rpm
sudo rpm -ivh mysql-community-release-el7-5.noarch.rpm
sudo yum install mysql-server
sudo systemctl start mysqld
sudo yum update
```

Once the packages are installed, it is necessary that Asterisk recognizes these new packages. To do so. migrate to the Asterisk directory, which in the MITRE build is in /usr/src/asterisk-13.8.0/.

From this directory, run:

```
./configure
make menuselect
```

Once the graphic window is displayed, check to make sure that the res_odbc module is selected. Save and Exit the graphic window.

Next, run the following:

```
make && make install
```

This will configure the Asterisk installation for use with the ODBC connection. Once the installation script finishes successfully, proceed to start the sql server.

Once MySQL has been installed, run the basic hardening script for MySQL to remove anonymous connections, the test database, and establish the root password. To do this, run the following script and follow the prompts:

```
sudo /usr/bin/mysql_secure_installation
```

2.4.3.2 Database Configuration

After installing MySQL and completing the initial configuration, log into MySQL using:

```
mysql -u root -p
```

Once logged in, create the database and the table needed to store the CDRs. To do so, run the following commands:

```
CREATE DATABASE asterisk;
USE asterisk;

CREATE TABLE `bit_cdr` (
`calldate` datetime NOT NULL default '0000-00-00 00:00:00',
`clid` varchar(80) NOT NULL default '',
`src` varchar(80) NOT NULL default '',
`dst` varchar(80) NOT NULL default '',
`dcontext` varchar(80) NOT NULL default '',
`channel` varchar(80) NOT NULL default '',
`dstchannel` varchar(80) NOT NULL default '',
`lastapp` varchar(80) NOT NULL default '',
`lastdata` varchar(80) NOT NULL default '',
`duration` int(11) NOT NULL default '0',
`billsec` int(11) NOT NULL default '0',
`disposition` varchar(45) NOT NULL default '',
`amaflags` int(11) NOT NULL default '0',
`accountcode` varchar(20) NOT NULL default '',
`userfield` varchar(255) NOT NULL default '',
`uniqueid` VARCHAR(32) NOT NULL default '',
`linkedid` VARCHAR(32) NOT NULL default '',
`sequence` VARCHAR(32) NOT NULL default '',
`peeraccount` VARCHAR(32) NOT NULL default '' );

ALTER TABLE `bit_cdr` ADD INDEX ( `calldate` );
ALTER TABLE `bit_cdr` ADD INDEX ( `dst` );
ALTER TABLE `bit_cdr` ADD INDEX ( `accountcode` );
```

To create a user account for remote connections, use the following syntax:

```
CREATE USER 'username'@'%' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON *.* TO 'username'@'%' WITH GRANT OPTION;
FLUSH PRIVILEGES;
```

2.4.3.3 Asterisk Integration

Now that the database is up and running, connect to Asterisk by using the ODBC connector, as follows:

```
vi /etc/odbcinst.ini
 [MySQL]
Description     = ODBC for MySQL
Driver          = /usr/lib/libmyodbc5.so
Setup           = /usr/lib/libodbcmyS.so
Driver64        = /usr/lib64/libmyodbc5.so
Setup64         = /usr/lib64/libodbcmyS.so
FileUsage       = 1

vi /etc/odbc.ini
 [asterisk-connector]
Description = MySQL connection to 'asterisk' database
Driver = MySQL
Database = asterisk
Server = localhost
User = root
Password = password
Port = 3306
Socket = /var/lib/mysqld/mysqld.sock
```

To test your configuration, run the following command from the CLI:

```
echo "select 1" | isql -v asterisk-connector
```

A message of Connected! should be returned.

```
| Connected!                               |
SQLRowCount returns 1 1 rows fetched
```

Now you will need to point Asterisk to your new database. To do so, modify the /etc/asterisk/res_odbc.conf file with the database name, username, password, and port of the ODBC connection you have set up. The dsn variable will point to the dsn identified:

```
vi /etc/asterisk/res_odbc.conf

[asterisk]
enabled => yes
dsn => asterisk-connectorusername => username
password => password
pooling => no
limit => 99999
pre-connect => yes
```

Next, edit /etc/asterisk/cdr_odbc.conf to include the following:

```
vi /etc/asterisk/cdr_odbc.conf
```

```
[global]
dsn=asterisk
loguniqueid=yes
table=bit_cdr
```

Note: the dsn variable will point to the DSN identified in res_odbc.conf, NOT /etc/odbc.ini.

Next, edit /etc/cdr_adaptive_odbc.conf to include the following:

```
vi /etc/asterisk/cdr_adaptive_odbc.conf
[asteriskcdr]
connection=asterisk
table=cdr
alias start=calldate
```

Next, ensure CDR writing is enabled in cdr_manager.conf:

```
vi /etc/asterisk/cdr_manager.conf
[general]
enabled = yes
```

Finally, restart Asterisk:

```
sudo service asterisk restart
```

2.4.3.4 Node.js Integration

The CDR reporting functionality relies on two Node.js servers: the Management Portal (server-db.js) and the acr-cdr (app.js). These servers provide the following capabilities:

- server-db.js
  - Receives GET call for /cdrinfo.
  - Performs GET call to app.js /getallcdrrecs.
  - Acts as a middleman for the cdr.html page to the app.js node server. This node server can be bypassed by changing the cdr.html GET call from /cdrinfo to http://<app.js location>/getallcdrrecs if the CDR Dashboard needs to be separated from the Management Portal.
- app.js
  - Receives GET call for /getallcdrrecs.
  - Performs query to the Asterisk CDR database table. Returns a JSON object of the data.

2.5 Agent / Agent Database

ACE Direct uses a database to store Agent information and to verify Agent identity when an Agent logs into the system.

To host the Agent data, a MySQL database server is required. A RESTful API developed using Node.js provides access to the data for Agent verification. See the ACE Direct documentation in the public repos for the required versions for the operating system, Node.js, and other tools.

2.5.1 MySQL Database Server Configuration

The Health FFRDC team developed a MySQL database containing several tables to store the Agent-related data. Create a database and use the dat/acedirectdefault.sql MySQL script found in https://github.com/FCC/ACEDirect to create the application tables.

Verify that the MySQL database is up, running, and accepting connections. A tool like MySQL Workbench can quickly verify that the configuration is correct. The dat/acedirectdefault.sql script pre-populates the database with default agent data; however, you must first modify dat/acedirectdefault.sql to have actual values for the EXTENSION_PASSWORD, _ASTERISK_PASSWORD_, and _ACEDIRECT_PASSWORD_ placeholders. Remember to update dat/config.json to include your actual database name, users, and passwords.

2.6 Video Relay Service User Database

The VRS user database was developed to emulate a VRS user lookup in the iTRS-URD from the Agent desktop until full access to the iTRS-URD is obtained. This lookup verifies that the Consumer-provided phone number is a registered VRS number. A MySQL database server is required. A RESTful API developed using Node.js provides access to the data for Consumer verification. See the ACE Direct documentation in the public repos for the required versions for the operating system, Node.js, and other tools.

2.6.1 MySQL Database Server Configuration

The Health FFRDC team developed a MySQL database containing a single table to store the emulated VRS lookup data. The dat/acedirectdefault.sql script pre-populates the database with default VRS data.

Verify that the MySQL database is up, running, and accepting connections. A tool like MySQL Workbench can quickly verify that the configuration (username, password) is correct.

2.7 STUN and TURN Server Installation

If a host is located behind a NAT firewall, it can be difficult (if not impossible) for that host to communicate directly with other hosts (peers). In these situations, the host must use the services of an intermediate node as a communication relay. This specification defines a protocol, TURN (Traversal Using Relays around NAT), which allows the host to control the operation of the relay and to exchange packets with its peers using the relay. TURN differs from other relay control protocols because it allows a client to communicate with multiple peers using a single relay address. A TURN server, which is an implementation of the STUN protocol, uses a relay to provide an alternate method for NAT discovery and traversal (STUN). TURN can traverse symmetric NAT instances. The STUN server may be used by Asterisk if there is a Public Asterisk IP-Address; it may also be used by the Signaling-Server and the Kurento (KMS) server. In general, a TURN Server can run in STUN, TURN, or both modes. In practice, the STUN and TURN Servers are separated; this helps with debugging and prevents some security issues. For demonstration purposes, STUN is installed on Centos, and TURN is installed on Ubuntu.

2.7.1 STUN Installation

For STUN services CentOS, turnserver was used. Before installing turnserver, ensure the dependencies are installed:

```
sudo yum -y install
sudo yum install -y make gcc cc gcc-c++ wget openssl-devel
libevent libevent-devel mysql-devel mysql-server
```

Then install the LibEvent modules, a dependency of turnserver:

```
wget https://github.com/downloads/libevent/libevent/libevent-
2.0.21-stable.tar.gz
tar xvfz libevent-2.0.21-stable.tar.gz
cd libevent-2.0.21-stable && ./configure
sudo make && sudo make install && cd ..
```

Finally, install turnserver:

```
wget http://turnserver.open-
sys.org/downloads/v3.2.3.8/turnserver-3.2.3.8.tar.gz
tar -xvzf turnserver-3.2.3.8.tar.gz
cd turnserver-3.2.3.8 && ./configure
sudo make && sudo make install
```

If desired, you may configure a turnserver config file to add user credentials as well as the address and port to listen on:

```
mkdir /etc/turnserver
vi /etc/turnserver/turnserver.conf

# setting static accounts
# Remember, "static" accounts are not dynamically checked by the
turnserver process.
user=username:password

# listen ports
listening-port=<port>
listening-ip=<local_ip>
```

Start the turnserver process. Use the first command to implement the config file, and the second not to:

```
nohup turnserver -v -r  ip:port -z -c
/etc/turnserver/turnserver.conf &
nohup turnserver -L <local_IP> -v -z --min-port 10000 --max-port
20000 -n &
```

2.7.1.1 STUN System Startup

You can enable turnserver to start on system boot if desired. The following instructions have been validated on CentOS servers and may or may not work on other Linux distributions. To do so, first create the following script, make it executable, and save it as /etc/rc.d/init.d/turnserver:

```
#!/bin/bash
# chkconfig: 2345 20 80
# description: Manage turnserver as a system service so it starts
on boot

# Source function library
. /etc/init.d/functions

# This variable will be used to define what IP address turnserver
listens on
# You may need to change this depending on how your network
service is configured
HOST=$(hostname -I | awk '{print $1}')


# This is the port that STUN will listen on.
PORT=3478

start() {
        /usr/local/bin/turnserver -L $HOST -p $PORT -v -z --min-
port 10000 --max-port 20000 -n
        echo "STUN has started successfully"
}

stop() {
        killall -9 turnserver
        echo "STUN server stopped successfully"
}

case $1 in
        start)
                start
                ;;
        stop)
                stop
                ;;
        restart)
                stop
                start
                ;;
        status)
                STATUS=$(netstat -tanp | grep turnserver | grep
$PORT)
                if [ ${#STATUS} == 0 ]
                then
                        echo "STUN server is not currently
```

```
running"
                else
                        echo "STUN server is currently running"
                        echo $STATUS
                fi
                ;;
        *)
                echo "Usage: service turnserver
(start|stop|restart|status)"
                exit 1
                ;;
esac
exit 0
```

Then, create the following systemd configuration file and save it as /usr/lib/systemd/system/turnserver.service:

```
[Unit]
Description=Turnserver Service
After=network-online.target
Wants=network-online.target

[Service]
Type=simple
ExecStart=/etc/rc.d/init.d/turnserver start
;ExecStop=/root/start-turn.sh stop
;ExecReload=/root/start-turn.sh restart

[Install]
WantedBy=default.target
```

Finally, reload the systemctl daemon, enable the turnserver service in systemctl, then reboot the server and use the 'netstat' command to confirm that turnserver is now starting upon boot:

```
systemctl daemon-reload
systemctl enable turnserver.service
reboot now
```

2.7.2 TURN Installation

For TURN services Ubuntu, coturn was used.

```
Run: apt-get update && apt-get install coturn
```

Edit: /etc/turnserver.conf and set the following variables (typical values are shown):

- listening-port (3478)
- listening-ip (private IP address)

- external-ip (public IP address)
- min-port (10000)
- max-port (20000)
- realm (domain name)
- user(user:password)

2.7.3 TURN System Startup

Edit /etc/init.d/coturn and add the following variables:

```
PATH=/usr/bin:/sbin:/usr/sbin:/bin
DESC=coturn             # COTURN
NAME=coturn             # TURN Server
PROCNAME=turnserver                  # Binary name
DAEMON=/usr/bin/turnserver
DAEMON_ARGS="-c /etc/turnserver.conf -o -v"            #
Arguments to run the daemon with
PIDFILE_DIR=/var/run
PIDFILE=/var/run/$PROCNAME.pid
SCRIPTNAME=/etc/init.d/$NAME
USER=turnserver
GROUP=turnserver

   Edit /etc/default/coturn and set TURNSERVER_ENABLED=1 (start
on boot)

   To start: turnserver –o –v, or run service start coturn
```

2.8 iTRS ENUM Database

For phone numbers of d/Deaf or hard of hearing users, iTRS is the authoritative database. The lookup of the iTRS ENUM database performs Domain Name System (DNS) queries to determine a Uniform Resource Identifier (URI) for a 10-digit telephone number. There is a GUI as well as a programmatic interface. **Note: Access to the iTRS ENUM database requires permission from the FCC.**

To query the production iTRS database requires establishing an IPSec tunnel with the iTRS ENUM administrator. The Health FFRDC team added the required IP address first in the DNS settings (/etc/resolv.conf) for the ENUM lookup to work. Note that with the default AWS instance settings, any changes made to the resolv.conf file will be overwritten on restart of the network service because new values are queries from DNS. This behavior must be disabled.

The following example snippet of code, which is part of an Asterisk Dial Plan, comes from an extensions.conf file. Subsection 3.2.2.1 provides more information on the Asterisk Dial Plan.

```
exten => _9.[1-9]XXXXXXXXX, 1,
Set(sipuri=${ENUMLOOKUP(+${EXTEN:1},sip,,1,itrs.us)})
```

```
same => n,NoOp("Outbound Direct Video Call to: ${EXTEN:1}") ;
just for informational purposes
same => s,n,SipAddHeader(P-Asserted-Identity: <sip:nnnnnnnnnn>)
;set the callerID number
same => n,NoOp("sipuri: ${sipuri:1}")
same => n,Dial(SIP/${sipuri:1},30)
```

2.9 StrongSWAN for Secure Socket Layer Tunnel

StrongSWAN is an open source VPN software that is widely popular within the IPSec industry. StrongSWAN can be installed on both CentOS and Amazon Linux servers; the following instructions were implemented on an Amazon Linux EC2 instance.

In the following installation scenario, the local side of the VPN is running within AWS. The remote side provider requires both a public IP for the VPN endpoint and for the tunneled traffic. The implementation requires two public IP addresses on the local side and translates all local traffic to one of those addresses for the VPN tunnel. Also, on the remote end, there is only one accessible IP address. Minor adjustments would be required to allow access to more than one destination IP address or to allow a range of IP addresses directly (with or without NAT) through the VPN.

All the following commands must be run as root (or via sudo).

2.9.1 Installation

First, use yum to install StrongSWAN:

```
yum install -y strongswan
```

Then, modify the /etc/strongswan/ipsec.conf table to reflect the following:

```
config setup
        #charondebug="ike 2, net 3, knl 2, cfg 2"    #useful
debugs
conn %default
        ikelifetime=480m
        keylife=60m
        rekeymargin=3m
        keyingtries=1
        keyexchange=ikev1
        authby=secret
conn PeerProvider
        auto=start
        ike=aes128-sha1-modp1024        #P1: modp1536 = DH group
2
        esp=aes128-sha1-modp1024        #P2

        left=<StongSWAN local address>  #Local outside address
        leftsubnet=<dummy address>/32   #network behind Local
        leftid=<StrongSWAN public IP>   #IKEID sent by Local
        leftfirewall=yes
```

```
        right=<remote peer address.     #PeerProvider outside
address
        rightsubnet=<remote server address>/32 #network behind
PeerProvider
        24ighted=<remote peer address.   #IKEID sent by
PeerProvider
```

Agree with your peer on a shared key and then modify the /etc/strongswan/ipsec.secrets file to reflect the following:

```
<StrongSWAN public IP> <remote server address> : PSK "<Key
obtained from PeerProvider in quotes>"
```

IPTables must be configured to perform NAT between the source subnet and the public IP identified for tunnel traffic. Create the /etc/iptables.conf file and add the following:

```
*nat
:PREROUTING ACCEPT [5:436]
:INPUT ACCEPT [1:92]
:OUTPUT ACCEPT [34:9996]
:POSTROUTING ACCEPT [34:9996]
-A POSTROUTING -s <VPC CIDR block>/24 -d <remote server
address>/32 -o eth0 -j SNAT –to-source <dummy address>
COMMIT


*filter
:INPUT ACCEPT [1063:95316]
:FORWARD ACCEPT [12:1032]
:OUTPUT ACCEPT [1018:375057]
COMMIT
```

The server needs a dummy interface associated with the public IP for tunnel traffic, and the IPTables configuration must be loaded at startup. Append the following lines to /etc/rc.d/rc.local:

```
/sbin/modprobe dummy
/sbin/ifconfig dummy0 <dummy address> netmask 255.255.255.0
/sbin/iptables-restore < /etc/iptables.conf
```

Move into the /etc/rc3.d directory and create the following symbolic links to ensure that StrongSWAN gracefully starts/stops when the EC2 instance is rebooted:

```
cd /etc/rc3.d
ln -s  ../init.d/strongswan S48strongswan
ln -s  ../init.d/strongswan K52strongswan
```

Routing must be enabled on the server. Modify the variable within the /etc/sysctl.conf file to the following:

```
net.ipv4.ip_forward = 1
```

Create the following script in /root/scripts to automatically restart StrongSWAN if ITRS queries fail:

```
#!/bin/bash

DEBUG=true
LOG=/var/log/itrsmon.log

if ! dig @<remote server address> in naptr
9.6.8.4.1.5.5.7.2.7.1.itrs.us>/dev/null 2>&1
then
   echo $(/bin/date) " - ITRS Lookup failed" >> $LOG
   /etc/init.d/strongswan restart
else
   $DEBUG && echo $(/bin/date) " - ITRS Lookup success" >> $LOG
fi
```

Add the following line to the crontab to monitor StrongSWAN by running the script once per minute:

```
$ crontab -e


* * * * * /root/scripts/itrs_mon.sh
```

Finally, reboot the instance to verify that the StrongSWAN service is running (using the 'strongswan status' command). Once the service is running, you can move onto the following "AWS Specific Config" section if you are in AWS. When you have completed those steps, your DNS queries to the iTRS database from Asterisk servers should be successful. An example of a successful iTRS query is as follows:

```
$ dig @<remote server address> in naptr
1.1.1.1.1.1.1.1.1.1.1.itrs.us
; <<>> DiG 9.9.4-RedHat-9.9.4-50.el7_3.1 <<>> @<remote server
address> in naptr 1.1.1.1.1.1.1.1.1.1.1.itrs.us
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 32364
;; flags: qr aa rd ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0,
ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION :
; EDNS : version : 0, flags :; udp : 5120
;; QUESTION SECTION :
;1.1.1.1.1.1.1.1.1.1.1.itrs.us. IN NAPTR

;; ANSWER SECTION:
1.1.1.1.1.1.1.1.1.1.1.itrs.us. 900 IN NAPTR 10 1 "u" "E2U+h323"
"!^(.*)$!h323:\\1@192.168.1.1!" .
```

```
;; Query time: 15 msec
;; SERVER: <remote server address>#53(<remote server address>)
;; WHEN: Fri Aug 04 14:40:59 UTC 2017
;; MSG SIZE rcvd: 115
```

2.9.2 AWS-Specific Configuration

2.9.2.1 VPC Route Table

A route must be created in the VPC route table to ensure that traffic to the iTRS database is routed to the VPN instance (which in turn routes it out over the VPN tunnel).

2.9.2.2 Source / Destination Checking

By default, instances drop packets when source and destination information do not match that of an instance. You can disable this behavior from the AWS console by selecting an instance and going to network options.

2.9.3 Troubleshooting

You can run 'strongswan statusall' to view the status of the VPN tunnel. The expected command output will look like the following:

```
$ strongswan statusall
Status of IKE charon daemon (strongSwan 5.4.0, Linux 4.9.38-
16.33.amzn1.x86_64, x86_64):
 uptime: 116 minutes, since Aug 04 12:46:06 2017
 malloc: sbrk 1622016, mmap 0, used 502608, free 1119408
 worker threads: 11 of 16 idle, 5/0/0/0 working, job queue:
0/0/0/0, scheduled: 2
 loaded plugins: charon aes des rc2 sha2 sha1 md4 md5 random
nonce x509 revocation constraints acert pubkey pkcs1 pkcs8 pkcs12
pgp dnskey sshkey pem openssl gcrypt fips-prf gmp xcbc cmac hmac
ctr ccm gcm curl attr kernel-netlink resolve socket-default farp
stroke vici updown eap-identity eap-md5 eap-gtc eap-mschapv2 eap-
tls eap-ttls eap-peap xauth-generic xauth-eap xauth-pam xauth-
noauth dhcp
Listening IP addresses:
 <StrongSWAN local address>
 <dummy address> ← both IP addresses should be listed
Connections:
 PeerProvider: <StrongSWAN local address>…<remote peer address.
IKEv1
 PeerProvider: local: [<StrongSWAN public address>] uses pre-
shared key authentication
 PeerProvider: remote: [<remote peer address.] uses pre-shared
key authentication
 PeerProvider: child: <dummy address>/32 === <remote server
address>/32 TUNNEL
Security Associations (1 up, 0 connecting):
```

```
 PeerProvider[1]: ESTABLISHED 116 minutes ago, <StrongSWAN local
address>[<StrongSWAN public address>]...<remote peer
address.[<remote peer address.]
 PeerProvider[1]: IKEv1 SPIs: <<SPI>>, pre-shared key
reauthentication in 21 hours
 PeerProvider[1]: IKE proposal:
AES_CBC_128/HMAC_SHA1_96/PRF_HMAC_SHA1/MODP_1024
 PeerProvider{3}: INSTALLED, TUNNEL, reqid 1, ESP SPIs: <<SPI>>
 PeerProvider{3}: AES_CBC_128/HMAC_SHA1_96/MODP_1024, 1494
bytes_i (8 pkts, 2s ago), 611 bytes_o (8 pkts, 2s ago), rekeying
in 47 minutes
 PeerProvider{3}: <dummy address>/32 === <remote server
address>/32
```

To find system messages related to StrongSWAN, use the following grep command:

```
$ grep charon /var/log/messages
#Shutdown
Jul 27 00:27:58 ServerName charon: 00[DMN] signal of type SIGINT
received. Shutting down
Jul 27 00:27:58 ServerName charon: 00[IKE] closing CHILD_SA
PeerProvider{1} with SPIs <<SPI>> (0 bytes) 5401197d_o (0 bytes)
and TS <Dummy Address>/32 === <remote server address>/32
Jul 27 00:27:58 ServerName charon: 00[IKE] sending DELETE for ESP
CHILD_SA with SPI cc95ab50
Jul 27 00:27:58 ServerName charon: 00[ENC] generating
INFORMATIONAL_V1 request 4051018568 [ HASH D ]
Jul 27 00:27:58 ServerName charon: 00[NET] sending packet: from
<StrongSWAN public IP>[500] to <remote peer address.[500] (76
bytes)
Jul 27 00:27:58 ServerName charon: 00[IKE] deleting IKE_SA
PeerProvider[1] between <StrongSWAN public IP>[<local server
address>]...<remote peer address.[<remote peer address.]
Jul 27 00:27:58 ServerName charon: 00[IKE] sending DELETE for
IKE_SA PeerProvider[1]
Jul 27 00:27:58 ServerName charon: 00[ENC] generating
INFORMATIONAL_V1 request 3332107879 [ HASH D ]
Jul 27 00:27:58 ServerName charon: 00[NET] sending packet: from
<StrongSWAN public IP>[500] to <remote peer address.[500] (92
bytes)
Jul 27 00:27:58 ServerName charon: 00[KNL] received netlink
error: Address family not supported by protocol (97)
#Startup
Jul 27 00:27:58 ServerName charon: 00[DMN] Starting IKE charon
daemon (strongSwan 5.4.0, Linux 4.9.32-15.41.amzn1.x86_64,
x86_64)
Jul 27 00:27:58 ServerName charon: 00[LIB] openssl FIPS mode(2) -
enabled
#No IPv6 – that's OK
Jul 27 00:27:58 ServerName charon: 00[NET] could not open socket:
Address family not supported by protocol
```

```
Jul 27 00:27:58 ServerName charon: 00[NET] could not open IPv6
socket, IPv6 disabled
Jul 27 00:27:58 ServerName charon: 00[KNL] received netlink
error: Address family not supported by protocol (97)
Jul 27 00:27:58 ServerName charon: 00[KNL] unable to create IPv6
routing table rule
#No certs – that's OK
Jul 27 00:27:58 ServerName charon: 00[CFG] loading ca
certificates fr'm '/etc/strongswan/ipsec.d/cace'ts'
Jul 27 00:27:58 ServerName charon: 00[LIB] opening directo'y
'/etc/strongswan/ipsec.d/cace'ts' failed: No such file or
directory
Jul 27 00:27:58 ServerName charon: 00[CFG] reading directory
failed
Jul 27 00:27:58 ServerName charon: 00[CFG] loading aa
certificates fr'm '/etc/strongswan/ipsec.d/aace'ts'
Jul 27 00:27:58 ServerName charon: 00[LIB] opening directo'y
'/etc/strongswan/ipsec.d/aace'ts' failed: No such file or
directory
Jul 27 00:27:58 ServerName charon: 00[CFG] reading directory
failed
Jul 27 00:27:58 ServerName charon: 00[CFG] loading ocsp signer
certificates fr'm '/etc/strongswan/ipsec.d/ocspce'ts'
Jul 27 00:27:58 ServerName charon: 00[LIB] opening directo'y
'/etc/strongswan/ipsec.d/ocspce'ts' failed: No such file or
directory
Jul 27 00:27:58 ServerName charon: 00[CFG] reading directory
failed
Jul 27 00:27:58 ServerName charon: 00[CFG] loading attribute
certificates fr'm '/etc/strongswan/ipsec.d/ace'ts'
Jul 27 00:27:58 ServerName charon: 00[LIB] opening directo'y
'/etc/strongswan/ipsec.d/ace'ts' failed: No such file or
directory
Jul 27 00:27:58 ServerName charon: 00[CFG] reading directory
failed
Jul 27 00:27:58 ServerName charon: 00[CFG] loading crls fr'm
'/etc/strongswan/ipsec.d/c'ls'
Jul 27 00:27:58 ServerName charon: 00[LIB] opening directo'y
'/etc/strongswan/ipsec.d/c'ls' failed: No such file or directory
Jul 27 00:27:58 ServerName charon: 00[CFG] reading directory
failed

#Interesting stuff starts here – loading secrets

Jul 27 00:27:58 ServerName charon: 00[CFG] loading secrets fr'm
'/etc/strongswan/ipsec.secr'ts'
Jul 27 00:27:58 ServerName charon: 00[CFG] loaded IKE secret for
<local server address> <remote server address>
Jul 27 00:27:58 ServerName charon: 00[LIB] loaded plugins: charon
aes des rc2 sha2 sha1 md4 md5 random nonce x509 revocation
constraints acert pubkey pkcs1 pkcs8 pkcs12 pgp dnskey sshkey pem
openssl gcrypt fips-prf gmp xcbc cmac hmac ctr ccm gcm curl attr
```

```
kernel-netlink resolve socket-default farp stroke vici updown
eap-identity eap-md5 eap-gtc eap-mschapv2 eap-tls eap-ttls eap-
peap xauth-generic xauth-eap xauth-pam xauth-noauth dhcp
Jul 27 00:27:58 ServerName charon: 00[JOB] spawning 16 worker
threads
Jul 27 00:27:58 ServerName charon: 05[CFG] received stroke: add
connecti‘n 'PeerProvi’er'
Jul 27 00:27:58 ServerName charon: 05[CFG] added configurati‘n
'PeerProvi’er'
Jul 27 00:27:58 ServerName charon: 09[CFG] received stroke:
initia‘e 'PeerProvi’er'
Jul 27 00:27:58 ServerName charon: 09[IKE] initiating Main Mode
IKE_SA PeerProvider[1] to <remote peer address.
Jul 27 00:27:58 ServerName charon: 09[ENC] generating ID_PROT
request 0 [ SA V V V ]
#Sending and RECEIVING packets – good sign
Jul 27 00:27:58 ServerName charon: 09[NET] sending packet: from
<StrongSWAN public IP>[500] to <remote peer address.[500] (228
bytes)
Jul 27 00:27:58 ServerName charon: 15[NET] received packet: from
<remote peer address.[500] to <StrongSWAN public IP>[500] (164
bytes)
Jul 27 00:27:58 ServerName charon: 15[ENC] parsed ID_PROT
response 0 [ SA V V ]
Jul 27 00:27:58 ServerName charon: 15[ENC] received unknown
vendor ID:
05:16:dc:8a:88:2c:54:a5:66:90:dc:05:bd:da:3b:9e:c8:05:e5:86:12:00
:00:00:1e:06:00:00
Jul 27 00:27:58 ServerName charon: 15[IKE] received DPD vendor ID
Jul 27 00:27:58 ServerName charon: 15[ENC] received unknown
vendor ID:
48:65:61:72:74:42:65:61:74:5f:4e:6f:74:69:66:79:38:6b:01:00
Jul 27 00:27:58 ServerName charon: 15[ENC] generating ID_PROT
request 0 [ KE No ]
Jul 27 00:27:58 ServerName charon: 15[NET] sending packet: from
<StrongSWAN public IP>[500] to <remote peer address.[500] (196
bytes)
Jul 27 00:27:58 ServerName charon: 11[NET] received packet: from
<remote peer address.[500] to <StrongSWAN public IP>[500] (196
bytes)
Jul 27 00:27:58 ServerName charon: 11[ENC] parsed ID_PROT
response 0 [ KE No ]
Jul 27 00:27:58 ServerName charon: 11[ENC] generating ID_PROT
request 0 [ ID HASH N(INITIAL_CONTACT) ]
Jul 27 00:27:58 ServerName charon: 11[NET] sending packet: from
<StrongSWAN public IP>[500] to <remote peer address.[500] (108
bytes)
Jul 27 00:27:58 ServerName charon: 07[NET] received packet: from
<remote peer address.[500] to <StrongSWAN public IP>[500] (76
bytes)
Jul 27 00:27:58 ServerName charon: 07[ENC] parsed ID_PROT
response 0 [ ID HASH ]
```

```
#Phase 1 (IKE) Established
Jul 27 00:27:58 ServerName charon: 07[IKE] IKE_SA PeerProvider[1]
established b <StrongSWAN public IP>[<local server
address>]...<remote peer address.[<remote peer address.]
Jul 27 00:27:58 ServerName charon: 07[IKE] scheduling
reauthentication in 86056s
Jul 27 00:27:58 ServerName charon: 07[IKE] maximum IKE_SA
lifetime 86236s
Jul 27 00:27:58 ServerName charon: 07[ENC] generating QUICK_MODE
request 688951572 [ HASH SA No KE ID ]
Jul 27 00:27:58 ServerName charon: 07[NET] sending packet: from
<StrongSWAN public IP>[500] to <remote peer address.[500] (316
bytes)
Jul 27 00:27:58 ServerName charon: 13[NET] received packet: from
<remote peer address.[500] to <StrongSWAN public IP>[500] (316
bytes)
Jul 27 00:27:58 ServerName charon: 13[ENC] parsed QUICK_MODE
response 688951572 [ HASH SA No KE ID ]
#Phase 2 (ISAKMP) Established
Jul 27 00:27:58 ServerName charon: 13[IKE] CHILD_SA
PeerProvider{1} established with SPIs X and TS <Dummy Address>/32
=== <remote server address>/32
Jul 27 00:27:58 ServerName charon: 13[ENC] generating QUICK_MODE
request 688951572 [ HASH ]
Jul 27 00:27:58 ServerName charon: 13[NET] sending packet: from
<StrongSWAN public IP>[500] to <remote peer address.[500] (60
bytes)
```

You can verify the proper configuration of the iptables rule to mask the source IP with the dummy address. The number of packets that have been applied to that rule should be displayed:

```
$ iptables -t nat -L -v
Chain PREROUTING (policy ACCEPT 804 packets, 66289 bytes)
 pkts bytes target     prot opt in     out     source
destination

Chain INPUT (policy ACCEPT 43 packets, 4165 bytes)
 pkts bytes target     prot opt in     out     source
destination

Chain OUTPUT (policy ACCEPT 51100 packets, 3555K bytes)
 pkts bytes target     prot opt in     out     source
destination

Chain POSTROUTING (policy ACCEPT 51076 packets, 3553K bytes)
 pkts bytes target     prot opt in     out     source
destination
  785 63924 SNAT       all  --  any    eth0    <<local
subnet>>/24        <remote server address>        to:<Dummy
Address>
```

You can use t‘e 'ip ro’te' command to ensure StrongSWAN has properly configured the route on the instance required to forward traffic into the VPN tunnel. The second and fourth lines of the following output are important:

```
$ ip route show table all
default via …
<<Remote Address>>/24 dev dummy0 proto kernel scope link src
<Dummy Address>
… dev eth0
<local subnet> dev eth0 proto kernel scope link src <StrongSWAN
public IP>
broadcast .. dev dummy0  table local  proto kernel  scope link
src <Dummy Address>
<< deleted the rest of the output >>
```

2.10 Commercial Customer Relationship Management Integration

To demonstrate integration with a commercial CRM service, ACE Direct connects to the Zendesk portal via the Enterprise Serial Bus (ESB). ACE Direct sends JSON-based messages to the RESTful Zendesk API to manage and query customer records. Zendesk is a suite of web-based products that help companies provide better customer service. ACE Direct uses Zendesk to capture Consumer complaints. When a Consumer files a complaint, the ACE Direct software uses the Zendesk web service API to create and store a complaint ticket. This API also allows queries and updates of created tickets. When an ACE Direct Agent answers a Consumer call, the application requests the ticket information from Zendesk and displays it on the ACE Direct Agent Desktop portal.

Integration with a CRM is done on a case by case basis. No two integrations are alike as the CRMs and the associated integration methods may differ.

2.11 FenDesk Customer Relationship Management

FenDesk is a server that emulates the Zendesk ticketing system for ACE Direct or any client that needs a simple ticketing system. The software only implements the subset of Zendesk RESTful API calls that ACE Direct uses; however, it is expandable to include other API calls.

FenDesk uses a simple storage scheme. It creates, updates, and returns tickets as simple JSON text files. The filename for a ticket follows the same naming convention as Zendesk: <ticketno>.json (e.g., 322.json). FenDesk offers RESTful API calls to test connectivity, add/update/delete/retrieve tickets, and search for all tickets with a specified VRS number.

2.12 Enterprise Service Bus

The ESB provides a generic method to integrate with legacy database systems as well as the diverse number of databases and unstructured data repositories in use and on the market today. ACE Direct ESB integrates with a COTS CRM service (e.g., Zendesk) as a ticketing system for the Agent to document service cases.

2.12.1 Background

Apache ServiceMix 6.1.2 is used as the service broker. Apache ServiceMix is an enterprise-class, open source, distributed ESB based on the service-oriented architecture (SOA) model. It is a project of the Apache Software Foundation and was built on the semantics and APIs of the Java Business Integration (JBI) specification JSR 208. The software is distributed under the Apache License.

The current version of ServiceMix fully supports the OSGi framework. ServiceMix is lightweight, easily embeddable, and has integrated Spring Framework support. It can be run at the edge of the network (inside a client or server), as a standalone ESB provider, or as a service within another ESB. ServiceMix is compatible with Java SE or a Java Enterprise Edition (EE) application server. ServiceMix uses ActiveMQ to provide remoting, clustering, reliability, and distributed failover. The basic frameworks used by ServiceMix are Spring and XBean.

ServiceMix comprises the latest versions of Apache ActiveMQ, Apache Camel, Apache CXF, and Apache Karaf. Additional installation features include:

- BPM engine via Activiti
- JPA support via Apache OpenJPA
- XA transaction management via JTA via Apache Aries

The ServiceMix ESB provides:

- Federation, clustering, and container-provided failover
- Hot deployment and life-cycle management of business objects
- Vendor independence from vendor-licensed products
- Compliance with the JBI specification JSR 208
- Compliance with the OSGi 4.2 specification through Apache Felix
- Support for OSGi Enterprise through Apache Aries

2.12.2 Installation Overview

First install and configure ServiceMix and its prerequisites on the host machine.

2.12.2.1 ServiceMix System Requirements

To run Apache ServiceMix itself, you will need Java Runtime Environment (JRE) 1.8.x (Java 8), and about 100 MB of free disk space for the default assembly.

If you are developing your own integration applications and OSGi bundles, you will also need:

- Java Developer Kit (JDK) 1.8.x (Java 8)
- MySQL
- Apache Maven 3.0.4 or higher

The ACRDEMO broker application depends on MySQL for a database and Maven for building the application.

2.12.2.2 Installing the JDK

Issue the following commands to install the Java 8 JDK:

```
Redhat/Fedora/CentOS Systems (SystemV):
      sudo yum install java-1.8.0-openjdk
```

Set the JAVA_HOME environment variable in the bash startup:

```
vi ~/.bashrc
```

Add the following lines to the end of the .bashrc script:

```
JAVA_HOME= /opt/jdk1.8.0_111
export JAVA_HOME
JRE_HOME=/opt/jdk1.8.0_111/jre
export JRE_HOME
PATH=$PATH:/opt/jdk1.8.0_111/bin:/opt/jdk1.8.0_111/jre/bin
export PATH
```

2.12.2.3 Installing Apache Maven

To install Apache Maven, issue the following command:

Redhat/Fedora/Centos Systems (SystemV):

```
sudo yum install maven
```

2.12.2.4 Installing MySQL

You will be able to set the password for the root account. **Note:** There is a current issue with the ESB that also requires setting the privileges for anonymous local users; accordingly, do not disable access for anonymous users. To install MySQL, issue the following commands:

Redhat/Fedora/Centos Systems (SystemV):

```
      sudo yum install unixODBC unixODBC-devel libtool-ltdl
libtool-ltdl-devel mysql-connector-odbc
      wget http://repo.mysql.com/mysql-community-release-el7-
5.noarch.rpm
sudo rpm -ivh mysql-community-release-el7-5.noarch.rpm
sudo yum install mysql-server
sudo systemctl start mysqld
sudo yum update
```

2.12.2.5 Configuring MySQL

The ServiceMix broker application for the ACR demo connects to the MySQL database; therefore, first create and configure the demo database.

Login to the MySQL command-line tool using the root account with the password you set earlier:

```
mysql -u root -p=somepassword
```

Create a database named “broker” for the ACR demo:

```
mysql> CREATE DATABASE broker;
mysql> USE broker;
```

Create the database user named “broker” for the ACR demo and set the password:

```
mysql> CREATE USER 'broker'@'localhost' IDENTIFIED BY
'somepassword';
```

Set the permissions for the database user “broker”. **Note:** This should be tuned to only grant the necessary privileges:

```
mysql> GRANT ALL PRIVILEGES ON broker.* TO 'broker'@'%' WITH
GRANT OPTION;
```

Note: There is a current issue with the ESB that also requires setting the privileges for anonymous local users. Privileges must be set for anonymous users:

```
mysql> GRANT ALL PRIVILEGES ON broker.* TO ''@'localhost' WITH
GRANT OPTION;
```

Create the “users” table:

```
mysql> CREATE TABLE `users` (
`user_id` bigint(20) NOT NULL,
`user_name` varchar(50) DEFAULT NULL,
`user_description` varchar(45) DEFAULT NULL,
`user_phone` varchar(20) DEFAULT NULL,
`user_address` varchar(50) DEFAULT NULL,
`user_account` varchar(50) DEFAULT NULL,
PRIMARY KEY (`user_id`));
```

Populate the “users” table with test records. **Note:** The user_id values need to correspond to IDs of Zendesk users:

```
mysql> INSERT INTO `users` VALUES
(3770168798,'John Doe ','Some Details','555-555-
1111',NULL,'121212'),
(4060741111,'Jane Doe','No Description','222-111-1111','12341
Main Street','12345671'),
(4758821111,'Tim','No Description','555-666-7777','','5656565');
```

2.12.2.6 Downloading and Building Broker Application

Set up SSH keys to access the git repository according to these instructions:

- Adding a new SSH key to your GitHub account

Create a folder for cloning the broker source code and navigate to that folder:

```
mkdir ~/code && cd ~/code
```

Clone the broker git repository from esb.git.

Navigate to the broker code folder:

```
cd camel-rest-proxy-blueprint/
```

Modify the applications blueprint file for your environment. The blueprint is configured to process messages intended for Zendesk. You will need to modify the blueprint to specify your Zendesk hostname and any proxy settings if you are accessing Zendesk from the ESB through a proxy.

Build the broker application with Maven:

```
mvn clean install
```

2.12.2.7 Downloading and Installing Apache ServiceMix

Apache ServiceMix 6.1.2 is available under the Apache License v2 and can be downloaded from http://servicemix.apache.org/downloads/servicemix-6.1.2.html.

Create and navigate to a folder where the downloaded zip file will be placed:

```
mkdir ~/dev-tools && cd ~/dev-tools
```

Download and uncompress the zip file. For example:

```
wget
http://mirror.cc.columbia.edu/pub/software/apache/servicemix/serv
icemix-6/6.1.2/apache-servicemix-6.1.2.zip
unzip apache-servicemix-6.1.2.zip
```

2.12.2.8 Running and Configuring ServiceMix

In a command shell, navigate to the ServiceMix bin directory (e.g., ~/dev-tools/apache-servicemix-6.1.2):

```
cd ~/dev-tools/apache-servicemix-6.1.2/bin
```

Start ServiceMix:

```
./servicemix
```

Install the following features:

```
karaf@root>feature:install jdbc
karaf@root>feature:install pax-jdbc-mysql
karaf@root>feature:install camel-jsonpath
```

```
karaf@root>feature:install camel-jetty
karaf@root>feature:install camel-jdbc
karaf@root>feature:install camel-http4
```

You may need to download the pax-jdbc artifact from the Maven repository if the pax-jdbc-mysql install does not work:

```
karaf@root>feature:repo-add pax-jdbc 0.6.0
```

Create the JDBC connection to the MySQL database:

Note: Depending on your version of jdbc, you will need either the command "jdbc:ds-create" or "jdbc:create". Type <tab> to print a list of available commands and find the one you need:

```
karaf@root>jdbc:ds-create -dn  mysql -url
jdbc:mysql://localhost:3306/demo?user=broker&password=somepasswor
d mySqlDataSource
karaf@root>jdbc:create -d  mysql -t MySQL -url
jdbc:mysql://localhost:3306/broker -u broker -p somepassword
mySqlDataSource
```

Check that the JDBC connection was created:

```
karaf@root>jdbc:datasources
```

Another way to check the database connection is to issue a query:

```
karaf@root>jdbc:query jdbc/mySqlDataSource "select * from users"
```

Install the broker application. The application will be installed as an OSGI bundle:

```
karaf@root>bundle:install -s mvn:org.apache.camel/camel-rest-
proxy-blueprint/2.16.3
```

Check that the broker was installed. The bundle should be the last bundle in the list and its status should be ACTIVE:

```
karaf@root>bundle:list
```

2.12.2.9 Install and Start ServiceMix as a Service

Start the ServiceMix if is not already started. Issue the following commands:

```
karaf@root>feature:install wrapper
karaf@root>wrapper:install -s AUTO_START -n KARAF -d Karaf -D
"Karaf Service"
```

A message similar to the following will be displayed:

> Setup complete. You may wish to tweak the JVM properties in the wrapper configuration file before installing and starting the service:

```
~/dev-tools/apache-servicemix-6.1.2/etc/KARAF-wrapper.conf
```

Redhat/Fedora/Centos Systems (SystemV):

To install the service:

```
$ ln -s ~//dev-tools/apache-servicemix-6.1.0/bin/KARAF-
service /etc/init.d/
$ chkconfig KARAF-service --add
```

To start the service when the machine is rebooted:

```
$ chkconfig KARAF-service on
```

To disable starting the service when the machine is rebooted:

```
$ chkconfig KARAF-service off
```

To start the service:

```
$ service KARAF-service start
```

To stop the service:

```
$ service KARAF-service stop
```

To uninstall the service:

```
$ chkconfig KARAF-service --del
$ rm /etc/init.d/KARAF-service
```

For systemd compliant Linux:

To install the service (and enable at system boot):

```
$ systemctl enable~/dev-tools/apache-servicemix-
6.1.2/bin/KARAF.service
```

To start the service:

```
$ systemctl start KARAF
```

To stop the service:

```
$ systemctl stop KARAF
```

To check the current service status:

```
$ systemctl status KARAF
```

To see service activity journal:

```
$ journalctl -u KARAF
```

To uninstall the service (and disable at system boot):

```
$ systemctl disable KARAF
```

Exit the ServiceMix/Karaf shell, by shutting down ServiceMix:

```
karaf@root>shutdown
```

Install the ServiceMix service:

```
sudo ln -s ~/dev-tools/apache-servicemix-6.1.2/bin/KARAF-service
/etc/init.d/
```

Set the service to start when the machine is rebooted:

```
sudo update-rc.d KARAF-service defaults
```

Start the ServiceMix service:

```
sudo /etc/init.d/KARAF-service start
```

To log back into ServiceMix once the service is started, issue the following commands:

```
cd ~/dev-tools/apache-servicemix-6.1.2/bin
./client
```

To exit the ServiceMix shell without shutting down the service, type ^D (i.e., Ctrl-D). **Note:** If you type shutdown in ServiceMix shell, the entire service will be shut down.

2.12.3 Editing blueprint.xml Application File

The blueprint.xml file is provided with placeholders that must be edited for your specific environment.

2.12.3.1 Update Zendesk Hostname

The blueprint.xml file has placeholders for the Zendesk hostname. You will need to provide your Zendesk hostname wherever you see the placeholder “<insert CRM hostname>”.

2.12.3.2 Enable / Disable Proxy

If there is a proxy between the ESB and Zendesk, you may need to add the following parameters to the set of parameters specified for each instance of the Zendesk endpoint. For example, you may need to replace:

```
/api?bridgeEndpoint=true&amp;throwExceptionOnFailure=false
with:
```

```
/api?bridgeEndpoint=true&amp;proxyAuthHost=<replace with proxy
ip>&amp;proxyAuthPort=<replace with proxy
port>&amp;proxyAuthScheme=http4&amp;
throwExceptionOnFailure=false
```

substituting your proxy host and port settings for the placeholders.

After editing the blueprint file, rebuild the application using Maven:

```
cd ~/code/camel-rest-proxy-blue-print/
mvn clean install
cd ~/dev-tools/apache-servicemix-6.1.2/bin
./client
karaf@root>bundle:install -s mvn:org.apache.camel/camel-rest-
proxy-blueprint/2.16.3
```

Make sure the bundle is successfully installed with no errors and active.

2.12.4 Testing the Broker Application

At this point, all of the code for the Broker application is contained in the OSGI Blueprint file (i.e., blueprint.xml), which can be viewed in the Github ESB broker esb application.

If the Broker application is running in ServiceMix running, you can check the application by using curl at the command line. For example:

```
$ curl -u
username@hostname/token:hLLUnPzJtpvMZ5WnntN3wCneKHkl20kP0Hhn5NrD
http://localhost:9090/api/v2/users/me.json  --insecure
```

where username is a Zendesk user account and hostname is your Zendesk host name.

You can check the status and statistics of the main Broker route in the ServiceMix/Karaf shell:

```
karaf@root>camel:route-info rest-http-zendesk-mysql-demo
```

Here is example output from the camel:route-info command:

```
Camel Route rest-http-zendesk-mysql-demo
Camel Context: camel-1
State: Started
State: Started

Statistics
Exchanges Total: 2
Exchanges Completed: 2
Exchanges Failed: 0
Exchanges Inflight: 0
Min Processing Time: 240 ms
Max Processing Time: 494 ms
Mean Processing Time: 367 ms
Total Processing Time: 734 ms
Last Processing Time: 240 ms
```

```
 Delta Processing Time: -254 ms
Start Statistics Date: 2016-07-19 14:43:44
Reset Statistics Date: 2016-07-19 14:43:44
First Exchange Date: 2016-07-19 15:12:15
Last Exchange Date: 2016-07-19 15:12:3
```

2.13 NGINX

NGINX is an open source HTTP and reverse proxy server. For ACE Direct, only the reverse proxy component is used. A reverse proxy allows ACE Direct to hide internal topology (ports and script names) from users.

Diagram 1**Error! Reference source not found.** shows the mapping between public-facing and internal URLs performed by NGINX. The left side of the figure shows the URLs available to the public via HTTPS. Each URL maps to another path and port internally on the Node.js server. In this example, NGIX is mapping port 443 on the public-facing interface to ports 8005 and 8081 on the internal side.

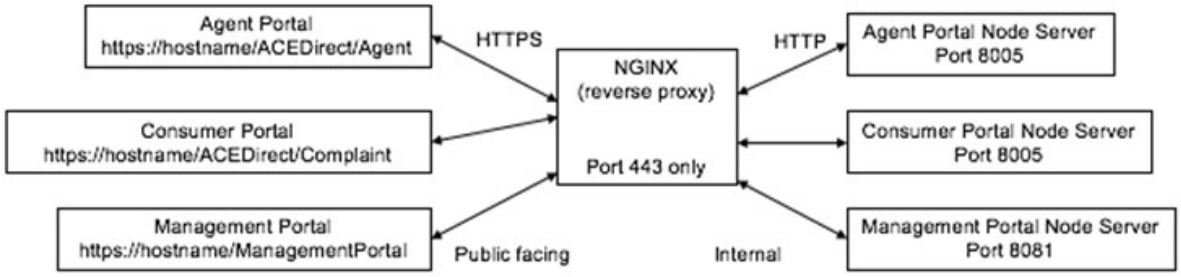

Diagram 1. Internal and External Paths with NGINX

In ACE Direct, NGINX is configured to work with OpenAM, directing incoming connections to the correct location based on URL and authentication level. Diagram 2 shows the relationship between NGINX and OpenAM and the mapping between public-facing URLs (to the left or outside of the firewall) and the internal processes.

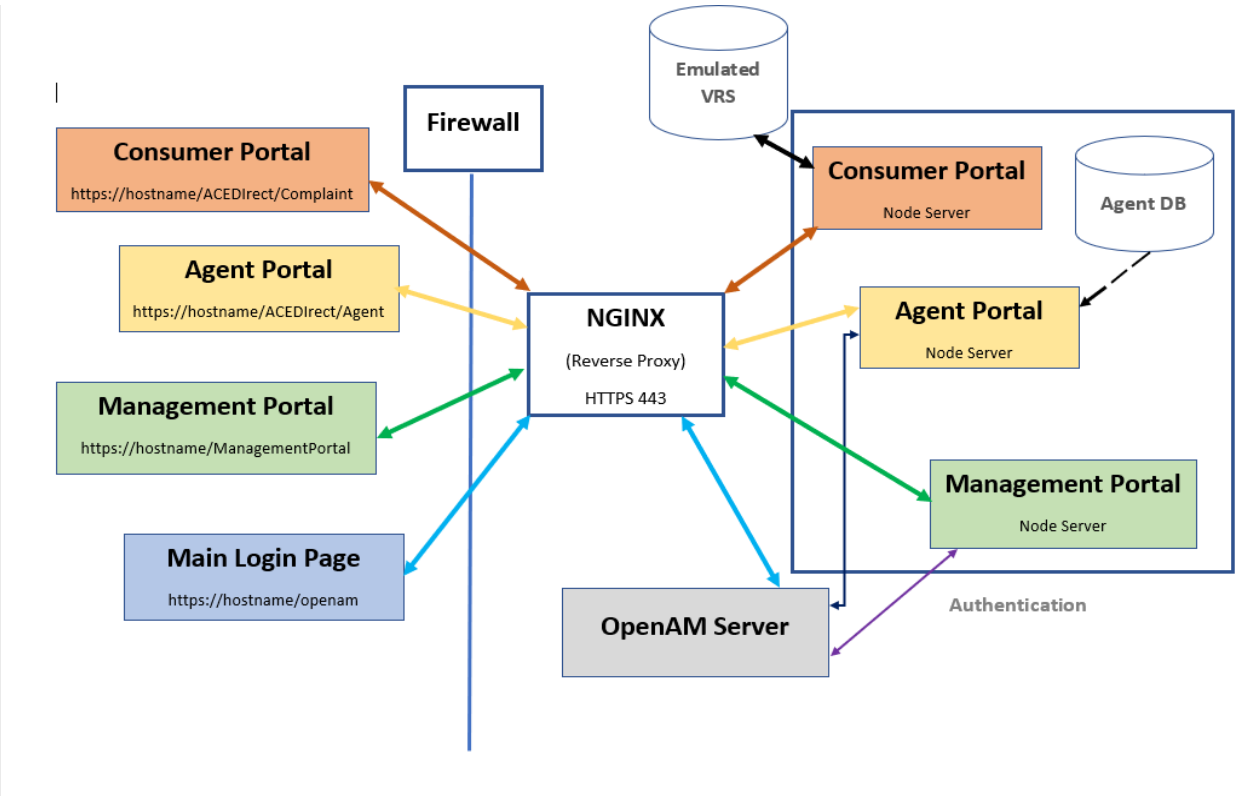

Diagram 2. NGINX and OpenAM Integration

2.14 Redis

Redis is an open source, in-memory, key-value data store. In ACE Direct, Redis is used to store state data that were previously stored in memory in the Node.js server. Some of the data stored in Redis include Agent status (active or away), maps of Agent extensions, and Agent information.

Redis offers advantages over in-memory storage. For example, Redis allows several applications to share the same data. This ensures that all applications have the same view of system state. Another advantage is that Redis supports different data formats, which means it can store more than just typical key-value pairs. The most widely used data types are strings; however, Redis also supports lists, sets, sorted sets, and hashes. Because Redis can also write data to disk, it can maintain state after a system restart. Additional information about Redis can be found here: https://redis.io.

2.15 Data Logger

The Data Logger is a tool developed by the MITRE team to support ACE Direct interoperability testing and debugging. The Data Logger creates packet capture (pcap) and Asterisk logs for a test session to support debugging. In this case, a test session consists of one or more video phone calls.

The Data Logger stores log data (Asterisk and pcap) in an Amazon AWS S3 bucket and metadata about the test (user, start and stop time, etc.) is stored in a MySQL database. This

manual assumes that a MySQL server is installed and accessible to the Data Logger. In addition, this manual assumes the instantiation of the database used by Asterisk to log CDRs.

2.15.1 Configuring MySQL

Open the MySQL shell to create the database and account for access:

```
mysql -uroot -p
mysql> create database data_logger;
```

Create the database accounts and issue the grants:

```
mysql> grant all on data_logger.* to '<user>'@'<MySQL host>'
identified by 'password here';
```

Exit the MySQL shell:

```
mysql> exit;
```

The next step is to instantiate the databases. The MySQL schema (data_logging.sql) is included as part of the public release download for the Data Logger.

From the command prompt (not the MySQL prompt), type the following and enter the password when prompted:

```
mysql -u<user> -p data_logger < data_logging.sql
```

2.15.2 Installing the Node Server

- Clone the data-logger repository from GitHub
- Install Node.js >= version 8.x
- Type `` `cd data-logger` ``
- Install the required Node.js modules:
  - `npm install`
- Install PM2 >= v2.4.6 by running:
  - `npm install -g pm2`

2.15.3 SSL Configuration

Data Logger software uses SSL, which requires a valid key and certificate. Enable SSL by specifying ‘https’ as the protocol value in the config.json file. The location of the SSL key and certificate can be specified in the config.json by using the ssl:cert and ssl:key parameters in the form of folder/file (e.g., ssl/mycert.pem and ssl/mykey.pem).

2.15.4 Server Software Configuration

The data-logger directory contains a file named config.json, which contains the configuration parameters used by the server. The configuration parameters and the definitions are:

```
{
    "debuglevel": "ALL | TRACE | DEBUG | INFO | WARN ERROR |
FATAL | OFF",
    "port": server listen port #,
     "logdirectory": "/home/centos/data_logger/logs",
    "ssl" : {
      "key": "path to key",
      "cert": "path to cert"
    },
    "session": {
      "name":"session name",
      "resave": "true",
      "saveUninitialized": "true",
      "secret": "secret here",
      "secure": "true"
    },
    "mysql": {
      "host": "localhost",
      "user": "mysql username",
      "password": "mysql password",
      "database": "<database_name>"
    },
    "asterisk": {
      "port": 5038,
      "host": "IP address of localhost",
      "user": "asterisk username",
      "password": "asterisk password"
    }
}
```

2.15.5 Starting the Server

To start the Data Logger Server:

- `cd data-logger`
- `npm start or node bin/www`

2.15.5.1 Accessing the site

To access the Data Logger, go to the following URL:

- https://host:port/

2.16 SIP Proxy Server – Kamailio

2.16.1 Introduction

Kamailio is a free, open source software framework licensed under GPL that functions as a SIP Proxy Server able to handle thousands of call setups per second. Kamailio has been adopted and integrated into ACE Direct to build a large-scale, full-duplex streaming video and voice call center for real-time communications. It is featured with capabilities to support WebRTC-based

VoIP (video and voice) calls; presence; Real-Time Text (RTT), i.e., Instant Messaging; and videomail. Moreover, it can be easily used for scaling up SIP-to-PSTN gateways, PBX systems (such as Asterisk), and media servers.

2.16.2 Installation and Configuration

Kamailio will be installed on a CentOS 7 server. Typically, each of the services used by ACE Direct, such as Kamailio, Asterisk, node.js, TURN, STUN, is hosted on its own server; however, they can also coexist on the same server.

The main install script (AD_kamailio-install.sh) will install and configure Kamailio for ACE Direct on a CentOS 7 server. This script, which is available at https://github.com/mitrefccace/kamailio, can be used to quickly install and configure the SIP Proxy Server and the supporting RTP Proxy Server (rtpengine). This installation process will be facilitated by running the main script in conjunction with the supporting scripts, sequel, and configuration files in this same repo.

Generally, the process will be straightforward to start and streamlined by running the main shell script at the directory where it is located, as shown, using "default" configuration values when prompted during the execution:

```
$ sudo ./AD_kamailio-install.sh
```

You will be required to know the IP address of the Asterisk server. Other fields that you will be prompted for give you a value to default to.

Default values have been used for the ongoing lab-based integration and associated tests on the Health FFRDC testbed. For any new installation and configuration on different deployment environments, however, different values should be used and supplied by the installers for their specific deployment.

2.17 Kurento Media Server

2.17.1 Introduction

Kurento is a WebRTC media server and set of client APIs that provide advanced media processing capabilities. Kurento has been adopted into ACE Direct to provide transcoding, group communications, recording, monitoring, and bandwidth controls.

2.17.2 Installation and Configuration

The Kurento media server requires an Ubuntu Linux 18.04 LTS host. Follow these steps to perform the initial installation.

```
## Update *apt* and install *kurento*
sudo apt-get update

## Import the key from Kurento TEAM
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
5AFA7A83
```

```
## Set DISTRO to "bionic" for Ubuntu 18.04, "xenial" for Ubuntu
16.04
DISTRO="bionic"

## Adding repo to source
sudo tee "/etc/apt/sources.list.d/kurento.list" >/dev/null << EOF
# Kurento Media Server - Release packages
deb [arch=amd64] http://ubuntu.openvidu.io/6.11.0 $DISTRO kms6
EOF
unset DISTRO

## Install kurento-media-server
sudo apt-get update && sudo apt-get install --yes kurento-media-
server
```

The Kurento media server uses a combination certificate that combines the private key, certificate, and intermediate CA:

```
cat key.pem fullchain.pem > server.pem
```

Once the Kurento media server is installed, configuration files need to be modified.

- /etc/kurento/kurento.conf.json
  - The main configuration file to provide settings of Kurento Media Server
  - Defines the WebSocket(WS)/WebSocket Secure(WSS) ports and path to combo certificates
  - Please refer to this link for more details
- /etc/kurento/modules/kurento/WebRtcEndpoint.conf.ini
  - Specific parameters for WebRtcEndpoint
  - Please refer to the repository directory in acedirect-kurento/confs/kurento/ examples
- /etc/kurento/modules/kurento/MediaElement.conf.ini
  - Generic parameters for all types of MediaElements
  - Please refer to the Kurento GitHub page for examples
- /etc/kurento/modules/kurento/SdpEndpoint.conf.ini
  - Audio/video parameters for SdpEndpoints (i.e., WebRtcEndpoint and RtpEndpoint)
  - Please refer to the Kurento GitHub page for examples
- /etc/kurento/modules/kurento/HttpEndpoint.conf.ini
  - Specific parameters for HttpEndpoint
  - Please refer to the Kurento GitHub page for examples

2.18 Signaling Server

2.18.1 Introduction

The signaling server is a Node.js server that provides an API to the Kurento media server to make calls using Asterisk.

2.18.2 Installation and Configuration

The signaling server should be installed on the same server instance as the Kurento media server. The prerequisites for the signaling server are:

- MySQL
  - The MySQL server previously install can be used to support the signaling server
- Node.js
  - Version 10.16.0 LTS or Version 8.16.0

To install the signaling server, follow these steps:

- Clone the repository to the desired location
- cd acedirect-kurento
- npm run build
- Configure these files:
  - confs/kurento/WebRtcEndpoint.conf.ini
  - src/config/db.js
  - src/config/development.json
- npm run sequelize db:migrate
- npm run dev
- Open this URL in a WebRTC compatible browser: `https://localhost:8443`

2.19 Busylight Installation

2.19.1 Installation on Windows

1. Download the Busylight UI to your computer Desktop: lightserver.jar.
2. Verify Oracle Java Runtime version:
   - From your computer, open a *Command Prompt* and type: java -version.
   - Verify that it says *Java(TM) SE Runtime Environment* 1.8 or newer. OpenJDK and Node.js versions of the BusyLight UI are also available.
   - If it is not installed, download Java 1.8 here and install it on your computer.

2.19.2 Using the Busylight With ACE Direct

1. Plug the *Kuando Busylight* into an available USB port on your computer. The light will flash red, green, blue twice, then turn off.
2. Double-click the lightserver.jar file. If prompted, open the file as a *Java(TM) Platform SE binary*.
3. The *Busylight - ACE Direct* UI will start, and the Busylight device will flash rainbow colors and turn off.
4. From a browser, log into ACE Direct.
   - If prompted, allow the browser to use your camera.
   - A Busylight dialog will appear. Click the Test Busylight Connection button.
   - A new browser tab will appear. Click Advanced and click Proceed to localhost (unsafe).
   - BusyLight access will be granted. Close this tab.
   - From the Agent Portal, close the Busylight dialog.
5. You are now ready to use the Busylight device with ACE Direct.

2.20 Installing ACE Direct with Docker

Docker is a Linux-based container management system. Some of the components of the ACE Direct system can be run using Docker.

A full deployment using Docker is considered a future capability.

2.20.1 Docker Overview

Docker Containers are special Linux processes, started from Docker Images, and managed by the Docker Engine, which must be installed on each host. A host may run zero or more Docker Containers (depending on capacity).

Docker images are built by starting with a base image and layering software, data, and configurations on top. Layers are described using a special file called a Dockerfile. There are Dockerfiles for most ACE Direct components. For example, the “acedirect” container is built using the “node” image (corresponding to Docker’s official NodeJS image). ACE Direct specific code is then layered on top of the node image to produce the acedirect image.

Each Dockerfile is used to build a corresponding Docker Image. The images can then be executed using the Docker Engine.

The ‘docker’ command is used to send commands to the Docker Engine. These commands can be used to build images, start containers, stop containers, check status, etc.

2.20.2 Limitations

At present, ACE Direct containers have only been tested in a single host configuration. The Kurento media server, signaling server, and Kamailio SIP proxy have not been tested in Docker containers.

2.20.3 Prerequisites

Docker adds resource and management overhead over and above the ACE Direct requirements. ACE Direct has been tested on AWS EC2 instances having at least 8 GB of RAM and 160 GB of disk.

2.20.4 Docker Installation

The base Docker package must be installed following the instructions at docs.docker.com.

Some corporate, government, or educational environments may require additional customization of the Docker installation to properly access the Internet through proxy servers. As described in the ACE Direct documents, the Docker engine must have access to the Internet to download pre-built Docker official images.

2.20.5 Configuring ACE Direct Containers

Each ACE Direct component requires at least two Docker bind volumes. These are persistent storage areas, meaning they will persist even if the container is stopped or restarted. By default, storage in a container is not persistent.

For each component, there is a "volumes" directory. Within that directory, there is at least a "logs" directory containing component-specific logs. Other directories are created manually before running components. The docker-compose.yml file contains the mapping of directories to Docker bind volumes.

There is an ".env-template" file that must be copied to ".env" in the top-level source directory. It is necessary to edit that file and set appropriate configuration settings for each property. These properties are passed into the Docker container as environment variables as well as made available to the docker-compose build processes.

2.20.6 Building Docker Images

ACE Direct containers are not shipped as binary images. They must be built in your environment.

The build process is managed by docker-compose. This mean the .env file must be customized for your environment prior to launching a build.

To build an image, type:

```
$ cd container
$ docker-compose build
```

There should be no errors.

2.20.7 Running ACE Direct Using Docker

This section covers the prerequisites for running and stopping ACE Direct using Docker.

2.20.7.1 Prerequisites

To execute Docker commands, a user must be either part of the “docker” Linux group or have “sudo” access to the “docker” command.

Next, each ACE Direct component must have been built using docker-compose (as described in subsection 2.14.6).

2.20.7.2 Running

Each container must be started in the correct order (from back to front). That is, databases are started first, then application services, then the NGINX and Asterisk servers. This ensures that containers do not start before their dependencies.

A script is provided, called “start-all.sh”, to perform this task. This script can be customized to suit specific needs.

2.20.7.3 Stopping

To stop all the containers, run the “stop-all.sh” script. This script stops each container from front to back, starting with NGINX and ending with databases and/or backend services.

2.20.7.4 Monitoring

Monitoring infrastructure and processes is a complex topic. The following guidance covers only the basics that apply to Docker.

For basic monitoring of Docker, use:

```
$ sudo docker ps
```

For continuous monitoring, similar to Linux “top”, use:

```
$ sudo docker stats
Press ^C to cancel monitoring.
```

2.20.7.5 Troubleshooting

There should always be one docker container running for each ACE Direct component, with the exception of Asterisk and OpenAM. At the time of writing, six containers should be running.

If a container refuses to start, check that there are no other containers listening on the same IP address and port.

3. Conclusion

ACE Direct is a large system of simple and complicated components. This installation guide simplifies the installation, configuration, and deployment of ACE Direct. Where appropriate, this guide refers to other helpful documents online and in the GitHub repositories.

For more information or to post an issue or question, visit the FCC ACE Direct GitHub repo (https://github.com/FCC/ACEDirect).

Acronyms

ACE	Accessible Communications for Everyone
ACR	Auto Call Routing
ADA	Americans with Disabilities Act
AMA	Automatic Message Accounting
AMI	Asterisk Management Interface
API	Application Programming Interface
ASL	American Sign Language
AWS	Amazon Web Services
CA	Communication Assistant, Certificate Authority
CDR	Call Detail Record
COE	Center of Expertise
COTS	Commercial Off-the-Shelf
CRM	Customer Relationship Management
CSR	Customer Service Record
CSV	Comma Separated Value
DNS	Domain Name System
DVC	Direct Video Calling
EIP	Elastic Internet Protocol
ENUM	E.164 Number to URI Mapping
ESB	Enterprise Service Bus
FCC	Federal Communications Commission
FQDN	Fully Qualified Domain Name
FFRDC	Federally Funded Research and Development Center
GUI	Graphical User Interface
HSTS	HTTP Strict Transport Security
HTTPS	HyperText Transport Protocol Secure
iTRS	Internet Telecommunications Relay Service
I/O	Input/Output
IP	Internet Protocol

IPSec	Internet Protocol Secure
Java EE	Java Enterprise Edition
JBI	Java Business Integration
JDBC	Java Database Connectivity
JDK	Java Developer Kit
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
NAT	Network Address Translation
NGINX	A web server which can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache
ODBC	Open Database Connectivity
OpenAM	Open Access Management
OS	Operating System
OSGi	OSGi Alliance (formerly Open Systems Group Initiative)
PBX	Private Branch Exchange
POC	Proof of Concept
PSTN	Public Switch Telephone Network
QoS	Quality of Service
REST	Representational State Transfer
RFC	Request for Comment
RTCP	RTP (Real-time Transport Protocol) Control Protocol
RTT	Real-Time Text
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SOA	Service-Oriented Architecture
SQL	Structured Query Language
SRTP	Secure Real-Time Transport Protocol
SSH	Secure Shell
SSL	Secure Socket Layer
STUN	Session Traversal Utilities for NAT
TCP	Transmission Control Protocol

TURN	Traversal Using Relay NAT
UDP	User Datagram Protocol
URD	User Registration Database
URI	Uniform Resource Identifier
URL	Universal Resource Locator
VM	Virtual Machine
VoIP	Voice Over Internet Protocol
VPC	Amazon's Virtual Private Cloud
VPN	Virtual Private Network
VRS	Video Relay Service
VyOS	Open source network operating system
WebRTC	Web Real-Time Communication

Notice

For further information, please contact The MITRE Corporation, Contracts Management Office, 7515 Colshire Drive, McLean, VA 22102-7539, (703) 983-6000.