



# AKAMAI MEDIA DELIVERY SOLUTIONS

## Course Overview and Agenda

March 2023



```
fmt.Fprintf(w, "ACTIVE"); } else { fmt.Fprintf(w, "INACTIVE"); } return; case <- timeout: fmt.Fprintf(w, "TIMEOUT"); }); log.Fatal(http.ListenAndServe(":1337", nil)); } package main; import ( "fmt"; "html"; "log"; "net/http"; "strconv"; "strings"; "time" ); type ControlMessage struct { Target string; Count int64; }; func main() { controlChannel := make(chan ControlMessage); workerCompleteChan := make(chan bool); statusPollChannel := make(chan chan bool); workerActive := false; go admin(controlChannel, statusPollChannel, respChan); go workerActive; case msg := <- controlChannel: workerActive = true; go doStuff(msg, workerCompleteChan); case status := <- workerCompleteChan: workerActive = status; }); func admin(cc chan ControlMessage, stcp.HandleFunc("/admin", func(w http.ResponseWriter, r *http.Request) { hostTokens := strings.Split(r.Host, ":"); r.ParseForm(); count, err := strconv.Atoi(r.FormValue("count"), 10, 64); if err != nil { fmt.Fprintf(w, err.Error()); return; r.FormValue("target"), Count: count); cc <- msg; fmt.Fprintf(w, "Control message issued for Target %s, count %d", html.EscapeString(r.FormValue("target")), count); }); http.HandleFunc("/status", func(w http.ResponseWriter, r *http.Request) { reqChan := reqChan; timeout := time.After(time.Second); select { case result := <- reqChan: if result { fmt.Fprintf(w, "ACTIVE"); } else { fmt.Fprintf(w, "INACTIVE"); }; return; case <- timeout: fmt.Fprintf(w, "TIMEOUT"); }); log.Fatal(http.ListenAndServe(":1337", nil)); } package main; import ( "fmt"; "html"; "log"; "net/http"; "strconv"; "strings"; "time" ); type ControlMessage struct { Target string; Count int64; }; func main() { controlChannel := make(chan ControlMessage); workerCompleteChan := make(chan bool); statusPollChannel := false; go admin(controlChannel, statusPollChannel); for { select { case respChan := <- statusPollChannel: respChan <- workerActive; case msg := <- controlChannel: workerActive = true; go doStuff(msg, workerCompleteChan); case status := <- workerCompleteChan: workerActive = status; }); func admin(cc chan ControlMessage, statusPollChannel chan chan bool) { http.HandleFunc("/admin", func(w http.ResponseWriter, r *http.Request) { hostTokens := strings.Split(r.Host, ":"); r.ParseForm(); count, err := strconv.Atoi(r
```

## Course Overview

The Akamai Media Delivery Solutions course is designed to familiarize participants with the Akamai media delivery offering. During the course the participants learn how to configure, test, and fine tune Akamai media delivery solutions. The training also provides an overview of reporting, basic media APIs, and common troubleshooting guidelines for stream delivery. This course includes live demonstrations and hands-on exercises to enhance user experience.

## Objectives

At the end of this course, you will be able to:

- Describe the Akamai platform and media product portfolio.
- Explain compression and codecs.
- Explain live and on-demand media workflows.
- Implement and test live and on-demand configurations.
- Describe streaming media security risks.
- Use media reports.
- Understand the basics of media APIs.
- Troubleshoot common streaming issues.

```
fmt.Fprintf(w, "ACTIVE"); } else { fmt.Fprintf(w, "INACTIVE"); }; return; case <- timeout: fmt.Fprintf(w, "TIMEOUT");}); log.Fatal(http.ListenAndServe(":1337", nil)); },package main; import ( "fmt"; "html"; "log"; "net/http"; "strconv"; "strings"; "time"); type ControlMessage struct { Target string; Count int64; }; func main() { controlChannel := make(chan ControlMessage);workerCompleteChan := make(chan bool); statusPollChannel := make(chan chan bool); workerActive := false;go admin(controlChannel, statusPollChannel, respChan := workerActive; case msg := <-controlChannel: workerActive = true; go doStuff(msg, workerCompleteChan); case status := <- workerCompleteChan: workerActive = status; }); func admin(cc chan ControlMessage, stp.HandleFunc("/admin", func(w http.ResponseWriter, r *http.Request) { hostTokens := strings.Split(r.Host, ":"); r.ParseForm(); count, err := strconv.Atoi(r.FormValue("count"), 10, 64); if err != nil { fmt.Fprintf(w, err.Error()); return; r.FormValue("target"), Count: count); cc <- msg; fmt.Fprintf(w, "Control message issued for Target %s, count %d", html.EscapeString(r.FormValue("target")), count); }); http.HandleFunc("/status", func(w http.ResponseWriter, r *http.Request) { reqChan := reqChan;timeout := time.After(time.Second); select { case result := <- reqChan: if result { fmt.Fprintf(w, "ACTIVE"); } else { fmt.Fprintf(w, "INACTIVE"); }; return; case <- timeout: fmt.Fprintf(w, "TIMEOUT");}); log.Fatal(http.ListenAndServe(":1337", nil)); },port ( "fmt"; "html"; "log"; "net/http"; "strconv"; "strings"; "time" ); type ControlMessage struct { Target string; Count int64; }; func main() { controlChannel := make(chan ControlMessage);workerCompleteChan := make(chan bool); statusPollChannel := false;go admin(controlChannel, statusPollChannel); for { select { case respChan := <- statusPollChannel: respChan <- workerActive; case msg := <-controlChannel: workerActive = true; go doStuff(msg, workerCompleteChan); case status := <- status; }); func admin(cc chan ControlMessage, statusPollChannel chan chan bool) {http.HandleFunc("/admin", func(w http.ResponseWriter, r *http.Request) { hostTokens := strings.Split(r.Host, ":"); r.ParseForm(); count, err := strconv.Atoi(r
```

## Agenda

The Akamai Media Delivery Solutions course curriculum can be delivered either as:

- CLASSROOM TRAINING: 2 day (8 hours),
- ONLINE TRAINING: 3 days (4,5 hours each).

The agenda for this training is listed below.

```

fmt.Fprintf(w, "ACTIVE"); } else { fmt.Fprintf(w, "INACTIVE"); }; return; case <- timeout: fmt.Fprintf(w, "TIMEOUT");}); log.Fatal(http.ListenAndServe(":1337", nil)); };package main; import ( "fmt"; "html"; "log"; "net/http"; "strconv"; "strings"; "time"); type ControlMessage struct { Target string; Count int64; }; func main() { controlChannel := make(chan ControlMessage);workerCompleteChan := make(chan bool); statusPollChannel := make(chan chan bool); workerActive := false;go admin(controlChannel, statusPollChannel, respChan); case msg := <-controlChannel: workerActive = true; go doStuff(msg, workerCompleteChan); case status := <- workerCompleteChan: workerActive = status; }); func admin(cc chan ControlMessage, statusPollChannel chan chan bool, respChan chan ControlMessage) { http.HandleFunc("/admin", func(w http.ResponseWriter, r *http.Request) { hostTokens := strings.Split(r.Host, ":"); r.ParseForm(); count, err := strconv.Atoi(r.FormValue("count"), 10, 64); if err != nil { fmt.Fprintf(w, err.Error()); return; } r.ParseForm(); count, err := strconv.Atoi(r.FormValue("count"), 10, 64); if err != nil { fmt.Fprintf(w, err.Error()); return; } r.ParseForm(); count, err := strconv.Atoi(r.FormValue("target"), Count: count); cc <- msg; fmt.Fprintf(w, "Control message issued for Target %s, count %d", html.EscapeString(r.FormValue("target")), count); }); http.HandleFunc("/status", func(w http.ResponseWriter, r *http.Request) { reqChan := reqChan;timeout := time.After(time.Second); select { case result := <- reqChan: if result { fmt.Fprintf(w, "ACTIVE"); } else { fmt.Fprintf(w, "INACTIVE"); }; return; case <- timeout: fmt.Fprintf(w, "TIMEOUT");}); log.Fatal(http.ListenAndServe(":1337", nil)); }; } type ControlMessage struct { Target string; Count int64; }; func main() { controlChannel := make(chan ControlMessage);workerCompleteChan := make(chan bool); statusPollChannel := <- statusPollChannel; for { select { case respChan := <- statusPollChannel: respChan <- workerActive; case msg := <-controlChannel: workerActive = true; go doStuff(msg, workerCompleteChan); case status := <- workerCompleteChan: workerActive = status; }); func admin(cc chan ControlMessage, statusPollChannel chan chan bool) { http.HandleFunc("/admin", func(w http.ResponseWriter, r *http.Request) { hostTokens := strings.Split(r.Host, ":"); r.ParseForm(); count, err := strconv.Atoi(r

```

Duration (min)	Module Name & Description
90	<p><b>MODULE 1: INTRODUCTION</b></p> <p>This module describes the basics of the Akamai Intelligent Edge Platform, shoes how to navigate through Akamai Control Center and explain what Property Manager is.</p>
60	<p><b>MODULE 2: SECURE DELIVERY</b></p> <p>In this class, we will learn about HTTPS and TLS Certificates, and how Akamai works with sites that are HTTPS enabled. You will also learn about the various types of certificates and how they can be obtained and deployed on the Akamai network.</p> <p>In this time, you will do a tech set-up of your systems and get started on configuring an HTTPS site on Akamai.</p>
45	<p><b>MODULE 3: COMPRESSION AND CODECS</b></p> <p>This module defines what codec is, tells the difference between constant and variable bit rates, describe the different types of compression and explain what formats/containers are. It also provides an overview of H.264/AVC vs H.265/HEVC.</p>
45	<p><b>MODULE 4: MEDIA DELIVERY WORKFLOW</b></p> <p>This module provides an overview of the components of the on-demand and live media workflow.</p>
75	<p><b>MODULE 5: ADAPTIVE MEDIA DELIVERY (AMD)</b></p> <p>This module provides an in-depth look how Adaptive Media Delivery enables streaming for HTTP-based formats, how to provision an AMD configuration using Property Manager, test video playback using Akamai support players and refresh Akamai server caches of stale On Demand content.</p> <p><b>LAB: PROVISIONING ADAPTIVE MEDIA DELIVERY</b></p>
75	<p><b>MODULE 6: MEDIA SERVICES LIVE (MSL)</b></p> <p>This module explains the differences between the two current versions of MSL, teaches how to provision MSL 4 via Akamai Control Center and test video playback for various formats.</p> <p><b>LAB: PROVISIONING A LIVE STREAM</b></p>





