Anywall Linux - Technical onboarding

09/10/2024

Diego Contaldi

Crisma Security S.R.L.

**Repository:** https://github.com/gcongedi/AnywallPython

Richiedere accesso a g.congedi@crismasecurity.it

Overview

Anywall è un'applicazione per Oracle Linux 9, e in futuro multipiattaforma, per la visualizzazione di multipli flussi video rtsp su schermo, fino ad un massimo di 16 in contemporanea (per il momento), con possibilità di personalizzare layout, zoom delle singole finestre, visualizzare allarmi, utilizzare browser web.

È in grado di creare overlay direttamente sull’immagine (drawing): etichette, bordi di allarme. In futuro anche filigrana, icone personalizzate.

Le performance di riferimento che vogliamo raggiungere sono quelle del software VLC che prende in ingresso uno stream rtsp FullHD in h.264 e che lo renderizza con un’etichetta in drawing.

Scritta in Python 3.9, si divide in diversi moduli riassumibili in 5 macro-aree:

**Rendering:** Viene utilizzato OpenCV per decodificare i flussi video, recuperare le matrici di pixel (frames) e manipolarle (cambiare dimensioni, disegnare overlay, ecc.), mentre per la creazione di finestre e rendering dei frame viene utilizzato OpenGL tramite la libreria freeGLUT.

Per la visualizzazione delle finestre browser viene utilizzato Selenium.

Sorgenti: dev/opengl.py

dev/browser.py

**Server:** Viene utilizzato Django in coppia con MySQL per consentire una comunicazione con l'utente tramite API. Django fa da backend e scrive sul DB.

Sorgenti: django/anywall\_app/views.py

django/anywall\_app/service.py

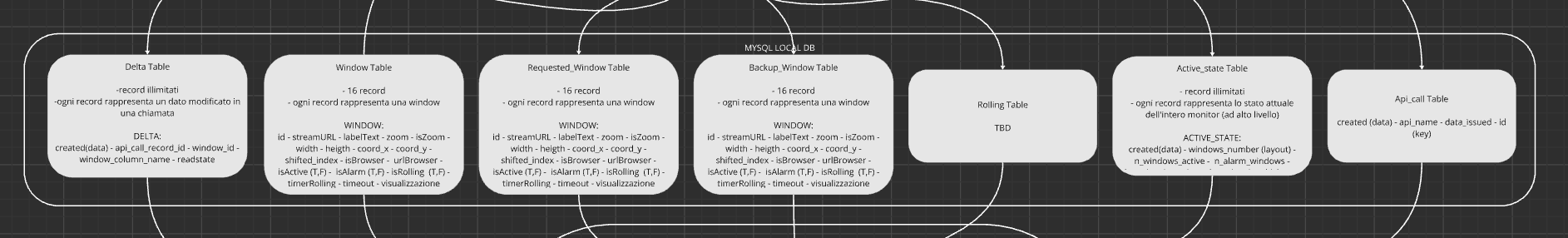
django/anywall\_app/models.py

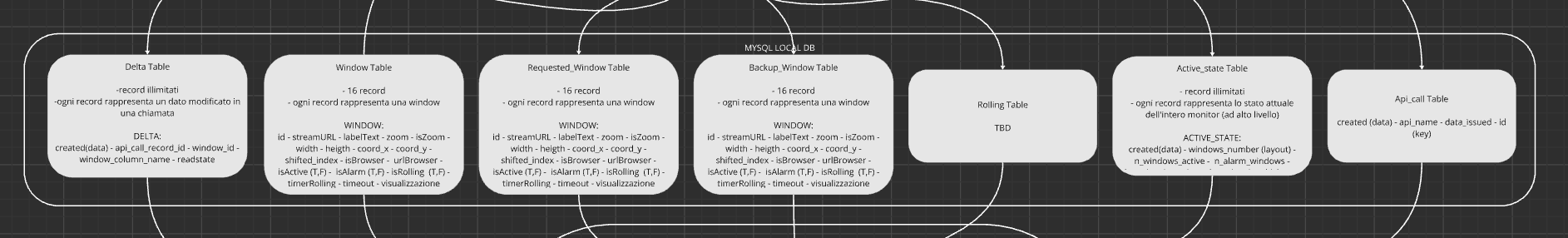
django/anywall\_app/urls.py

django/anywall\_app/serializers.py

**Database**: Il DB funziona come strumento di comunicazione asincrona tra utente e software, nonché come storico delle chiamate ricevute e strumento di recupero di stati precedenti.

TODO: aggiornare con Rolling Table





**Manager:** Gestore degli aggiornamenti della visualizzazione. è l'unico modulo non django che chiama API e che quindi scrive sul DB.

Riceve notifica dei cambiamienti delle tabelle di rappresentazione finestre ed ha il compito di interpretare i dati e comunicarli alle singole finestre.

Sorgenti: dev/manager.py

dev/screen\_helper.py

dev/window\_handler.py

**Monitor:** Gestore dei processi. 16 processi: 1 per ogni finestra OpenGL o Browser, 1 per il server, 1 per il manager, 1 per la struttura dati di IPC (Intercomunicazione tra processi).

Sorgenti: dev/monitor.py

dev/process\_manager.py

Il primo script in ordine di avvio è monitor.py, che avvia e conserva riferimenti a tutti i processi creati e rimane in ascolto per chiamate API di safetyconf (reset) da parte dell’utente o per eventuali errori durante l’esecuzione del software per un riavvio automatico dei processi interessati.

Setup ambiente di sviluppo:

**Guida:** <https://github.com/gcongedi/AnywallPython/blob/main/README.MD>

**Opzionale:** VSCode come IDE

<https://code.visualstudio.com/sha/download?build=stable&os=linux-rpm-x64>

Offre un debugger facile da utilizzare (Run > Start Debugging)

Esecuzione:

Prima dell’esecuzione, assicurarsi di essere all’interno del python environment e di aver fatto il seguente export nelle variabili d’ambiente

export PYTHONPATH=PYTHONPATH:/path/to/repo/rootdir/django/anywall/

Questo permette agli script presenti nella cartella dev/ di utilizzare funzioni di codice da django/anywall

Compilazione:

Per racchiudere il software in un singolo eseguibile utilizziamo PyInstaller.

pyinstaller main.spec

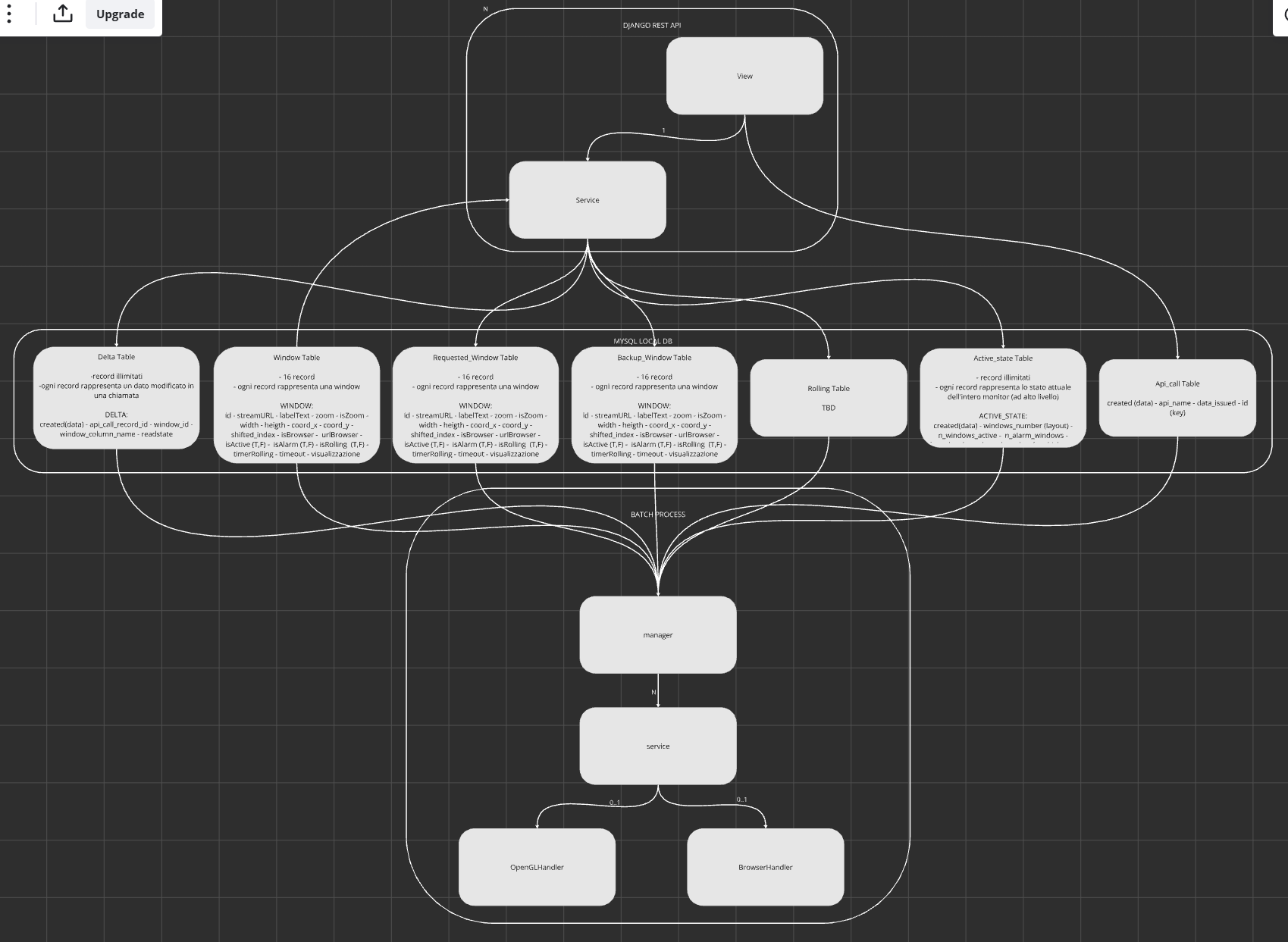
L’eseguibile sarà prodotto al path dist/Anywall

Durante l’esecuzione dell’eseguibile, i path non saranno più relativi ai path della repository ma a una cartella temporanea /tmp/\_MEIxxx/. In caso ce ne fosse bisogno, è possibile aggiungere manualmente file e cartelle qui utilizzando la variabile ‘datas’ nel file main.spec

N.B.: Il termine compilazione è tecnicamente errato. PyInstaller comprime il codice python fornitogli in un unico eseguibile. In futuro potrebbe essere necessario utilizzare un tool più simile a CPython.

Flusso Avvio:

Flusso API:



Decisioni architetturali

* Separazione funzionale del software:

**Implementata:**

* + Server (django/)
  + Manager (dev/)
  + Monitor (dev/)
  + Rendering (dev/)

Ragioni: Consistenza dati e comunicazione, solidità del software, modularità.

* Tecnologie per decoding:

**Implementata:**

* + OpenCV

Ragioni: Semplicità di utilizzo e immediatezza, moderno rispetto ad altre tecnologie.

Seppur questa libreria sia largamente utilizzata in ambito Computer Vision, offre l’elemento VideoCapture() che si rivela un metodo semplice e immediato per leggere uno stream rtsp, e che non impatta particolarmente sulle performance. Al contrario, il suo metodo di rendering (imshow()) ha fini principalmente dimostrativi ed è molto gravoso sull’utilizzo di risorse.

**In sospeso:**

* + GStreamer

Ragioni: Performance ottime, ma di difficile utilizzo. Permette di lavorare con flussi video senza recuperare la matrice di frame in memoria.

I wrapper python testati non si sono rivelati adatti.

Molto difficile capire cosa è andato male durante un’esecuzione e capirne la risoluzione.

Richiede l’installazione di pacchetti linux precisi tra i numerosi offerti dalla community.

L’utilizzo delle pipeline è particolarmente scomodo: per ogni azione che si vuole effettuare (e.g. drawing) c’è bisogno di identificare l’elemento corretto da utilizzare in pipeline e tra quali elementi di questa posizionarlo.

Con una maggiore expertise, si rivelerebbe uno strumento ottimo in grado di migliorare la performance di Anywall.

* + FFMPEG

Ragioni: sullo stesso filone di GStreamer, non una libreria moderna.

Wrapper python non immediati.

Per quanto poco sia stata esplorata come opzione, sembra più semplice e solida rispetto ad errori rispetto a GStreamer.

**PostMortem:**

* + VLC

Ragioni: Essendo il software di riferimento per performance, utilizzare parte delle sue funzioni per il decoding sarebbe ottimo. Purtroppo, non esistono wrapper python che espongono le funzioni necessarie.

In futuro potrebbe rivelarsi la tecnologia più adatta ad Anywall.

* Tecnologie per rendering:

**Implementata:**

* + OpenGL freeGLUT:

Ragioni: Ottime performance per rendering di un flusso rtsp rispetto ad OpenCV.

Per quanto l’utilizzo inizialmente non sia stato immediato, fornisce controllo complete sulle finestre.

È stata una delle prime tecnologie scelte e quella dove è stato investito più tempo.

Ci sono molti elementi gestibili (textures, vertici, vbo, vao, shader, etc.) e non ancora esplorati, ma non tutto potrebbe essere utile, in quanto freeGLUT, come altre librerie di rendering, è in grado di fare molto di più che visualizzare uno stream rtsp in una finestra. Una maggiore expertise potrebbe migliorare leggermente le performance rispetto allo stato attuale.

Contro: Utilizza una funzione di loop propria (GlutMainLoop()) non controllabile, a meno di utilizzare GlutMainLoopEvent(), introdotta in freeGLUT, all’interno di un ciclo, cercando di gestire le sue funzioni di callback allo stesso tempo.

Per quanto offra tutto quello di cui ci sia bisogno per il nostro use case, non è una libreria affatto moderna.

**In sospeso:**

* + Pygame e altre librerie moderne basate su OpenGL o SDL:

Ragioni: tutte le opzioni esplorate, con impostazioni base, risultavano in performance molto peggiori di freeGLUT. Sono tutti, in modo più o meno avanzato, wrapper di librerie C.

Come con freeGLUT, maggiore expertise = maggiore possibilità di migliorare le performance.

SDL era la libreria utilizzata nella precedente interazione di Anywall, scritta in C++, per cui utilizzare un wrapper di questa potrebbe essere una buona strada.

* Configurazione:

Vogliamo che l’utente possa configurare quanto più possibile senza aver accesso ai sorgenti.

**Implementata:**

* + File di configurazione config.env in /usr/local/share/Anywall/conf/

Concept utilizzato al momento solo per la password del Database MySQL. (Dovrà essere configurabile e salvata non in chiaro in futuro.)

Dal codice è possibile caricare il contenuto di questo file nelle variabili d’ambiente utilizzando Dotenv per poi ricavarne i valori col modulo os.

**In sospeso:**

* + Finestra grafica di configurazione. Oltre a grandezza etichetta, font, colori di vari elementi, è ancora da decidere cosa sarà configurabile o meno, e cosa preferiamo auto-detectare (e.g. la risoluzione dello schermo).

Potrebbe essere utile nel gestire casi con più di uno schermo collegato o risoluzioni diverse.

* + Upload di icone, font, e filigrane custom.
  + Abilitazione supporto GPU
    - Scrivere sezione a parte per questa
* ???:
  + Kafka

Misc

VLC performance test setup

* Installare VLC Media Player
* Tools > Preferences
  + Show settings su All
    - Video > Subtitles / OSD
      * Overlays > Subpictures source module
        + Abilitare “Marquee display”
      * > Marquee
        + Impostare testo, posizione e font
* Media > Open Network Stream
  + Inserire URL rtsp
* Verificare utilizzo risorse

N. B. Utilizzando top, disabilitare la Irix mode (Shift + I) in modo da visualizzare le percentuali di utilizzo di un processo rospetto della capacità massima della totalità dei core