PYTHON NO CALLCENTER

GUILHERME BENCKE

PORTO ALEGRE - AGOSTO, 2018

github.com/gbencke/PythonCallCenter

PROUDLY MADE WITH 

OBJETIVO DA APRESENTAÇÃO

Muito tem se falado nas várias funcionalidades do Python como linguagem de programação e de seus potenciais, mas, sinto a necessidade de demonstrar a versatilidade inerente do Python em termos de resolução de problemas reais numa empresa real de Callcenter

Ou seja, nessa apresentação eu irei falar sobre como aplicamos as técnicas do python em problemas reais e não nas técnicas em si

QUEM SOU EU?

Meu nome é Guilherme Bencke, e sou formado em Ciência da Computação pela ULBRA em 2003, programo desde 1986 e profissionalmente desde 1998.

Já trabalhei com dezenas de linguagens e tecnologias em projetos tanto para o Brasil, quanto EUA e França.

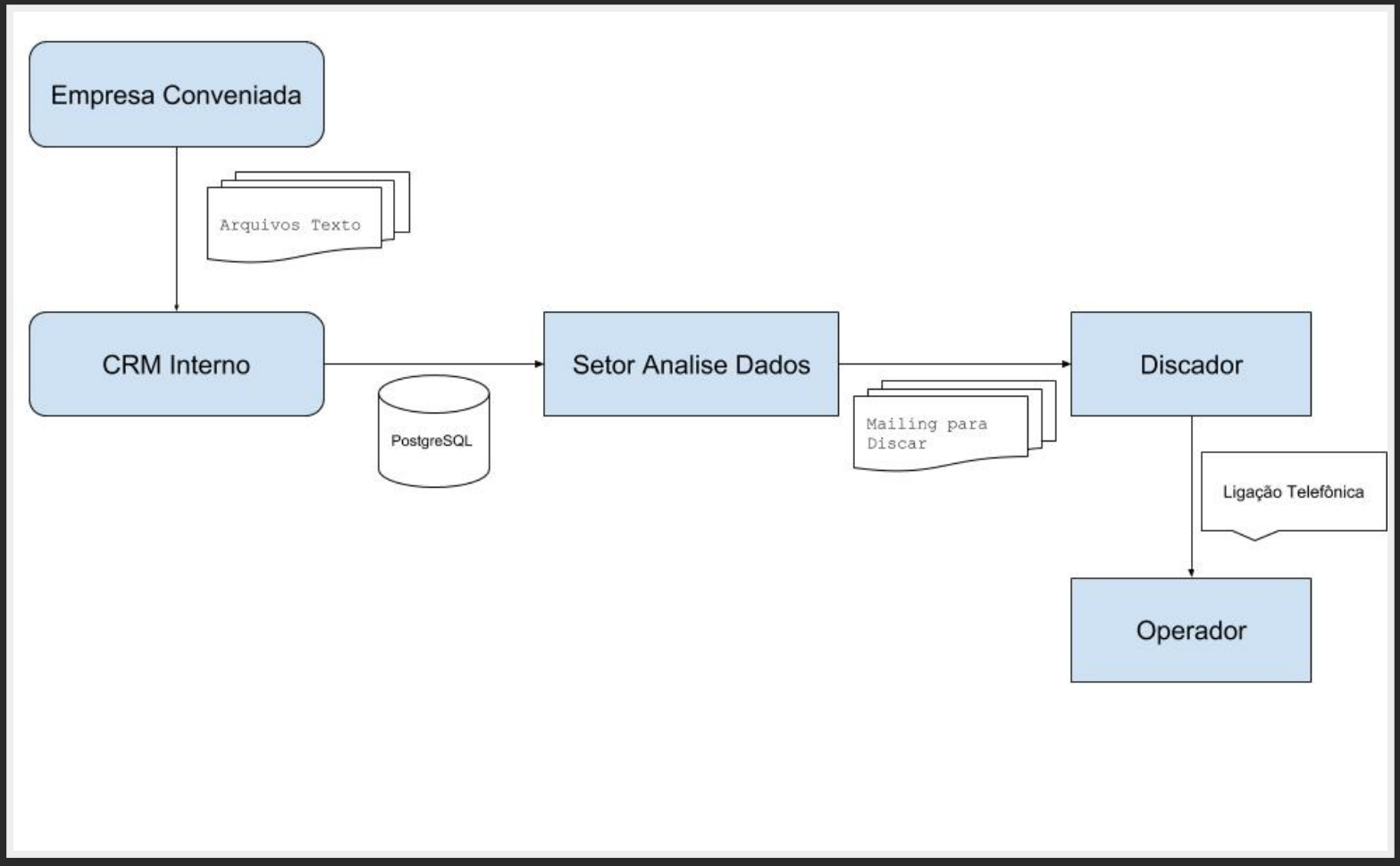
Atualmente (Dependendo do Dia) sou Desenvolvedor / Líder Técnico e Gerente de Desenvolvimento de Software.

COMO FUNCIONA UM CALLCENTER?

Um CallCenter, ou Contact Center, é uma empresa especializada em prestar o serviço de receber um conjunto de dados sobre pessoas a serem contactadas e realizar uma série de ações de contato com essas pessoas.

A Zanc é uma empresa de CallCenter de Cobrança em que o objetivo do contato é o pagamento de uma dívida que um cliente tem com alguma empresa conveniada a Zanc para a cobrança dessa dívida.

FLUXO DE DADOS



DESAFIOS NOS FLUXOS DE DADOS

Basicamente um CallCenter enriquece, filtra, segmenta, gera listas de contato e executa esses contatos. Atualmente existem diversos desafios no dia a dia da empresa, principalmente relacionado com:

DESAFIOS NOS FLUXOS DE DADOS (2)

PORQUE PYTHON?

Podemos ver que a maior parte dos fluxos de dados dentro da empresa tem a necessidade de pequenos ajustes de programação diários e que devem ser feitos no momento em que o problema é detectado.

A Empresa sempre usou bastante .NET e PHP nos seus processos internos, mas, existe a necessidade de treinar pessoas sem formação em programação para que programassem pequenos programas para pequenos tratamentos de arquivos e de ajustes em dados ou análises.

PORQUE PYTHON? (2)

Nesse quesito de ser uma linguagem de fácil aprendizado, versatil tanto na parte de análise de dados, como tratamento deles, além de possibilidade de servir web e rodar em qualquer plataforma, fez com que selecionassemos o python como "lingua franca" da empresa.

Atualmente todos os futuros desenvolvimentos de sofware da empresa são feitos em python.

PYTHON NA RECEPCAO DE DADOS

A Vasta maioria dos dados é recepcionado por arquivos de texto, esses arquivos de texto são recepcionados por FTP em diversos formatos como CSV, delimitado, campo fixo entre outros. Esses arquivos tem em geral centenas de campos que podem mudar de uma hora para outra. Também existe muitas vezes a necessidade de se cruzar dados com os que estão no banco de dados do CRM pois algumas empresas conveniadas mandam dados duplicados ou em ordem errada, oque é necessário fazer uma validação interna antes da carga.

Antes de 2017 diversas pessoas importavam esses arquivos em excel, e gastavam dias para processa-los e valida-los. Decidimos criar um treinamento interno de python para treinar pessoas, que não conheciam programação para que possam abrir os arquivos de texto e realizar as validações necessárias, inclusive acessando os dados no banco de dados do CRM. E atualmente mesmo quando a necessidade de ajustes no tratamento dos arquivos, essa operação leva algumas horas em vez de alguns dias.

Esse treinamento está disponível no github:

https://github.com/gbencke/PythonFlatFilesProcessingC

Importante Salientar que as pessoas que realizaram esse curso jamais tinham programado antes.

PYTHON NA ANÁLISE DOS DADOS

Uma vez carregado os dados para dentro do CRM, é necessário que se faça a análise de quais clientes serão acionados ou não, oque gera o arquivo de mailing que é carregado no discador que efetua as ligações, uma vez o cliente atendendo, essa ligação é passada para um atendente que efetua o contato.

Essa é uma área rica para machine learning, pois é necessário encontrar os padrões que melhor definem os clientes potenciais e gerar um score.

Para isso, o Python tem nos ajudado no sentido de usar a biblioteca XGBoost que cria arvores de decisão que geram um score de diversas variáveis com a probrabilidade do cliente atender o telefone, a probabilidade dele honrar os pagamentos, entre outros. Antes usavamos R para isso, mas, nosso conhecimento com R era limitado a poucas pessoas.

Temos 2 projetos piloto utilizando python para machine learning nessa area:

- **Herval Deep Mailing -** Para cálculo de propensão de pagamentos da carteira Herval (Usando XGBoost) https://github.com/gbencke/HervalDeepMailing

- **Deep Mailing -** Para cálculo de probabilidade de atendimento de ligação da carteira ITAUCARD (Usando XGBoost) https://github.com/gbencke/DeepMailing

PYTHON NA LOCALIZACAO

Como falamos anteriormente, um cliente pode estar em diversos convenios ao mesmo tempo, então é fundamental que as ocorrencias entre os CRMs sejam compartilhados. Isso é extremamente crítico no caso da localização de telefones onde um cliente pode ter 5 telefones num CRM que atende uma empresa, mas, o mesmo cliente pode estar cadastrado em outro CRM de outro Convênio com outros telefones. E existe uma oportunidade de economia se não discamos os telefones de um cliente num CRM, se sabemos que esse cliente já foi discado por outro CRM em outro convênio

Para resolver isso temos uma base de localização com todas as ligações efetuadas por todos os CRMs e para todos os telefones de todos os clientes. Atualmente essa base tem em torno de 500 milhoes de ligacoes com milhoes de telefones cadastrados.

Essa aplicação está sendo rescrita usando banco de dados postgresql, mas, com um web server tornado usando o async do python 3.6, uma vez que todas as ligações de toda a empresa são cadastradas nessa base é possível ter milhares de conexões e atualizações simultâneas, oque o async nos pareceu a melhor solução.

DISCADOR

Mas, o principal impacto que a utilização do python na nossa empresa trouxe foi no discador, que é o software que realiza a ligação ao cliente final, que identifica se ocorreu o alô dele, e passa para um operador para realizar a ligação.

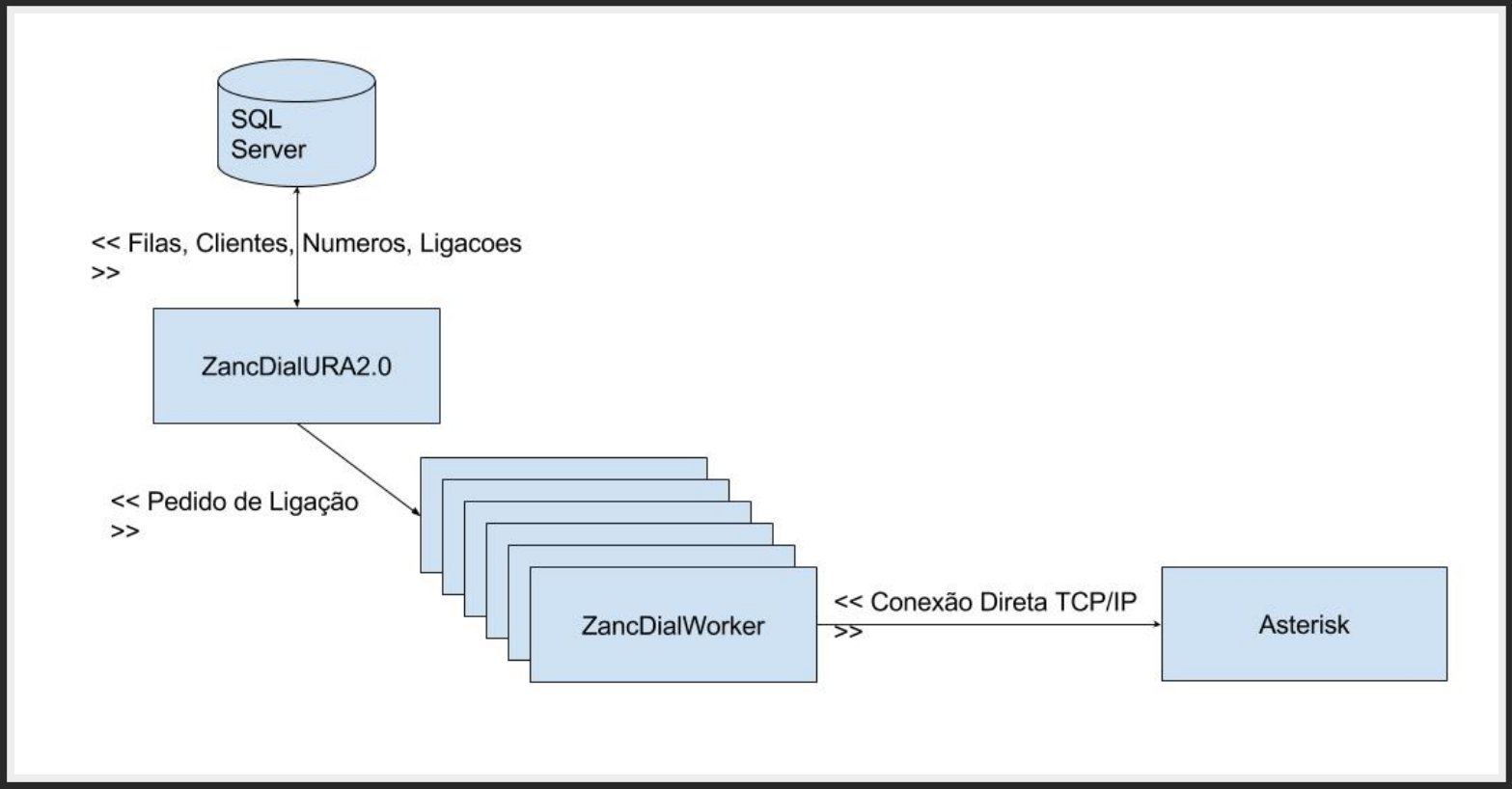
Nossa solução atual foi feita em 2005, em .NET, mas, infelizmente com o tempo se mostrou inadequada e estamos trocando a arquitetura da discagem em si de .NET para python, oque veremos nos próximos slides.

OQUE É O MULTCALL

O Multcall é o sistema discador desenvolvido pela própria zanc pelos ultimos 10 anos. Ele é escrito em .NET e utiliza um banco de dados SQL Server tanto para os dados internos da aplicação quanto os dados do négócio (Clientes, Ligações, Contatos, etc...)

Atualmente a Zanc tem testado diversos outras plataformas de discagem, mas, o retorno ao negócio não tem sido significativo o suficiente para justificar o custo adicional de licenças de um software terceiro.

ARQUITETURA ATUAL DO MULTCALL 1.0



PROBLEMAS COM O MULTCALL 1.0

Com o passar do tempo, as limitações do Multcall 1.0 começaram a aparecer. Em termos concretos, os problemas do Multcall Atualmente são:

- **Escalabilidade**: Capacidade de fácilmente aumentar o número de operadores logados.
- **Versao Asterisk**: Atrelamento fixo do Multcall a uma específica versão do Multcall
- **Balanceamento de Carga**: Dificuldade em distribuir a carga de ligações entre diversas centrais.

ESCALABILIDADE (1)

Um dos grandes problemas com o Multcall 1.0 é que o mesmo banco de dados serve tanto para os dados do negocio quanto para os dados internos da aplicação (Direitos, Operadores Disponiveis, Filas de Discagem, etc...), dessa forma, o quanto mais operadores você colocar em uma operação, mais o sistema irá ter a sua performance degradada.

ESCALABILIDADE (2)

Um dos exemplos mais claros é a questão das leituras de parâmetros, pois por ser um sistema em tempo real em que a reconfiguração dos parametros do sistema tem que ter impacto direto na operação, essa leitura de parametros (que não pertencem ao negocio em si) pode causar lentidão em operações de sistema como a carga e/ou manipulação de um mailing.

VERSÃO ASTERISK

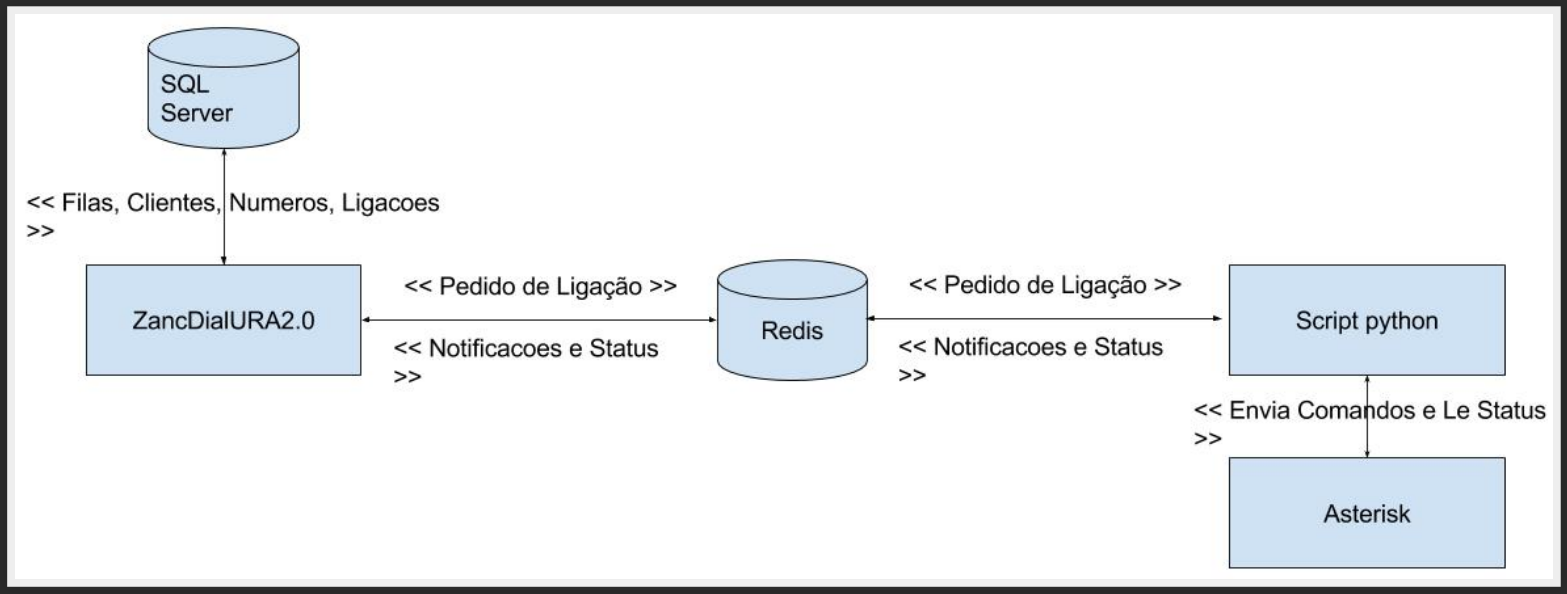
A Versão atual do MultCall conecta o codigo .NET diretamente a Central Telefonica, oque causa 2 problemas:

- Não existe suporte oficial do Asterisk ao .NET, dessa forma, deve haver uma "Adaptação" de bibliotecas terceiras para que funcione corretamente
- Essa adaptação tem que ser testada em todas as versões do Asterisk, pois não é garantido que funcionará em todas as situações.

BALANCEAMENTO DE CARGA

Atualmente para balanceamento de carga, o Multcall se conecta com o AsteriskProxy que é um programa provido pelo próprio asterisk para balanceamento e distribuição de carga. Esse software não é mais suportado pelo Asterisk, oque causa uma série de problemas de suporte, além desse programa ser relativamente instável oque causa paradas na operação.

ARQUITETURA MULTCALL 2.0



MULTCALL 2.0

Como podemos ver na figura anterior, para resolver os 3 problemas atuais do Multcall, criamos uma camada intermediaria entre o multcall e a central aonde colocar um banco de dados de alta performance chamada Redis.

Dessa forma, o Multcall escreve no banco de dados Redis sem contato com a central telefonica em si, e na central telefonica instalamos um script em python que realiza a discagem propriamente dita.

REDIS (1)

Redis é um banco de dados do tipo "Chave-Valor", ou seja, Não-SQL, de alto desempenho e que roda em memória. Objetivo dele é armazenar e disponibilizar rapidamente os dados internos da aplicação, ou seja, aqueles dados que não interessam ao Negócio, ao NEC, e que jamais serão alvos de Relatorios.

REDIS (2)

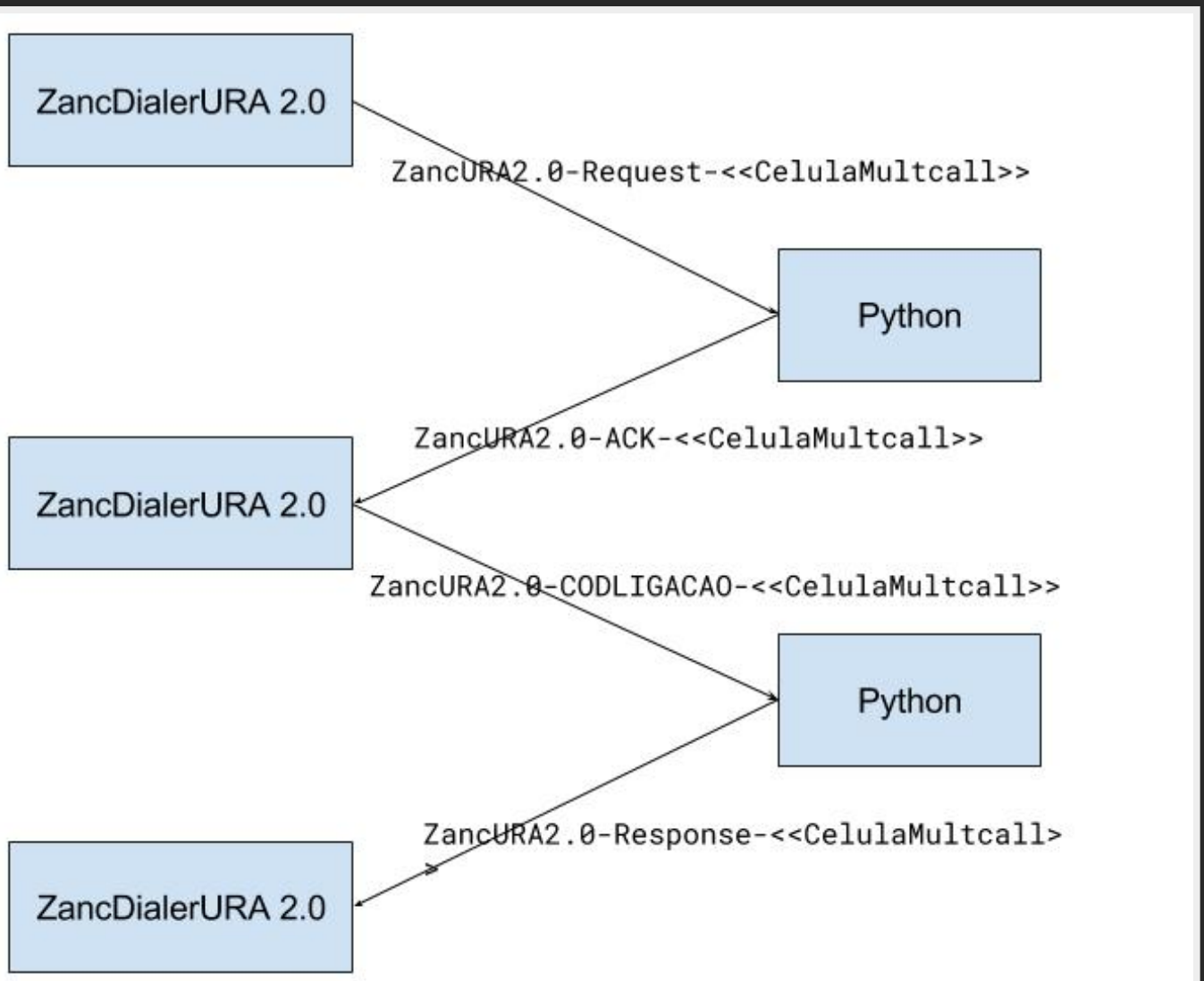
Por ele não permitir consultas em "Massa" como no SQL, e sim um valor apenas por vez, ele é muito escalavel e dificilmente tem problemas de sobrecarga, ao contrario do SQL Server, em que operações de Gerenciamento podem ter impacto na performance do sistema como um todo.

PYTHON

Atualmente, a interface padrão com o Asterisk é feita através da linguagem de programação Python, ela é suportada em todas as versões do Asterisk tanto presentes, passadas quanto futuras.

O fluxo de controle das ligações fica no python que usa uma série de filas do Redis para comunicação com o Discador. Usando o Redis como meio de comunicação entre o discador e a central, e usando o python dentro da Central, podemos garantir que o Mulcall 2.0 irá funcionar em qualquer versão presente, passado e futura do Asterisk

FLUXO DE FILAS DO REDIS



<< Aonde CelulaMultcall é o nome da célula no Multcall >>

GESTÃO DO MULTCALL (1)

Em termos de Gestão do Multcall, nada muda, tudo continua nas mesmas telas e o detalhe de se usar os DialWorkers ou o Redis para comunicação com a Central é apenas uma questão interna de configuração. Existem 5 parametros que configuram o Multcall para usar o REDIS:

GESTÃO DO MULTCALL (2)

- **Redis Ativar**: Ativa a comunicação com a Central via Redis
- **Redis URL**: URL que aquele Convenio ou Celula irá se conectar ao Redis
- **Redis Workers Ligaca**:Determina o numero de threads que será usada para gerar ligações ao Redis
- **Redis Workers ACK**: Determina o numero de threads que receberá o ACK da central e gerará o CodLigacao
- **Redis Workers Response**: Determina o numero de threads que irá processar o resultado de ligação do Redis

GESTÃO DO LADO DA CENTRAL (1)

Do lado da central é apenas necessário que se instale e execute o script em python disponivel no seguinte repositorio do git:

http://srv01stm136.dominio.zanc.com.br/ Telecom/ UraReconhecimentoVoz

GESTÃO DO LADO DA CENTRAL (2)

Para a instalação é apenas necessário:

- Executar o git clone desse repositorio
- Instalar o pacote redis para python através do comando:

```
pip install redis
```

- Configurar as filas em **src/settings.py**

EXECUÇÃO DO PYTHON NA CENTRAL

Uma vez instalado o redis, podemos ir para o diretorio criado pelo git clone e executar

```
.\run.sh $NomeCelula
```

Aonde NomeCelula é o numero da celula configurado no arquivo settings.py do diretorio src/settings.py

FORMATO ARQUIVO SETTINGS.PY

```
SETTINGS = {
    "LOG_LEVEL": "DEBUG",
    "LOG_FORMAT": "%(thread)d - %(asctime)-15s %(message)s",
    "LOG_LOCATION": "../logs",
    "redis_host": "localhost",
    "redis_port": "6379",
    "Timeout_Command_Response": 20,
    "Timeout_Command_ACK": 10,
    "Retries": 2,
    "queues": [
        {"nome": "ASR_TIM", "threads": "100"},
        {"nome": "ura-novoteste2", "threads": "5"},
        {"nome": "ura-novoteste3", "threads": "5"}]
}
```

ARQUIVO SRC/SETTINGS.PY (1)

- **LOG_LEVEL**: O Nivel de log a ser gerado pelo script python, nesse momento o default é DEBUG
- **LOG_FORMAT**:O Formato a ser gerado em cada linha de log do script
- **LOG_LOCATION**: A localização dos logs dentro da central
- **redis_host**: O host aonde está configurado o redis para essa central
- **redis_port**: A porta aonde o redis esta ouvindo

ARQUIVO SRC/SETTINGS.PY (2)

- **Timeout_Command_Response** Tempo em segundos pela qual devemos esperar pelo ack da resposta da central ao nosso comando
- **Timeout_Command_ACK** Tempo em segundos pela qual devemos esperar pelo ACK do multcall ao nosso ACK
- **Retries**: Numero de tentativas que deve ser feitas de envio de comando a central.

QUEUES EM SRC/SETTINGS.PY

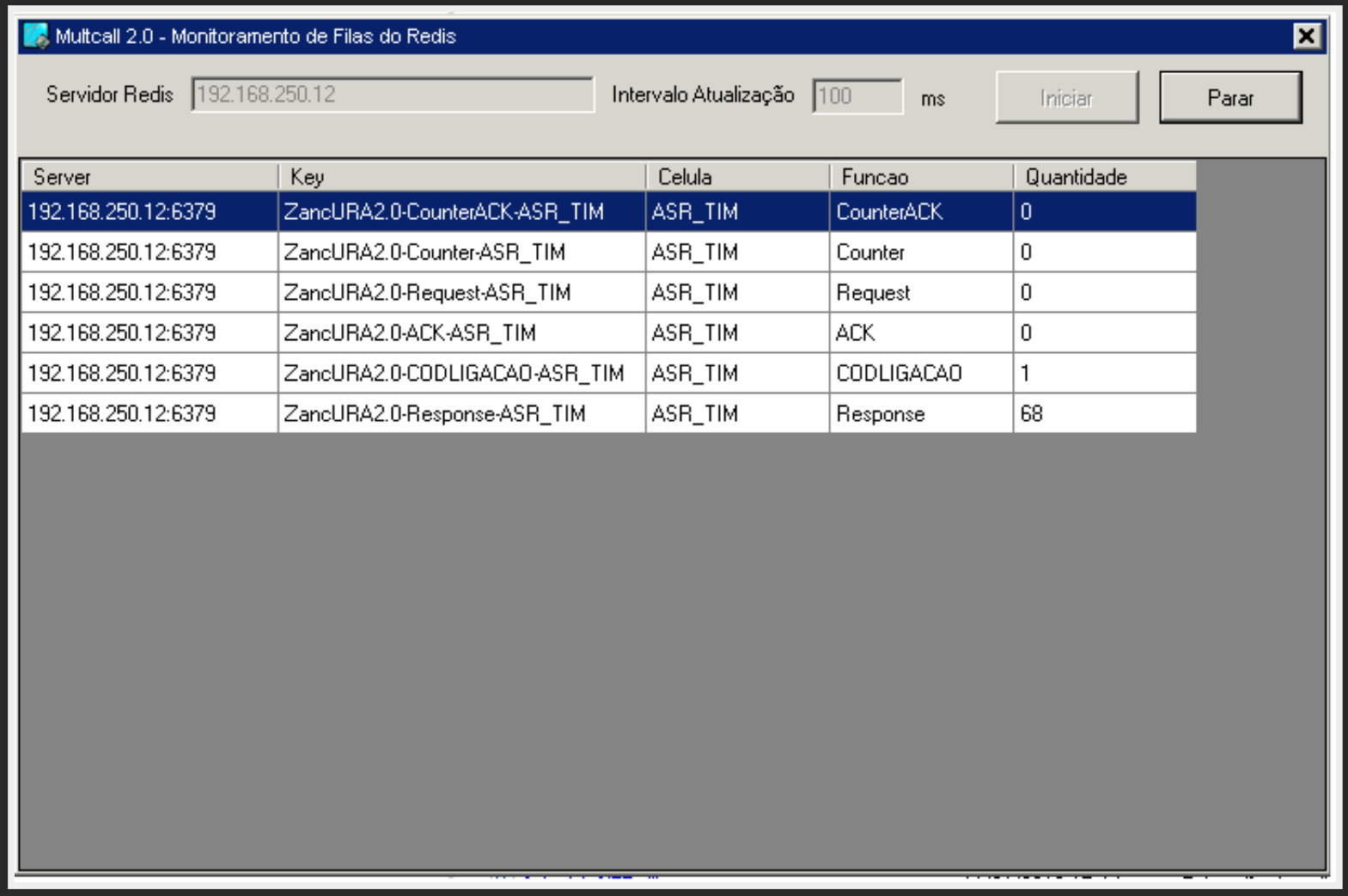
O Campo Queues do arquivo de settings é um array com todas as células que esse script pode executar. Ele contém apenas 2 campos:

- **nome**: O nome da célula no Multcall, que deve ser igual a usada no multcall, senão não funcionará
- **threads**: O numero de threads, ou seja, o numero máximo de ligações que poderá ser executado por esses script.

MONITORAMENTO DAS FILAS DO REDIS (1)

Para monitoramento das filas do Redis, e detecção de qualquer gargalo ou sobrecarga, foi desenvolvido um simples programa de monitoramento. Ele é bastante simples, apenas informar o servidor e o intervalo de atualização desejado e clicar em iniciar e o programa irá mostrar em tempo real todas as informações armazenadas no Redis como em demonstrado embaixo:

MONITORAMENTO DAS FILAS DO REDIS (2)



CONCLUSÃO E TRABALHOS FUTUROS

Criando uma camada intermediária entre a Central e o Multcall, conseguimos resolver dessa forma, uma série de problemas que comprometiam o seu uso na operação no dia-a-dia. Os proximos passos sao:

- Mover todos os dados internos da aplicação para fora do SQL Server e para o Redis
- Implantar a metodologia apresentada na discagem Preditiva