****

**FACULDADE 7 DE SETEMBRO - FA7**

**CURSO GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO**

**Jorge Luís Lima da Silva**

**SISTEMA DE GESTÃO PÚBLICA PARA O CADASTRAMENTO DE OCORRÊNCIAS UTILIZANDO ARQUITETURA DE MICROSERVIÇOS**

**FORTALEZA – 2015**

**Jorge Luís Lima da Silva**

**SISTEMA DE GESTÃO PÚBLICA PARA O CADASTRAMENTO DE OCORRÊNCIAS UTILIZANDO ARQUITETURA DE MICROSERVIÇOS**

Artigo Cientifico apresentada à Faculdade 7 de Setembro como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Robério Gomes, Mestre.

**FORTALEZA – 2015**

**SISTEMA DE GESTÃO PÚBLICA PARA O CADASTRAMENTO DE OCORRÊNCIAS UTILIZANDO ARQUITETURA DE MICROSERVIÇOS**

Artigo Cientifico apresentada à Faculdade 7 de Setembro como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Robério Gomes, Mestre.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Jorge Luís Lima da Silva

Artigo aprovado em: \_\_\_\_/\_\_\_\_\_/\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Robério Gomes, Mestre. (FA7)

1º examinador: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2º examinador: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof. Marun Simão Filho, MSc. (FA7)

Coordenador do curso

**RESUMO**

Este trabalho propõe o desenvolvimento de uma aplicação como serviço sobre a arquitetura de microserviços, para o gerenciamento de ocorrências voltados para órgãos de segurança pública e defesa civil. Neste artigo serão abordados temas sobre uso de informação em gestão de segurança pública, aplicativos já utilizados pelo governo federal para integração de dados e o guia contendo os padrões de interoperabilidade utilizadas pelo mesmo. Serão abordados os conceitos de microserviços, mudança para microserviços, microserviço como componente e expostas as tecnologias existentes que possibilitam a implementação de um microserviço. A aplicação proposta tem intuito de prover um sistema escalável, independente e de alto desempenho, utilizando uma interface pública com o objetivo de garantir a integração e interoperabilidade entre diversos tipos de sistemas e plataformas, possibilitando o envio e recebimento de dados através de requisições *web*.

Palavras-chave: Microserviços, REST, HTTP, Integração.

*ABSTRACT*

*This research proposes the development of an application as a service on the microserviços architecture for the incident management focused on law enforcement agencies and civil defense. This article will discuss topics on the use of information on public security management, applications already used by the federal government for data integration and the guide containing the interoperability standards used by the same. Microserviços the concepts are addressed, changing to microserviços, as microserviço existing technologies that enable component and exposed to implement a microserviço. The application proposal is intended to provide a scalable, independent and high performance using a public interface in order to ensure the integration and interoperability between various types of systems and platforms, allowing customers to send and receive data through web requests.*

*Keywords – Microservices, REST, HTTP, Integration.*

1. **INTRODUÇÃO**

Segundo o Ministério da Justiça[1], fatores como expansão urbana, vulnerabilidade social e concentração de pessoas em áreas de risco estão diretamente relacionados com a violência e criminalidade. Diante desses fatores os governos necessitam implantar políticas públicas que visam garantir os direitos constitucionais do cidadão. Além disso é necessário utilizar ferramentas de tecnologia e conhecimentos científicos para o auxílio ao combate desses problemas.

Cardozo[2], Ministro da Justiça naquela época, relata que o sistema de informações sobre segurança pública do Brasil é “um desastre”. Cardozo ressaltou que não há combate à criminalidade sem informação. “É preciso saber como e onde os crimes ocorrem para poder estabelecer políticas”, afirmou o ministro[2]. De acordo com [2], a integração do sistema de informação sobre segurança pública ajudará a gerenciar melhor a alocação de recursos na área. “Voluntarismo e boa vontade são importantes, mas resolvem pouco. Sem planejamento e gestão vamos desperdiçar dinheiro” [2].

Ainda em [2], um dos principais problemas da segurança é falta de informação, é preciso unificar essas informações em um sistema, que possa se comunicar com vários outros de forma independe e transparente. O surgimento de novas tecnologias e arquiteturas possibilitam um novo olhar para solução efetiva desse problema. Uma possível solução é exemplificada em[3] onde um estudo realizado pela *Rand Corporation*[4], em 1998, avaliou os resultados de cinco ações diferentes realizadas nos Estados Unidos para retirar crianças do mundo dos crimes, em termos dos custos e benefícios. Este relatório fundamentou uma mudança significativa na ação do governo dos Estados Unidos em termos da gestão de políticas de segurança pública. O atraso em desenvolvimento de sistemas que gerencie indicadores no contexto de segurança pública, resulta no aumento das taxas de criminalidade, pessoas em áreas de riscos e a impossibilidade da avaliação de impactos e custos de ações e projetos dos setores públicos neste contexto.

Em meio a muitas arquiteturas existentes, no deparamos com um novo de conceito, o conceito de Microserviços. A abordagem de microserviço utiliza o conceito de desenvolvimento de aplicações através de um conjunto de pequenos serviços independentes, em que cada serviço utiliza a sua própria estrutura e são executados em processos separados. Os serviços podem se comunicar através de mecanismos leves que utilizam o protocolo HTTP (*Hypertext Transfer Protocol*). Essas características auxiliam na construção de aplicativos que disponibilizam recursos que podem ser consumidos por diferentes plataformas.

Este trabalho irá conceituar a abordagem de microserviços demonstrando as suas características, área de atuação, tecnologias, ferramentas, vantagens e desvantagens. Como prova de conceito, este artigo propõe o desenvolvimento de um serviço como uma interface de programação sendo executado em instancias e processos diferentes. Este serviço será um componente reutilizável e escalável utilizando os conceitos abordados, com o objetivo de coletar e disponibilizar dados no contexto de segurança pública e defesa civil. A aplicação irá coletar e armazenar várias informações como: local, jurisdição e áreas de riscos, disponibilizando os dados através de *web services*. Esses dados auxiliarão o órgão públicos na tomada de decisão.

O restante deste trabalho estar organizado da seguinte forma: Seção 2 apresenta uma visão geral sobre sistemas governamentais, padrão de interoperabilidade, conceitos gerias sobre microserviços, trabalhos relacionados, tecnologias e ferramentas; a seção 3 descreve a visão geral de arquitetura utilizada e suas implementações; e a seção 4 conclusões e trabalhos relacionados.

1. **REFERENCIAL TEORICO**

**2.1 USOS DA INFORMAÇÃO EM GESTÃO DE SEGURANÇA PÚBLICA ATRAVES DE SISTEMAS INTEGRADOS.**

A maioria dos países, principalmente na Europa, desenvolve atualmente sistemas de informação utilizando registro de ocorrências policiais e de atividade judicial bem como gera dados a partir de questionários[3]. Em torno de 1825 o Estado francês foi pioneiro na coleta regular de dados das cortes criminais como instrumento de auxílio à administração pública quinze anos após a deflagração da revolução francesa, a partir daí vários países passaram a desenvolver formas de monitoramento de dados. O Brasil encontra-se com algumas décadas de atraso[15] sendo o primeiro estudo de coleta sobre informação sendo realizado em 1870.

Deste o ano de 2003, a Secretaria Nacional de Segurança Pública (SENASP) vem realizando uma série de ações, buscando valorizar a informação como principal ferramenta de ação das organizações de segurança pública[3]. Uma grande ação foi a criação do Sistema Nacional de Estatísticas de Segurança Pública e Justiça Criminal – SINESPJC (atual módulo de estatística do SINESP[5]), implantado em 2004, com o módulo Polícia Civil e posteriormente o módulo Policia Militar em 2006. Esse sistema teve como principal objetivo iniciar o processo informatizado de coleta de dados estatísticos junto aos 27 Estados da Federação[3]. As estatísticas criminais são utilizadas para retratar a situação da segurança pública e permitir o planejamento de ações policiais e de investimentos no setor. A compilação dos dados é realizada pela Secretaria Nacional de Segurança Pública - SENASP por intermédio do módulo de estatísticas do Sistema Nacional de Informações de Segurança Pública, Prisionais e sobre Drogas – SINESP. Até 2012, as estatísticas criminais eram geradas somente com dados dos municípios com população superior a 100 mil habitantes, a partir de janeiro de 2013, já é possível o tratamento e análise de dados de todos os municípios brasileiros, permitindo, assim, retratar o Brasil de forma mais detalhada. Isto e muitas outras conquistas só foram possíveis através da assinatura de um termo de adesão ao SINESP realizada pela União com os Estados e o Distrito Federal, tornando efetivo o apoio dos participantes no que tange o cumprimento da lei 12.681 de 04 julho de 2012, que instituiu o SINESP. O SISNESP foi regulamentado pela Lei 12.681.

Como relatado anteriormente o governo Federal vem tomando atitudes para a construção de sistemas para integração de informações que ajudem na gestão de segurança pública. Além disso também foi criado o Sistema Integrado de Informações sobre Desastres - S2ID que visa informatizar o processo de transferência de recursos em virtude de Desastres. O objetivo é qualificar e dar transparência à gestão de riscos e desastres no Brasil, já que, além de agilizar o processo, garantirá o acesso a informações sobre desastres em diversos níveis[6]. Estes são exemplos de sistemas criados pelo governo federal afim de prover informação que ajudem na tomada de decisões.

**2.2 LEI 12.681**

Buscando a garantir a excelência e a legalidade, o presente trabalho utilizou a LEI 12.681[6], que regulamenta as funcionalidades do SISNESP como base para definição dos requisitos do serviço que será utilizado como estudo de caso. Um dos principias artigos utilizados da lei foram:

* Art. 1o É instituído o Sistema Nacional de Informações de Segurança Pública, Prisionais e sobre Drogas - SINESP, com a finalidade de armazenar, tratar e integrar dados e informações para auxiliar na formulação, implementação, execução, acompanhamento e avaliação das políticas relacionadas com [6].
* Art. 2º O Sinesp tem por objetivos[6]:

I - proceder à coleta, análise, atualização, sistematização, integração e interpretação de dados e informações relativos às políticas de que trata o art. 1º[18];

IV - garantir a interoperabilidade dos sistemas de dados e informações, conforme os padrões definidos pelo Conselho Gestor[6].

Esses dois artigos da lei refletem os principais objetivos da aplicação.

* 1. **ePING**

O ePING[7] pode ser entendido como um guia que define padrões a serem utilizados por órgão federais para integração de dados em vários níveis.

A arquitetura ePING – Padrões de Interoperabilidade de Governo Eletrônico – define um conjunto mínimo de premissas, políticas e especificações técnicas que regulamentam a utilização da Tecnologia de Informação e Comunicação (TIC) no governo federal, estabelecendo as condições de interação com os demais Poderes e esferas de governo e com a sociedade em geral[7].

A interoperabilidade pode ser entendida como uma característica que se refere à capacidade de diversos sistemas e organizações trabalharem em conjunto (interoperar) de modo a garantir que pessoas, organizações e sistemas computacionais interajam para trocar informações de maneira eficaz e eficiente[20].

A ePING é concebida como uma estrutura básica para a estratégia de governo eletrônico, aplicada ao governo federal – Poder Executivo, não restringindo a participação, por adesão voluntária, de outros Poderes e esferas de governo. Para os órgãos do governo federal, Poder Executivo brasileiro, a adoção dos padrões e políticas contidos na ePING é obrigatória (Portaria SLTI/MP nº 92, de 24 de dezembro de 2014). No âmbito das entidades supramencionadas, são obrigatórias as especificações contidas na ePING para[20]:

* + Todos os novos sistemas de informação que vierem a ser desenvolvidos e implantados no governo federal e que se enquadram no escopo de interação, dentro do governo federal e com a sociedade em geral;
  + Sistemas de informação legados que sejam objeto de implementações que envolvam provimento de serviços de governo eletrônico ou interação entre sistemas;
  + Aquisição ou atualização de equipamentos de TIC.

As áreas cobertas pela EPING estão segmentadas em:

* Interconexão;
* Segurança;
* Meios de Acesso;
* Organização e Intercâmbio de Informações;
* Áreas de Integração para Governo Eletrônico.

Outra importante propriedade abordada nesse padrão é a escalabilidade. Segundo o guia[7] as especificações selecionadas deverão ter a capacidade de atender alterações de demanda no sistema, tais como, mudanças em volumes de dados, quantidade de transações ou quantidade de usuários. Os padrões estabelecidos não poderão ser fator restritivo, devendo ser capazes de fundamentar o desenvolvimento de serviços que atendam desde necessidades mais localizadas, envolvendo pequenos volumes de transações e de usuários, até demandas de abrangência nacional, com tratamento de grande quantidade de informações e envolvimento de um elevado contingente de usuários.

Por fim o guia é utilizado qualificação de especificações técnicas para tecnologias utilizadas em aplicações governamentais. Utilizando os termos: Adotado, Recomendado, Em transição, Em Estudo.

**2.4 MICROSERVIÇOS**

**2.4.1 CONCEITOS**

A arquitetura de microserviços surgiu recentemente com um novo conceito que vem se popularizando em meio as muitas arquiteturas de software[1], causando várias discussões em torno de seus conceitos, benefícios e restrições. Algumas empresas têm se destacado pela evolução de suas arquiteturas para microserviços, como é o caso da Netflix[[1]](#footnote-1), SoundCloud[[2]](#footnote-2) e Twitter[[3]](#footnote-3) [1].

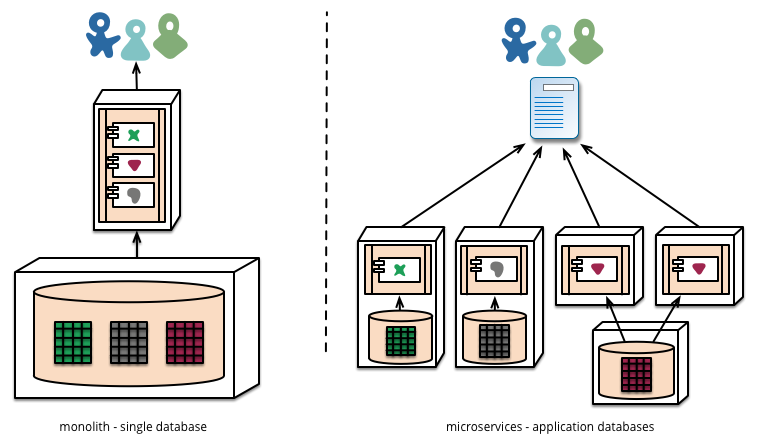
Segundo Newman[8], microserviços são pequenos serviços autônomos que trabalham em conjunto. Outros autores como Lewis e Fowler[9], um dos criadores de termos como POJO, descrevem microserviços como um termo novo que surgiu nos últimos anos, para implantar uma maneira particular de construir aplicações de software, como suítes de serviços independentemente e implementáveis, cada um executando em seu próprio processo. Segundo o autor [10], microserviços é uma abordagem de termo relativamente novo para a criação de componentes independentes e reutilizáveis. Embora não exista uma definição precisa deste estilo arquitetônico, há certas características comuns ao redor dessa abordagem, como a capacidade de negócios, implantação automatizada e o controle descentralizado de linguagem e de dados[9]. Cada serviço é executado em um processo independente, podendo ser implementados como um serviço isolado, uma plataforma para serviço ou como um próprio processo do sistema operacional[8]. A comunicação entre os serviços pode ser através de mecanismos leves, geralmente em uma API com recursos HTTP[8].

Para começar a discutir o estilo microserviços é importante compará-lo com o estilo de estruturas monolíticas, que são construídas como um bloco único[9]. Em sistemas que utilizam arquitetura JEE[[4]](#footnote-4) podemos citar como estrutura monolítica os componentes WAR. Resumidamente esses aplicativos executam em um servidor de aplicação ou em um contêiner, que lidam com requisições HTTP[[5]](#footnote-5), executam lógica do domínio, recuperam e atualizam dados do banco de dados (que consiste em muitas tabelas inseridas em um sistema de gerenciamento de banco de dados comum), e selecionam e preenchem visualizações HTML[[6]](#footnote-6) a serem enviadas para o navegador[9].

De acordo com [11], esse padrão funciona razoavelmente bem para pequenas aplicações, pois o desenvolvimento, testes e implantação de pequenas aplicações monolíticas é relativamente simples. No entanto, para aplicações grandes e complexas, a arquitetura monolítica torna-se um obstáculo ao desenvolvimento e implantação, dificultam a utilização de uma entrega contínua, além de limitar a adoção de novas tecnologias. Essas estruturas executam em único processo e qualquer mudança no sistema envolvem criação e implantação de uma nova versão do aplicativo, isso significa que a cada mudança todo o bloco é suspenso causando indisponibilidade dos outros serviços e dificultando a implantação de novas versões[8]. Para cada linha de código ou biblioteca atualizada todos os times de desenvolvimento precisam estar cientes da modificação, isso diminui a autonomia dos times podendo interferir nas entregas continuas do projeto[8]. A medida que aplicativos monolíticos crescem em estrutura, sua complexidade e dificuldade de gerenciamento aumentam proporcionalmente, causando dificuldade na integração de novos membros nas equipes de desenvolvimento, dependendo da tecnologia utilizada fatores como curva de aprendizado e atualizações de ferramentas de terceiros afetam diretamente o ciclo de desenvolvimento do software[8].

A abordagem de microserviços propõem a decomposição de sistemas monolíticos em suíte de serviços autônomas, com criação, gerenciamento e implantação independes entre si [8]. Com microserviços, podemos fazer uma mudança para um único serviço e implantá-lo de forma independente do resto do sistema, isso permite uma implantação descentralizada e sem a necessidade de reiniciar o sistema central[8]. Serviços individuais são mais fáceis de entender e documentar, estão focados nos domínios da aplicação, isso facilita a adoção de novos membros nas equipes como a adoção de novas tecnologias[8]. A Figura 1 apresenta um sistema monolítico entre os clientes e base de dados, já na Figura 2 é apresentado o mesmo sistema dividido em microserviços.

Figura(1) Figura(2)



Fonte[9]

Um dos grandes benefícios da abordagem de microserviços em relação a estruturas monolíticas é o isolamento de falhas. Nas estruturas monolíticas se uma funcionalidade falha toda a estrutura fica indisponível, na arquitetura de microserviços se um serviço falha, este é isolado rapidamente, enquanto os outros permanecem em execução, garantido a disponibilidade e tornando fácil e rápida a correção de erros[8]. Essa é uma das principais razões pelas quais as organizações como Amazon[[7]](#footnote-7) e Netflix utilizam essas arquiteturas[8]. Isso também garante que cada serviço seja escalável independentemente.

Outro conceito importante dos microserviços são os gerenciamentos dos serviços. A gerência do serviço é completamente separada de outros serviços, permitindo que cada estrutura seja construída utilizando diferentes tecnologias e linguagens de programação[8], usando próprio modelo de dados e garantindo a heterogeneidade dos serviços. Isso possibilita que as equipes de desenvolvimento sejam pequenas, autônomas e mais produtivas, focando nas regras de negócio do projeto e escolhendo pela melhor estrutura pros seus serviços, consequentemente se obtém menos linhas de código escritas e novos profissionais podem ser incluído mais facilmente aos projetos[8]. Os microserviços por serem independentes permitem a adoção de novas tecnologias mais facilmente, diminuindo o impacto das mudanças. Caso ocorra uma falha ao [experimentar uma nova tecnologia](http://www.infoq.com/presentations/evolving-architecture-guardian-uk), pode-se descartar o uso e voltar para versão anterior sem afetar todo o projeto. Essa opção é bastante crítica em estruturas monolíticas, nessas estruturas todas as mudanças impactam em todas as funcionalidades, podendo causar problemas de execução e indisponibilidade.

Como qualquer outra arquitetura, microserviços possuem suas desvantagens como: complexidade de se lidar com vários sistemas distribuídos, complexidade operacional de particionar os serviços, regra de negócio duplicada e teste automatizados, a sua adoção não deve ser levada de forma trivial.

Finalizando a introdução dos conceitos de microserviços podemos analisar dois princípios importantes na engenharia de software, o baixo acoplamento e auto coesão. Baixo acoplamento em microserviços significa que a mudança e implantação do serviço não devem interferir nas outras estruturas e alta coesão significa que os serviços devem ser auto suficientes diminuindo o número de chamadas remotas para outros serviços[8].

**2.4.2 MUNDANÇAS PARA MICRO-SERVIÇOS**

A abordagem de microserviço possui vários desafios em sua implementação, além da complexidade de se administrar vários sistemas distribuídos, os testes são mais difíceis para essas aplicações[10]. Outro desafio, é que essa abordagem também leva ao aumento do consumo de memória, devido ao próprio espaço de endereço para a cada serviço[10].

Um dos maiores desafios é decidir como dividir (particionar) o sistema em microserviços. Segundo Newman[12], obter o domínio errado de um serviço pode ser caro, e que componentes excessivamente acoplados poderiam ser piores do que apenas ter um único sistema monolítico.

Para construção de um modulo para microserviço, existe o desafio[12] de separar o código de forma limpa, criando uma API de granularidade grossa que faça sentido para ser usado em uma rede. Ainda de acordo com[10], encontramos duas estratégias para a divisão de uma grande aplicação em pequenos serviços, a primeira é a divisão por verbos: Por exemplo, serviço que implementa o subsistema de *login*, subsistema de *backup* e etc. A segunda é particionar o sistema por substantivos ou recursos, este tipo de serviço é responsável por todas as operações sobre as entidades escolhidas. O ideal é que cada serviço tenha apenas um pequeno conjunto de responsabilidades.

De acordo com[12] uma das soluções para diminuir os problemas de partição em aplicações é o mapeamento dos seus módulos que podem no futuro tornar-se serviços em seu próprios processos e esquemas de dados. Obviamente essa abordagem tem um custo, mais esse custo vai facilitar implementação de mudanças para microserviços.

**2.4.3 MICROSERVIÇO COMO COMPONENTE**

Primeiramente será analisado os conceitos de componente e biblioteca e suas diferenças. Componente é uma unidade de software que pode ser aprimorada e substituída de forma independente[9] e bibliotecas são componentes que estão ligados a um programa[10], geralmente disponibilizado por terceiros e carregadas na memória via funções de chamadas entre métodos.

Serviços podem ser classificados como componentes porque estão fora do processo principal e as comunicações entre eles são realizadas utilizando chamadas de procedimento remoto, ao contrário das bibliotecas que estão acoplados em processo único[9].

Arquiteturas microserviço utilizam bibliotecas, mas a sua principal forma de construir componentes é “quebrando” o seu próprio software em serviços[9].

Segundo Lewis e Fowler[9] uma das principais razões para a utilização de serviços como componente, em vez de bibliotecas em uma estrutura monolítica, é que os serviços são independentemente implementáveis. Como analisado anteriormente, cada pequena alteração em bibliotecas utilizadas em único processo(monolítico) todo a estrutura é reimplantada inteiramente, já em casos que a aplicação é decomposta em múltiplos serviços só será necessário reimplantar o serviço alterado. Toda a estrutura como suas dependências estão presentes dentro do próprio componente, isso garante o seu isolamento e diminui a quantidade de chamadas remotas para outros componentes, pois quase tudo que o serviço precisa estar contido nele.

Como dito anteriormente, em uma arquitetura de microserviço os componentes se comunicam através de chamadas remotas, muitas vezes implementados através de uma API[8]. Algumas dessas aplicações utilizam a arquitetura *REST (Representacional State Transfer)* como interface pública, esses serviços são fortemente baseados no protocolo HTTP. O HTTP é um protocolo padrão na web e foi desenvolvido de maneira a ser o mais flexível possível, por isso a sua interface de comunicação é clara. O protocolo é utilizado por várias linguagens e tecnologias, o que permite uma melhor integração.

No caso das chamadas remotas entre os componentes, o que é um benefício também pode se torna um enorme problema. Isso por conta que chamadas remotas são mais caras do que chamadas entre processos, e dependem de uma boa estrutura de rede para funcionar corretamente. Outro caso que também se torna uma desvantagem é que APIs remotas precisam ter uma granularidade alta, o que muitas vezes se torna complicado de usar[9]. Por isso desenvolvedores precisam ser muito cautelosos na construção de um serviço como um componente, como relatado por Newman[12] o domínio tem que estar bem definido, diminuindo o acoplamento e garantido o isolamento do componente. O isolamento físico é uma das principais forças de microserviços, é a chave para um serviço escalável com alta disponibilidade[10].

A seguir serão citados as algumas primitivas necessárias para o desenvolvimento de um microserviço isolado, visando diminuir os custos de conexões entre os componentes[10]:

* Eventos assíncronos devem fluir em tempo real em ambas as direções
* As requisições e as respostas podem fluir em qualquer direção,
* Formato de dados em mensagens devem ser bem conectáveis. Assim, os desenvolvedores podem usar, por exemplo, JSON[[8]](#footnote-8), XML[[9]](#footnote-9), etc.

Este trabalho pretende exemplificar através de estudo caso e implementação, a construção de um microserviço como componente.

**2.5 TECNOLGIAS**

**2.5.1 REST**

O surgimento da arquitetura REST foi idealizado por Roy Fielding, que é um dos principais autores do protocolo HTTP, e ele propôs em sua tese de doutorado um estilo de arquitetura que faz extenso uso dos recursos oferecidos por este protocolo. Enquanto nos serviços tradicionais que utilizam SOAP os recursos do HTTP são muito pouco explorados (inclusive porque o SOAP é independente de transporte), nos serviços REST umas das principais características é a utilização de muitos recursos do HTTP para elaborar um protocolo de comunicação conciso e claro[13].

Todo serviço REST é baseado nos chamados recursos, que são entidades bem definidas em sistemas, que possuem identificadores e endereços (URLs[[10]](#footnote-10)) próprios. Para interagir com as URLs, os métodos HTTP são utilizados. A regra de ouro para esta interação é que URLs são substantivos, e métodos HTTP são verbos. Isto quer dizer que os métodos HTTP são os responsáveis por provocar alterações nos recursos identificados pelas URLs. Estas modificações são padronizadas, de maneira que:

* + - GET – recupera os dados identificados pela URL;
    - POST – cria um novo recurso;
    - PUT – atualiza um recurso;
    - DELETE – apaga um recurso.

Esses quatro métodos principais podem ser diretamente relacionados a operações de bancos de dados. Assim, para recuperar o produto de número 1 do banco de dados, basta utilizarem o método GET em conjunto com a URL /produto/1; para criar um novo produto, basta utilizar o método POST sobre a URL /produto (o identificador será criado pelo banco de dados); para atualizar este produto, é utilizado o método PUT sobre a URL /produto/1 e, finalmente, para apagar o produto, o método DELETE é utilizado sobre a URL /produto/1[13]. Todas estas operações são lógicas. Isto quer dizer que utilizar o método DELETE, por exemplo, não significa necessariamente excluir o dado do banco de dados – significa apenas disponibilizar o recurso para consumo pelo método GET. Ou seja, DELETE pode apenas marcar o dado no banco de dados como desativado [13].

Um dos pilares de REST é o uso de *media types* para alterar as representações de um mesmo conteúdo, sob perspectivas distintas. *Media types* estão diretamente ligados aos conceitos de negociação de conteúdo. Através de uma mesma URl os serviços REST podem fornecer representações distintas de um mesmo recurso, podendo ser representado através de uma imagem, um registro no formato XML ou JSON. O cliente no momento da requisição solicita o formato através do atributo *accept* no cabeçalho HTTP[13].

**2.5.1 SPRING BOOT**

O SPRING-BOOT é um projeto criado pela PIVOTAL[15], empresa que criou e mantêm o projeto SPRING FRAMEWORK[[11]](#footnote-11). O BOOT é uma formidável ferramenta para construção de microserviços em Java, construindo aplicações SPRING prontas para produção. A framework é baseado no já conhecido SPRING e acumula todos os benefícios desta ferramenta, diminuindo a complexidade das estruturas de configuração e permitindo agilidade na construção de microserviços. A ferramenta utiliza o conceito de convenção sobre configuração, conceito presente em outros frameworks como RAILS[[12]](#footnote-12) e GRAILS[[13]](#footnote-13)[15]. Muito do SPRING-BOOT visa a produtiva do desenvolvedor através de seus princípios.

Os princípios do SPRING-BOOT são[15]:

1. Prover uma experiência de início de projeto (*getting started experience*);
2. Apresentar uma visão bastante opinativa (opinionated) sobre o modo como devemos configurar nossos projetos SPRING, mas ao mesmo tempo flexível o suficiente para que possa ser facilmente substituída de acordo com os requisitos do projeto;
3. Fornece uma série de requisitos não funcionais já pré-configurados para o desenvolvedor como, por exemplo, métricas, segurança, acesso a base de dados, servidor de aplicações/servlet embarcado, etc.;
4. Não prover nenhuma geração de código e minimizar a zero a necessidade de arquivos XML.

O SPRING-BOOT pode ser utilizado para criar aplicações JAVA, como arquivo jar ou implantações de componentes WAR mas tradicionais. Essa ferramenta também possibilita a utilização de linha de comando para execução de scripts “spring”[15].

O BOOT também é descrito como um micro framework e seu funcionamento é descrito da seguinte forma[15]:

* O BOOT é construído em módulos agregados, conhecidos como "starters". Estes módulos iniciais são composições de bibliotecas que interagem entre si e podem ser usados para fornecer funcionalidade de um aplicativo.
* Essas bibliotecas também são a estrutura pela qual BOOT é capaz de expor suas “opiniões” em torno da configuração do aplicativo, isso facilita a convenção sobre configuração durante os ciclos do desenvolvimento, evitando a utilização de arquivos de configuração.

**2.5.2 DOCKER**

O Docker é uma plataforma de código aberto e desenvolvida pela dotCloud[16], utilizada por desenvolvedores e administradores de sistemas para construir, embarga e executar aplicações distribuídas. Essa plataforma é composta pelo Docker *Engine*, uma ferramenta de tempo de execução que empacota ferramentas leves e portáteis, e o Docker Hub, um serviço de nuvem para o compartilhamento de aplicativos e automatização de fluxos de trabalho[17].

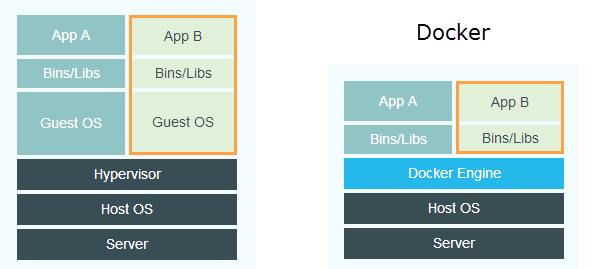
O Docker alcança sua robustez de conter a aplicação (e, portanto, de processos e recursos) via Contêineres Linux (por exemplo, *namespaces* e outras características do *kernel*) [17]. Seus novos recursos vêm de componentes e partes próprias do projeto, que extraem toda a complexidade de trabalhar com ferramentas/APIs Linux de baixo nível usadas para o sistema e para o gerenciamento de aplicações, no que diz respeito a conter os processos com segurança. O Docker permite que aplicativos sejam montados rapidamente a partir de componentes e elimina o atrito entre os ambientes de desenvolvimento, controle de qualidade e de produção. Como resultado, a TI pode enviar mais rápido e executar o mesmo aplicativo, sem alterações, em laptops ou centro de dados implantados em maquinas virtuais[17].

Os seguintes elementos são usados pelas aplicações que formam o projeto Docker[17]:

* Contêineres docker: São basicamente, diretórios que podem ser empacotados (agrupados como .tar por exemplo) como qualquer outro, e então, compartilhados e executados entre várias máquinas e plataformas (hosts). A única dependência é ter os hosts ajustados para executar os contêineres (ou seja, ter o docker instalado). A contenção aqui é obtida através de Contêineres Linux (LXC);
* Imagens docker: As imagens docker constituem a base para os contêineres docker de onde tudo começa a se formar. Elas são muito similares às imagens de disco padrão de sistema operacional que são utilizadas para executar aplicações em servidores e computadores de mesa;
* Dockerfiles: São scripts contendo uma série sucessiva de instruções, orientações e comandos que devem ser executados para formar uma nova imagem docker. Cada comando executado traduz-se para uma nova camada, formando o produto final. Elas basicamente substituem o processo de se fazer tudo manualmente e repetidamente. Quando um Dockerfile conclui a execução, você acaba tendo formado uma imagem, que então, você utiliza para iniciar (ou seja, criar) um novo contêiner.

Em comparação as maquinas virtuais o Docker compreende apenas o aplicativo e suas dependências. Ele é executado como um processo isolado no espaço do usuário no sistema operacional hospedeiro, compartilhando o *kernel* com outros recipientes. Em VM´s o aplicativo virtualizado inclui não apenas a aplicação, binários e bibliotecas necessárias, mas também um sistema operacional convidado inteiro[9]. Assim o Docker usufrui de isolamento, alocação e de benefícios e recursos das VMs, mas é muito mais portátil e eficiente[16]. A Figura 3 mostra um comparativo entre máquina virtual e o Docker.

Figura(3)



Fonte[16]

Segundo Uri Cohen[18], um dos líderes de produtos na *GigaSpaces*, descreve o Docker uma excelente opção para microserviços pelo fato de isolar contêineres para um processo independente. Essa utilização intencional de contêineres para serviços ou processos individuais, faz com que seja muito simples gerenciar e atualizar esses serviços.

**2.5.3 MONGODB**

MongoDB é um banco de dados de documentos de código aberto que oferece alta performance, alta disponibilidade e escala automática[19].

Um registro em MongoDB é um documento, que por sua vez é uma estrutura de dados composta por pares de campo e de valor. Documentos MongoDB são semelhantes aos objetos JSON e esses documentos são agrupados em *collections*. Os valores dos campos podem incluir outros documentos, matrizes e matrizes de documentos[19].

Principais características:

1. Alta performance;

* Suporte para modelos de dados incorporados reduz a atividade de E / S no sistema de banco de dados.
* Utiliza Índices onde para consultas. O que torna mais rápido as consultas

2. Alta disponibilidade:

* Facilidade de replicação e particionamento.

3. Escalabilidade automatizada:

* MongoDB oferece escalabilidade horizontal.

O MongoDB estar no conjunto dos bancos de dados não-relacionais e que não utilizam linguagem SQL para operações, os chamados NoSql. A utilização desse tipo de banco em um projeto de microserviço serve para experimentar a heterogeneidade destes. Um dos princípios da arquitetura de microserviço é a autonomia, isso significa que o serviço tem controle total de todos os componentes, permitindo a ele utilizar vários tipos de bancos de dados de diferentes estruturas. Um ponto importante é que essa característica permite ao time de desenvolvedores a escolha certa de tecnologias para o projeto, e não os limita a utilização dos tradicionais bancos de dados relacionais.

**2.6 TRABALHOS RELACIONADOS**

De acordo com os autores[9], não existe uma definição precisa para esse estilo arquitetônico, eles descrevem que o termo “Arquitetura de Microserviço” surgiu nos últimos anos para descrever uma maneira particular de conceber aplicações de software como suítes de serviços independentemente e implementáveis. Tendo visto muitos projetos utilizando este estilo nos últimos anos, os autores relatam que os resultados têm sido positivos, e em alguns casos se tornaram o estilo padrão para construção para aplicações corporativas. Estes serviços são construídos em torno de capacidades de negócios e implementáveis independentemente por máquinas de implantação totalmente automatizados. Ainda segundo os autores, uma das formas para explicar microserviço é compará-lo com o estilo monolítico. Eles descrevem que aplicações monolíticas podem ser bem sucedidas, mas que as pessoas estão sentindo frustradas com elas, especialmente à medida que mais aplicativos estão sendo implantados para a nuvem. Uma alteração feita uma pequena parte do aplicativo, requer que toda a estrutura seja reconstruída e implantada novamente. Ao longo do tempo é difícil manter uma boa estrutura modular, tornando-se mais difícil as manutenções e alterações. Estas dificuldades levaram à arquitetura de microserviço. Neste trabalho são destacadas as características que uma arquitetura de microserviço deve possuir: Componentes via Serviços; Organizado segundo as capacidades de negócios e governança descentralizada.

Os autores em [10], introduzem a arquitetura de microserviço como um termo relativamente novo no padrão de arquitetura de software. Conceituam a abordagem como uma plataforma para o desenvolvimento de um aplicativo como um conjunto de pequenos serviços independentes. Sendo cada serviço executado em seu próprio processo, podendo ser desenvolvidos, gerenciados e implantados absolutamente separados um dos outros. Além disso, podem ser construídos em diferentes linguagens de programação, usando modelos próprios de dados e se comunicando através de mecanismos leves que utilizam o protocolo HTTP. Os autores apostam na arquitetura de microserviços como uma abordagem a solucionar problemas encontrados em estruturas monolíticas. Eles também apresentam os desafios da arquitetura como o gerenciamento de múltiplos serviços distribuídos, testes e a comunicação. Além desses desafios a abordagem também leva ao aumento do consumo de memória, devido ao próprio espaço de endereçamento para os serviços. Um dos maiores desafios é decidir como dividir (partição) o sistema em microserviços. O autor descreve pelo menos duas estratégias de divisão do aplicativo em microserviços: Particionamento através de verbos, por exemplo, o serviço implementa o subsistema de login, subsistema de backup e etc. e segunda é o particionamento do sistema por substantivos ou recursos. Em[3], são apresentadas formas de comunicação entre microserviço e discutido os desafios relacionados as chamadas remotas entre eles. A primeira é que cada serviço pode se comunicar com outros serviços diretamente. Certamente essa é a forma mais flexível, porem o maior problema é o atraso nas chamadas. Portanto, o próximo passo é diminuir a quantidade de chamadas remotas. Através de várias formas de cache, monitores de transação em bases de dados e uma aplicação *middleware.*

Outra solução é a utilização de algum *service-bus,* Essa solução utiliza um ponto central de orquestração de chamadas, o que ajuda principalmente em chamadas assíncronas

De acordo com [20], o autor descreve o estilo arquitetônico como um único aplicativo dividido em pequenas aplicações individuais que se comunicam através de uma rede, e que se estilo vem se tornando popular em aplicações corporativas. Cada aplicação tem seu próprio ciclo de desenvolvimento. Esta dissociação permite que pequenas equipes trabalharem em aplicações individuais. Tendo em vista que uma arquitetura de microserviço consiste em múltiplos serviços independentes, estes também podem falhar individualmente. Isso leva às perguntas de como essas falhas individuais se manifestam e como elas influenciam em outras aplicações. Como falhas são propagadas através dos sistemas? Como eles influenciam a experiência do usuário final? O autor acredita que a compreensão e a modelagem das dependências entre aplicações são parte cruciais para compreender a confiabilidade de uma arquitetura microserviço. O autor propõe o conceito de gráficos de dependência para modelar as dependências entre aplicações em uma arquitetura microserviço.

1. **METODOLOGIA**

Objetivando o estudo da arquitetura de microserviços como solução para o desenvolvimento de um serviço, visando como funcionalidade o cadastramento e controle de ocorrências no contexto de segurança pública e defesa civil, a metodologia utilizada nesse trabalho pode ser descrita nos seguintes tópicos:

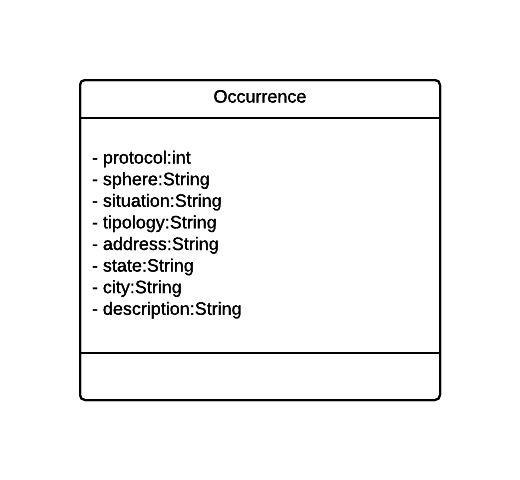
**3.1 ESTUDOS REALIZADOS**

* Estudos bibliográficos e pesquisa de artigos sobre o desenvolvimento de aplicações no contexto de segurança pública, foram elencados a soluções já utilizadas pelo governo federal como o SINESP e S2ID.
* Por se tratarem de aplicações governamentais, cada aplicação é regulamentada por lei (Princípio da legalidade). Foram feitos estudos sobre a lei 12.681 que regulamenta o SINESP, objetivando uma base para a aplicação a ser desenvolvida.
* Estudos sobre os padrões de interoperabilidade de sistemas governamentais(ePING).
* Estudo bibliográfico e artigos relacionados a arquitetura de microserviços, objetivando o estudo dos seus conceitos, tecnologias e elencar as suas vantagens e desvantagens em relação as tradicionais estruturas monolíticas.
* Estudos sobre tecnologias como SPRING-BOOT, MONGODB e DOCKER que possibilitaram a construção do serviço seguindo os conceitos de microserviços.
* Pesquisa sobre trabalhos relacionados sobre a abordagem de microserviços.
* Estudos para que fossem levantados os requisitos necessários para o desenvolvimento de um serviço para o controle ocorrências para defesa civil.

**3.2 DEFINIÇÕES DO DOMINIO E INTEGRAÇÃO PARA O MICROSERVIÇO**

De acordo com Lewis e Fowler[9], é preciso evitar o modelo de projeto em que visa entregas de pedaços de software, em vez disso na construção de microserviço é necessário desenvolver um aplicativo como um produto único. Newmam[8] também define que um serviço precisa ter um domínio bem definido para o seu desenvolvimento. Seguindo as orientações dos autores citados, ficou definido como principal produto e domínio do serviço, as ocorrências. Como forma de representação gráfica da entidade foi utilizado como recurso o diagrama de classe da UML 2.0(figura 4).

Figura(4)



Fonte: autoria própria

Após a definição do domínio, foi definido a forma integração do serviço. Ainda de acordo com [8], é importante evitar utilizar tecnologias de integração que restrinjam a pilha de protocolos específicos, como por exemplo o SOAP. É necessário facilitar o consumo do serviço diminuindo o acoplamento. Os autores citados nesse tópico concordam que uma das melhores formas de integração são os serviços REST. Outro fato que torna o REST uma ótima opção é fato do conceito de recursos e a flexibilidade quanto à forma de consumo dos dados, já que essa abordagem permite usar uma variedade de formatos, como XML e JSON.

Seguindo o padrão de interoperabilidade de sistemas governamentais(ePING) a Tabela 1 representa todas as especificações técnicas do microserviço:

Tabela 1 Especificações técnicas

|  |  |
| --- | --- |
| Especificações técnicas | |
| *Web Service* | REST |
| Protocolo para acesso a *Web Service* | HTTP/1.1 (RFC 2616) |
| Transporte | TCP |
| Linguagem para intercâmbio de dados | JSON (*Javascript Object Notation*) Como definido pela IETC <http://www.ietf.org/rfc/rfc4627.txt> |

Seguindo os conceitos da arquitetura REST*,* Aa Tabela 2 especifica todas as rotas para o consumo de serviço implementados dentro da aplicação. Afim de facilitar a integração com outras plataformas:

Tabela 2 Rotas para consumo de recurso ocorrências

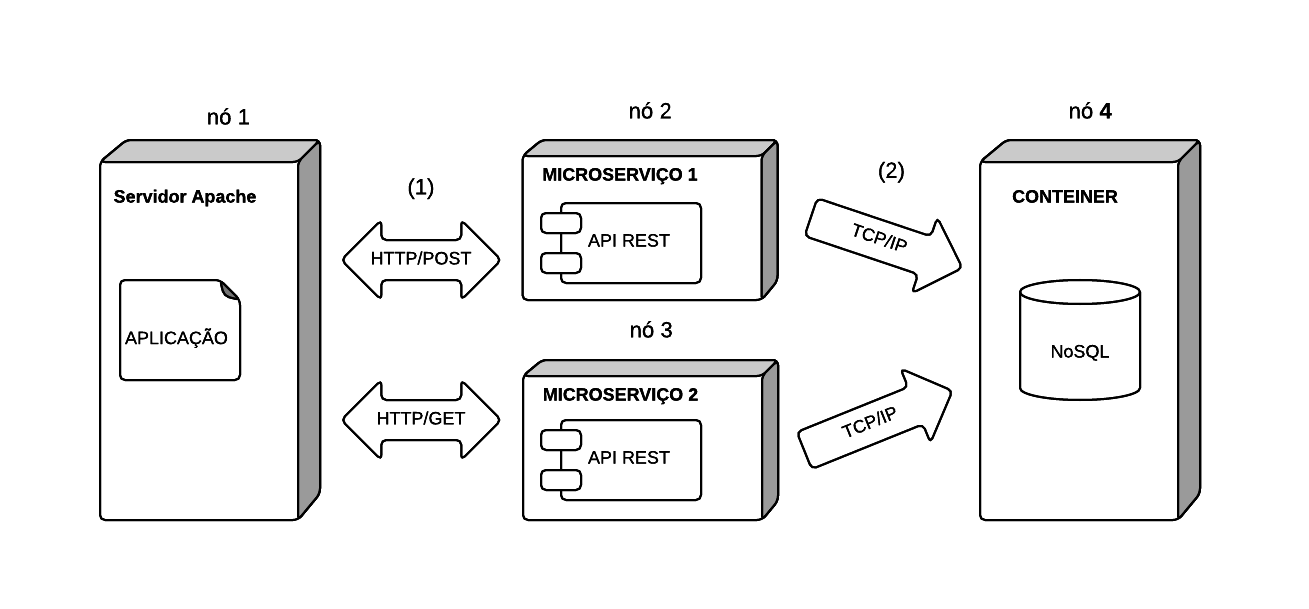
|  |  |  |
| --- | --- | --- |
| Rotas para consumo de recurso ocorrências | | |
| Ação | URL | VERBO HTTP |
| Adicionar ocorrência | /occurrences | POST |
| Consultar ocorrências | /occurrences | GET |

* 1. **VISÃO GERAL DA ARQUITETURA**

No contexto geral da aplicação foram utilizadas duas arquiteturas, uma monolítica e outra microserviço. A estrutura monolítica representa o cliente que irá consumir dados do serviço. A estrutura de microserviço é responsável pelo controle de requisições e armazenamento das informações no banco de dados. A arquitetura geral está representada na figura 5.

* O primeiro nó 1 representa um servidor web contendo uma aplicação monolítica, sendo utilizada como interface para os usuários. A aplicação contida neste nó se comunica com duas instâncias do microserviço através de requisições HTTP.
* Os nós 2 e 3 representam duas instâncias do microserviço como um componente autocontido e independente implementados como contêiner gerenciados pelo docker, sendo os dois executados em processos separados e com endereço de rede distintos. Estes componentes utilizam uma API implementada sobre a arquitetura RESTpara comunicação e transferência de dados entre os nós. Sendo responsável por toda logica de negócio, controle e estrutura de dados do recurso ocorrência.
* O nó 4 representa o banco de dados contido em um contêiner.

Figura(5)



Fonte: autoria própria

**3.3 PROVA DE CONCEITO**

**3.3.1 DETALHES DE IMPLENTAÇÃO: MICROSERVIÇO TAIGA E O MONGODB**

A implementação do microserviço foi chamado de taiga. O nome taiga faz referência ao ecossistema encontrado no norte do Alasca, Canadá, sul da Groenlândia, parte da Noruega, Suécia, Finlândia, Sibéria e Japão [21]. O microserviço foi chamado assim, porque a sua definição se assemelha ao conceito de ecossistema. O ecossistema é um conjunto formado por todas as comunidades bióticas que vivem e interagem em determinada região[22], e os microserviços são conjunto de componentes independentes, com ciclo de vida próprio que interagem entre si.

O taiga é o principal componente da arquitetura. Ele é o serviço responsável pela lógica de negócio, controle e estrutura de dados das ocorrências, além da inserção e busca no banco de dados. Esse componente é composto por duas estruturas, a primeira é o serviço em si construído sobre uma arquitetura REST e implementado pelo *framework* SPRING BOOT. A segunda estrutura trata-se de um contêiner gerenciado pelo DOCKER, sendo a primeira contida na segunda estrutura.

O taiga se comunica com outro contêiner DOCKER, estes implenta o banco de dados MONGODB.

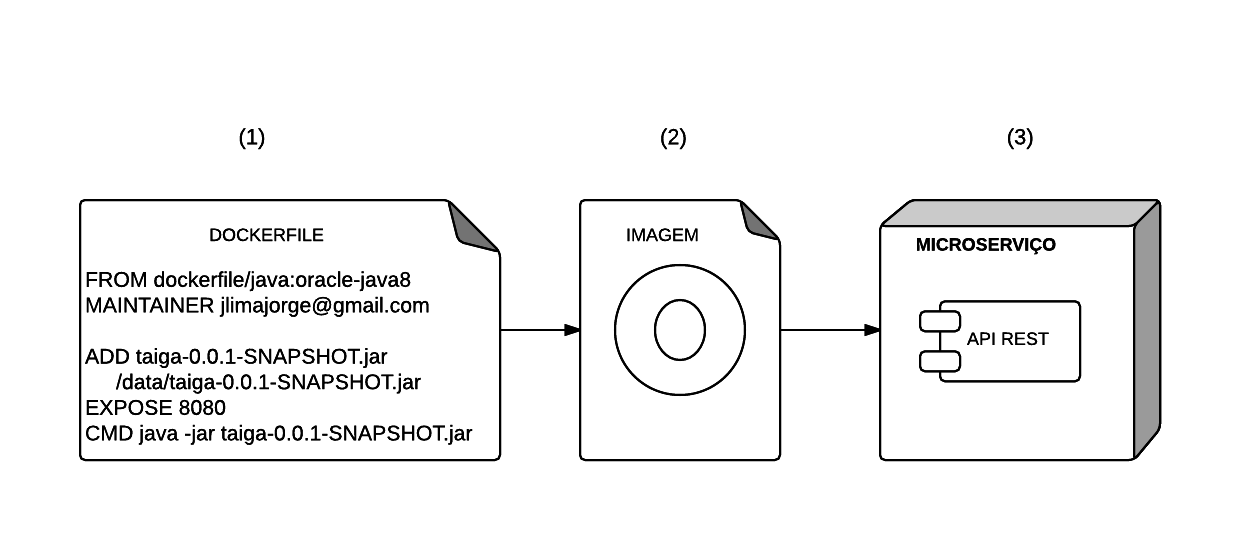
Os seguintes passos foram seguidos para a implementação;

* O SPRING BOOT foi utilizado na construção da API baseada na arquitetura REST*,* toda a estrutura e dependências, inclusive o contêiner *web(tomcat8)*, foram compilados em um arquivo jar.
* O Docker através de sua *engine* foi utilizado para criação e gerenciamento dos componentes como contêineres, provendo uma interface de rede e as dependências necessárias:

Passos para criação do microserviço (Figura 6):

1. Criação do *DockerFile* contendo como base a imagem da jdk8, comandos para manipulação do arquivo jar e execução do serviço.
2. A partir do *DockerFile* foi gerado a imagem, com todos os comandos compilados.
3. A partir da imagem foram criados os contêineres, como o serviço em execução, cada instância do contêiner possui um endereço de rede próprio que é mapeado pelo docker para sua interface de rede.

Figura(6)



Fonte: autoria própria

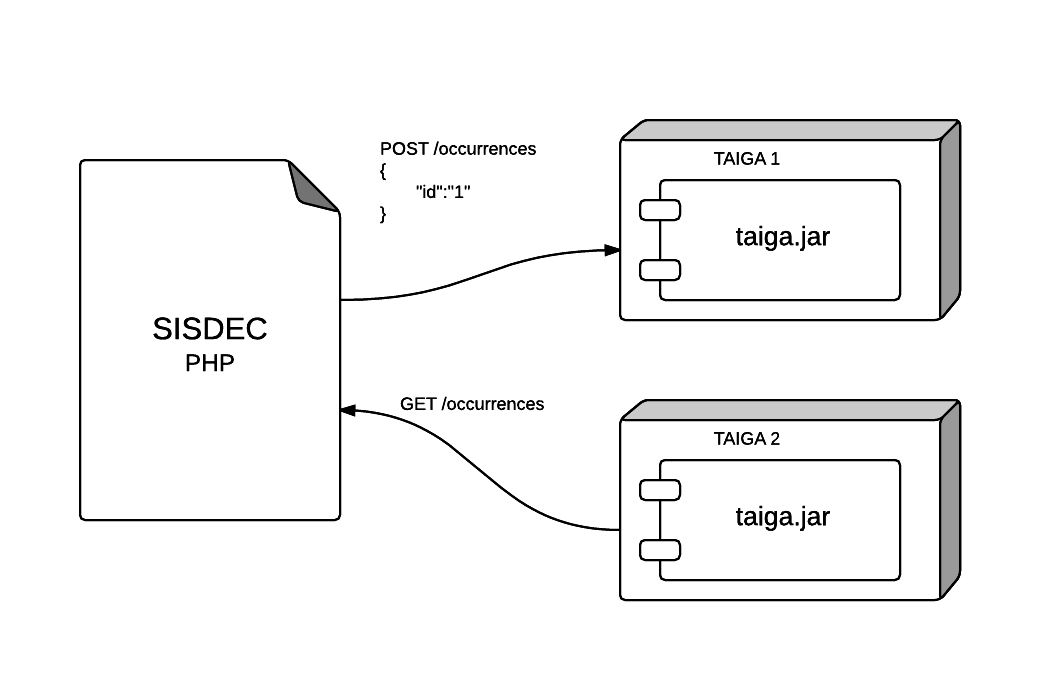
* Implementação do MongoDB como contêiner docker.
* Configuração das comunicações entre os contêineres seguindo as especificações técnicas definidas anteriormente.
* Os requisitos para tratamento e transferência de dados e os tipos de protocolos.

O objetivo de usar duas instâncias do taiga é demonstrar os benefícios da abordagem de microserviços, como: Isolamento de falhas, escalabilidade, alto desempenho e a transparência. Atuando como verdadeiro sistema distribuído.

**3.3.3 A APLICAÇÃO CLIENTE**

Para exemplificação de integração entre plataforma e microserviço, foi utilizado uma aplicação *web* implementa na linguagem PHP[[14]](#footnote-14), consumindo serviços das instâncias do taiga. Essa aplicação foi identificada com o nome SISDEC.O SISDEC está sendo representada por um sistema web executado dentro de um servidor apache e possui como funcionalidade o cadastro e login de usuários, cadastro e visualização de ocorrências. As telas de cadastro e visualização de ocorrências estão sendo utilizadas apenas como interface com usuário, toda a lógica de negócio estar contigo no taiga. Após o cadastramento de uma ocorrência o SISDEC faz uma requisição HTTP repassando os dados em formato JSON para o primeiro microserviço, de forma análoga a aplicação faz a requisição para o segundo microserviço para consultar as ocorrências (figura 7).

Figura(7)



Fonte: autoria própria

**3.1.2 CONSIDERAÇÕES E RESULTADOS OBTIDOS**

Após as implementações, o resultado obtido foi um microserviço como componente, auto gerenciável, escalável, independente e transparente, seguindo todos os conceitos e benefícios da abordagem de microserviços. O microserviços taiga são contêineres gerenciados pelo docker, que possuem todas as dependências necessárias para execução do serviço, essa característica torna fácil a implantação em maquinas servidores, já que essas maquinas necessitam apenas da *engine* do DOCKER para executa-lo.

Também foi obtido com êxito o principal objetivo do taiga, que é servi como ponto de integração entre diversos sistemas de controle de ocorrência, embora o exemplo utilizado foi o controle de ocorrência da defesa civil, essa estrutura também poderia ser utilizada em qualquer tipo de ocorrências no âmbito de segurança pública. Toda a estrutura do taiga é baseada em especificações de sistemas governamentais, objetivando uma melhor integração e regulamentação do seu uso.

Outro ponto importante alcançado foi a portabilidade das informações para diferentes tipos de plataformas, o taiga se integra facilmente a sistemas *mobile* e *web*, pois utiliza um protocolo comum utilizados nessas aplicações, e a forma de transferência de dados é leve e portável.

1. **CONCLUSÃO E TRABALHOS FUTUROS**

Este artigo buscou validar os benefícios propostos pela arquitetura de microserviços, buscando a construção de um aplicativo que funcione como serviço, e que tenha como principal objetivo ajudar os órgãos governamentais na gestão de políticas públicas, provendo um serviço que seja escalável, de fácil integração, alto desempenho e permita o armazenamento de várias informações, que servirão como indicadores para mensurar a qualidade das políticas em várias esferas sejam elas federal, estadual ou municipal

A proposta do serviço taiga é demonstrar que a abordagem de microserviço, apesar de nova, veio para solucionar os problemas de alto acoplamento e dificuldades de manutenção em encontradas em algumas arquiteturas tradicionais.

Para trabalhos futuros, pretende-se realizar a validação e aplicabilidade do aplicativo para ser utilizados como ponto de integração em sistemas governamentais, buscando informações sobre o nível de desempenho que um serviço construído em microserviço possa obter, verificando se os objetivos esperados do mesmo são alcançados. E com isso vários órgãos públicos possam agilizar as suas políticas de segurança e defesa social, no intuito de melhorar os seus serviços prestados aos cidadãos.

Outro trabalho futuro possível seria um estudo mais aprofundado da abordagem de microserviços, com objetivo de obter cada vez mais excelência na construção dessas aplicações.

**5. REFERÊNCIAS**

[1]. SENASP/ANP. (Org.). Violência, Criminalidade e Prevenção - VA. 2015. Disponível em: <http://ead.senasp.gov.br/>. Acesso em: 2 abril 2015.

[2]. CÃMARA NOTICIAS (BRASILIA). CÂMARA DOS DEPUTADOS. Sistema de informações sobre segurança é um desastre, diz ministro da Justiça. Disponível em: <http://www2.camara.leg.br/camaranoticias/noticias/SEGURANCA/197339-SISTEMA-DE-INFORMACOES-SOBRE-SEGURANCA-E-UM-DESASTRE,-DIZ-MINISTRO-DA-JUSTICA.html>. Acesso em: 27 de Outubro de 2014.

[3]. SENASP/ANP. (Org.). Uso da Informação em Gestão de Segurança Pública - VA. 2015. Disponível em: <http://ead.senasp.gov.br/>. Acesso em: 2 de abril 2015.

[4]. RAND CORPORATION (Org.). OBJECTIVE ANALYSIS. EFFECTIVE SOLUTIONS. Disponível em: <http://www.rand.org/>. Acesso em: 2 abr. 2015.

[5]. SINESP – Sistema Nacional de informações de segurança pública. Disponível em: https://www.sinesp.gov.br/estatisticas-publicas. Acesso em 05 de maio de 2015.

[6]. S2ID - Sistema Integrado de Informações sobre Desastres. Disponível em: http://www.mi.gov.br/defesa-civil/s2id. Acesso em 05 de maio de 2015.

[7]. Padrões de Interoperabilidade de Governo Eletrônico-ePing. Disponível em: <http://www.governoeletronico.gov.br/acoes-e-projetos/e-ping-padroes-de-interoperabilidade>. Acesso em 05 de maio de 2015.

[8]. NEWMAN, Sam. Building Microservices. Sebastopol: O'Reilly, 2015.

[9]. LEWIS James, FOWLER Martin. Microservices. Disponível em: <http://martinfowler.com/articles/microservices.html#ComponentizationViaServices>. Acesso em 4 de abril de 2015.

[10]. NAMIOT Dmitry; SNEPPE Manfred. On Micro-services Architecture. International Journal of Open Information Technologies, 2(9), 2014.

[11]. RICHARDSON Chis. Microservices: Decomposição de Aplicações para Implantação e Escalabilidade. Disponível em: http://www.infoq.com/br/articles/microservices-intro. Acesso em 5 de abril de 2015.

[12]. NEWMAN, Sam. Microservices For Greenfield?. Disponível em: <http://samnewman.io/blog/2015/04/07/microservices-for-greenfield/>. Acesso em 18 de abril de 2015.

[13]. SAUDATE, Alexandre. REST: Construa API’s inteligentes de maneira simples.São Paulo: Casa do código, 2015.

[14]. UHLE Johan. On Dependability Modeling in a Deployed Microservice Architecture. Hasso Plattner Institut, Universitat Potsdam, 2014.

[15]. PIVOTAL SOFTWARE INC. Disponível em: http://docs.spring.io/spring-boot/docs/1.3.0.BUILD-SNAPSHOT/reference/htmlsingle/. Acesso em 5 de abril de 2015.

[16]. DOCKER INC. Disponível em: https://www.docker.com/whatisdocker/. Acesso em 5 de abril de 2015.

[17]. O.S.Tezer. How To Install and Use Docker: Getting Started. Disponível em:< https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-getting-started>. Acesso em 5 de abril de 2015.

[18]. COHEN Uri. Containers, microservices, and orchestrating the whole symphony. Disponível em:<http://opensource.com/business/14/12/containers-microservices-and-orchestrating-whole-symphony/>. Acesso em 5 de abril de 2015.

[19]. MONGODB INC (New York) (Org.).Introduction to MongoDB.Disponível em: http://docs.mongodb.org/manual/core/introduction/. Acesso em: 21 abr. 2015.

[20]. VILLAMOR José Inácio, IGLESIAS Carlos, GARIJO Mercedes. A VOCABULARY FOR THE MODELLING OF IMAGE SEARCH MICROSERVICES. Departamento de Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid.

[21]. CONTEÚDO aberto. In: Wikipédia: a enciclopédia livre. Disponível em: < <http://pt.wikipedia.org/wiki/Wikip%C3%A9dia:Citando_a_Wikip%C3%A9dia/>> Acesso em: 21 abr. 2015.

[22]. CONTEÚDO aberto. In: Wikipédia: a enciclopédia livre. Disponível em: <http://pt.wikipedia.org/wiki/Ecossistema/> Acesso em: 21 abr. 2015.

1. Empresa norte americana de tv por internet. [↑](#footnote-ref-1)
2. Plataforma online de publicação de áudio. [↑](#footnote-ref-2)
3. Rede social para microblogging. [↑](#footnote-ref-3)
4. Plataforma de programação para servidores na linguagem de programação Java [↑](#footnote-ref-4)
5. Protocolo de comunicação entre sistemas de informação que permite a transferência de dados entre redes de computadores [↑](#footnote-ref-5)
6. Linguagem de marcação utilizada para produzir páginas na Web [↑](#footnote-ref-6)
7. Empresa multinacional de comércio electrónico dos Estados Unidos com sede em Seattle. [↑](#footnote-ref-7)
8. Notação de Objetos JavaScript. [↑](#footnote-ref-8)
9. Linguagem de marcação extensível. [↑](#footnote-ref-9)
10. Sigla que se refere ao endereço de rede no qual se encontra algum recurso informático. [↑](#footnote-ref-10)
11. *Framework open source* para a plataforma Java. [↑](#footnote-ref-11)
12. Projeto de código aberto escrito na linguagem de programação Ruby. [↑](#footnote-ref-12)
13. *Framework* para construção de aplicações para web através da linguagem de programação Groovy. [↑](#footnote-ref-13)
14. Linguagem de programação [↑](#footnote-ref-14)